Administrator's Advantage Series

Unix/Linux Survival Guide

- Teaches Unix/Linux system administrators what they need to know to avoid system breakdowns
- Explains a variety of tasks that are vital for an administrator, including programming small scripts, user administration, documentation, installation, setting up a test system, preventative maintenance techniques, and data security
- Includes a companion CD-ROM with third-party software used in the book and all of the code and figures

Erik Keller

CHARLES RIVER MEDIA

Unix/Linux Survival Guide

LIMITED WARRANTY AND DISCLAIMER OF LIABILITY

THE CD-ROM THAT ACCOMPANIES THE BOOK MAY BE USED ON A SINGLE PC ONLY. THE LICENSE DOES NOT PERMIT THE USE ON A NETWORK (OF ANY KIND). YOU FURTHER AGREE THAT THIS LICENSE GRANTS PERMISSION TO USE THE PRODUCTS CONTAINED HEREIN, BUT DOES NOT GIVE YOU RIGHT OF OWNERSHIP TO ANY OF THE CONTENT OR PRODUCT CONTAINED ON THIS CD-ROM. USE OF THIRD-PARTY SOFTWARE CONTAINED ON THIS CD-ROM IS LIMITED TO AND SUBJECT TO LICENSING TERMS FOR THE RESPECTIVE PRODUCTS.

CHARLES RIVER MEDIA, INC. ("CRM") AND/OR ANYONE WHO HAS BEEN INVOLVED IN THE WRITING, CREATION, OR PRODUCTION OF THE ACCOMPANYING CODE ("THE SOFTWARE") OR THE THIRD-PARTY PRODUCTS CONTAINED ON THE CD-ROM OR TEXTUAL MATERIAL IN THE BOOK, CANNOT AND DO NOT WARRANT THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THE SOFTWARE OR CONTENTS OF THE BOOK. THE AUTHOR AND PUBLISHER HAVE USED THEIR BEST EFFORTS TO ENSURE THEACCURACY AND FUNCTIONALITY OF THE TEXTUAL MATERIAL AND PROGRAMS CONTAINED HEREIN. WE HOWEVER, MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, REGARDING THE PERFORMANCE OF THESE PROGRAMS OR CONTENTS. THE SOFTWARE IS SOLD "AS IS" WITHOUT WARRANTY (EXCEPT FOR DEFECTIVE MATERIALS USED IN MANUFACTURING THE DISK OR DUE TO FAULTY WORKMANSHIP).

THE AUTHOR, THE PUBLISHER, DEVELOPERS OF THIRD-PARTY SOFTWARE, AND ANYONE INVOLVED IN THE PRODUCTION AND MANUFACTURING OF THIS WORK SHALL NOT BE LIABLE FOR DAMAGES OF ANY KIND ARISING OUT OF THE USE OF (OR THE INABILITY TO USE) THE PROGRAMS, SOURCE CODE, OR TEXTUAL MATERIAL CONTAINED IN THIS PUBLICATION. THIS INCLUDES, BUT IS NOT LIMITED TO, LOSS OF REVENUE OR PROFIT, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THE PRODUCT.

THE SOLE REMEDY IN THE EVENT OF A CLAIM OF ANY KIND IS EXPRESSLY LIMITED TO REPLACEMENT OF THE BOOK AND/OR CD-ROM, AND ONLY AT THE DISCRETION OF CRM.

THE USE OF "IMPLIED WARRANTY" AND CERTAIN "EXCLUSIONS" VARIES FROM STATE TO STATE, AND MAY NOT APPLY TO THE PURCHASER OF THIS PRODUCT.

UNIX/LINUX SURVIVAL GUIDE

ERIK M. KELLER

CHARLES RIVER MEDIA, INC.
Hingham, Massachusetts

Copyright 2006 by CHARLES RIVER MEDIA, INC.
All rights reserved.

Copyright © 2004 by Pearson Education Deutschland GmbH. All rights reserved.
First published in the German language under the title “Unix/Linux Survival Guide” by Addison-Wesley an imprint of Pearson Education Deutschland GmbH, München.

No part of this publication may be reproduced in any way, stored in a retrieval system of any type, or transmitted by any means or media, electronic or mechanical, including, but not limited to, photocopy, recording, or scanning, without *prior permission in writing* from the publisher.

Acquisitions Editor: James Walsh
Cover Design: Tyler Creative

CHARLES RIVER MEDIA, INC.
10 Downer Avenue
Hingham, Massachusetts 02043
781-740-0400
781-740-8816 (FAX)
info@charlesriver.com
www.charlesriver.com

This book is printed on acid-free paper.

Erik M. Keller. *Unix/Linux Survival Guide.*

ISBN: 1-58450-433-1
eISBN: 1-58450-656-3

All brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks should not be regarded as intent to infringe on the property of others. The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products.

Library of Congress Cataloging-in-Publication Data
Keller, Erik M.
Unix/Linux survival guide / Erik M. Keller.-- 1st ed.
p. cm.
Translation of Unix/Linux survival guide (text in German)
Includes index.
ISBN 1-58450-433-1 (pbk. with cd-rom : alk. paper)
1. UNIX (Computer file) 2. Linux. 3. Operating systems (Computers) I. Title.
QA76.76.O63.K449 2005
005.4'32--dc22
2005013583

Printed in the United States of America
05 7 6 5 4 3 2 First Edition

CHARLES RIVER MEDIA titles are available for site license or bulk purchase by institutions, user groups, corporations, etc. For additional information, please contact the Special Sales Department at 781-740-0400.

Requests for replacement of a defective CD-ROM must be accompanied by the original disc, your mailing address, telephone number, date of purchase, and purchase price. Please state the nature of the problem, and send the information to CHARLES RIVER MEDIA, INC., 10 Downer Avenue, Hingham, Massachusetts 02043. CRM’s sole obligation to the purchaser is to replace the disc, based on defective materials or faulty workmanship, but not on the operation or functionality of the product.

Contents

This page intentionally left blank

Foreword

This book is about the administration of *NIX systems. There is no real "right" or "wrong" approach in this respect. You usually have more than one way to achieve a certain goal.

**NIX administration is a creative process.*

Sound too good to be true? One of the goals of this book is to prove that this sentence is true. If you view a *NIX-based system as a box of wonderful tools providing almost limitless possibilities to create solutions, the meaning of the sentence starts to make some sense. Add your own creativity into the mix, and you are getting even closer to the real meaning of that sentence.

More often than not I have been astonished with how well structured and logical *NIX-based systems are. Not that I didn't know about this before, but after 18 years one takes a lot of things for granted and tends to forget what it is like the first time on a *NIX system, the countless things to investigate and explore. To make a long story short, while writing this book, I tried to see my favorite OS with the eyes of a beginner. I tried to forget experience and instinct and tried instead to make sense of manpages last viewed on Solaris 2.5.1 almost eight years ago. I did what I recommend in this book.

I have been using pretty much everything that resembles a *NIX system during the last 18 years. One important thing I learned during these years was the fact that even seemingly different operating systems like Solaris and AIX have more in common than you might think. The book describes techniques that are common to almost all flavors of *NIX available, and most of them will work on BSD-based systems like OS X as well. There might be some slight differences like the location of a file, but the contents of that file are used for the same purpose on every system.

If you understand what the entries in a configuration file are for, adapting this knowledge to a different brand of *NIX is not that much of a problem. As long as you keep a couple of things in the back of your head, it is possible to administer more than a couple of *NIX brands at a given time. My personal “record” was the administration of seven different brands of *NIX in a plethora of different releases. The same engine is “under the hood,” and the fact that some switches are in different places is pretty much the only difference you have to deal with. The same way you approach using an unknown make of a car can be used to approach a new *NIX system. Simply try to find the common ground and work from there. Consider the terminal window as the steering wheel. Most of the things you will find will look familiar already. It cannot be stated often enough: the logical base is the same. Switching from SAM (HP-UX) to smit (AIX) and further to the various GUI programs to be found on Solaris is much harder than firing up a terminal and having a look into `/etc` to find out what is going on right now. An added bonus of this approach is the fact that many of the hints in this book can be used on a terminal, from a PDA with solid vt100 emulation to a cell phone with a different vt100 emulation. You just have to know how.

Preface

As already mentioned in the foreword, *NIX has a strictly logical structure. Its paradigm is, “Better small programs doing one or two things right.” *NIX started its life as a development environment; because of this heritage, you will always find one or two ways to achieve a certain goal. The system can be (almost) completely customized to fit your needs. Proofs for this approach are *NIX-based cell phones or VCRs. And note, just because I say almost completely does not mean that something cannot be done, but rather that the last bit might take more time and effort than it’s worth. If you wish to exchange the graphical user interface, you can do so without problems; the whole system is built that way. If you feel inclined to create a completely new GUI for *NIX, you can do so as well. The sheer number of window managers available proves this point. But working as system administrators forces different priorities on you. How could you administer a system or a lot of systems using the path of least resistance? What parts of the system are to be configured how? This book presents a step-by-step approach to various topics regarding systems administration. It starts with a couple of hints about how to find out what is going on when confronted with an already operational system, and it discusses questions regarding backup and the installation and configuration of a new system, how to ready the new system for production, and how to secure that system.

Chapter 1, “Things to Check,” contains hints as to what to look for on an “inherited” system, which configuration settings are important, and how to get a general overview about the system.

Chapter 2, "Working on *NIX," starts with a short introduction about the creation of shell and Perl scripts and moves into how these scripts can be executed with the help of a basic *NIX service.

Chapter 3, "Maintenance," discusses recurring tasks on the systems and a description of the baselining technique. The continuous gathering of vital data to detect and predict problems before the users notice them is discussed, including hints on how to evaluate this data.

Chapter 4, "Backup," discusses various aspects of the topic and contains hints of what to look for.

Chapter 5, "Setting Up a System," discusses the installation of a new system and what is needed to ready the system for production use. It also contains a quick rundown of the boot process and how the init states are used on various brands of *NIX.

Chapter 6, "The Test System and the 'Secure' System," is about the benefits of having a system for tests available and includes arguments to get the system in the budget. It further explains the concept of the "secure system," an approach used by many seasoned system administrators to increase security as well. The chapter closes with a discussion of how to document tools to ease the creation of the necessary documentation.

Chapter 7, "Security," discusses various aspects regarding IT security. You will find hints and tips regarding this hot topic.

The book closes with Chapter 8, "That's It (Almost)," my goodbye to you. Appendix A, "One-Liners," contains useful commands (some longer than one line) to make the daily work easier.

Appendix B, "Software Used in This Book and Where to Get It," lists the Internet addresses where the software used in this book can be obtained.

Appendix C, "Internet," contains a list of useful Web sites.

All examples are set up in a way that they can be used instantly on almost every system. Some examples can have side effects, though; you should not try them on a production system. It boils down to a simple concept—if there are more users working on a system, then all users (including you) can expect a certain courtesy from each other. Your job is to make sure that the systems can be used to their full potential. If something should not work, then it is your job to minimize the impact on the performance of the machine. What can be done to reach these goals is the subject of this book.

1 Things to Check

In This Chapter

- Environment Variables
- Peparations

Not all systems you have to administer are set up by you, the person responsible for the administration. Usually you *inherit* systems that were installed by another person, installed according to that person's idea how things should be done. If the installation was documented accordingly, you could read the documentation, make sure to understand the *whys* of the installation, and be done. In theory, there is no "wrong" way to set up a *NIX box; there are just things to avoid—some at all costs.

However, many systems are in constant flux, as the services run on those systems are changed or the advancements and updates of the operating system are installed. This chapter is a quick rundown of things that need to be checked first.

By no means should you rush in and change everything according to some higher plan you made. Sometimes there is a reason for certain settings; however, some settings have just been overlooked. It is your job to find out what should be changed and what will probably break if changes are committed to the system. Priority number one is to document. It may sound like a broken record, but there will be times when proper documentation really saves your day.

1.1 ENVIRONMENT VARIABLES

There are a lot of settings concerning the environment variables on a system, enough to give you a headache. However, because all programs use them, you should be aware of the consequences surrounding them.

1.1.1 The `$PATH` Variable

Every *NIX system uses the `$PATH` variable to find all the executables available to the currently logged-in user account. If the `$PATH` variable looks like the one shown in Listing 1.1, there could be trouble, depending on the flavor of operating system (OS).

LISTING 1.1 Displaying the Current `$PATH`

```
$:~> echo $PATH
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/lib/java/jre/bin:/opt
/gnome/bin:.
$:~>
```

The fact that programs in `/usr/local/bin` are called first, might be okay on a Linux system; however, the same thing on Solaris could create some confusion if you are not aware of it.

For example, there are sometimes two `tar` programs on a Solaris system, and each behaves differently. One is in `/usr/bin`, and one is in `/usr/local/bin`. The latter knows how to use the `gzip` library to compress and uncompress the resulting archives. This means if you are switching between two Solaris systems, the command line `tar xzf` will work on the system with `/usr/local/bin` first in the `$PATH`. However, you will get an error message on the system that uses the `tar` in `/usr/bin`. This means that the `tar` command supplied by Solaris cannot open files created with the GNU version, the version installed in `/usr/local/bin`.

NOTE

Always include the path. Most programmers create variables containing the program name, including the complete path, if they want to make sure the program they intended to use gets executed, for example, `TAR=/usr/local/bin/tar`. *Then, if* *the* `tar` *program is needed, it is called with* `$TAR` *instead of* `tar`*. Relying on the operating system to call the "right" program is a risky approach, and it will go wrong eventually.*

That does not necessarily mean that you cannot have `/usr/local/bin` as first entry in your `$PATH`—you just need to be aware of the consequences. Changing the `$PATH` requires thorough testing and examination of the deployed scripts; they might rely on a certain order regarding the `$PATH` variable.

However, the setting at the end of the sample variable is more disturbing. If you try to start a program, the current directory `$PWD` will be checked as well. Because all other components of the `$PATH` will be checked first, you are less likely to launch a program you didn't intend to start, but nevertheless, the current directory shouldn't be included in the array of paths that are searched for executables. Again, if you decide to change this setting, make sure there are no scripts or worse, users, relying on the current setting.

1.1.2 The Prompt

If you have to switch between systems on a regular basis, you should set your `$PS1` variable to give you some hint about the user-context and machine you are on at the moment. Tailoring this variable to meet your individual needs is done best by using one of the files in the respective `$HOME` directory read by the shell during launch. Depending on the shell used, either `.profile` or one of the configuration files read by the respective shell could be used, e.g., `.cshrc` or `.bashrc`. Listing 1.2 shows the entry.

LISTING 1.2 Universal Prompt

```
PS1="$USER@`uname -n`> "
```

It is important to note that the characters around `uname -n` are so-called *back ticks* (`` ` ``). A single quote (') would not work. A command enclosed in back ticks is executed and the result replaces the command. The sequence in the listing works on almost every *NIX-based operating system.

If you use the "GNU Bourne-again shell", `bash` for short, the sequences in Listing 1.3 and Listing 1.4 would work as well.

LISTING 1.3 Alternate Sequence for `bash`

```
PS1="\u@\h> "
```

LISTING 1.4 Sequence Including the Current Path

```
PS1="\u@\h:\w> "
```

These alternate sequences are less "talkative," though.

This simple configuration item allows the running shell to display the information. If times are getting hectic, a quick glance at the prompt and you know which host you are on and the user-name you are working as.

1.1.3 The `$MANPATH` Variable

If you use the command `man` at the command line, you expect it to magically find all information about the installed programs. However, this "magic" requires that the system is set up to work that way. `man` uses the `$MANPATH` variable to search for manpages. Every directory included in `$MANPATH` will be searched; every directory not included will be ignored, obviously. More often than not, `$MANPATH` contains only `/usr/share/man`. In that case, for example, all the information in `/usr/local/share/man` would not be processed because `man` would not know about it. To make matters worse, `man -k` is rarely used on Solaris systems, because nobody bothered to create the index to search in initially. Listing 1.5 shows the code to create an index on a Solaris system.

LISTING 1.5 Creating the Index on Solaris

```
root@unixhost:> echo $MANPATH
/usr/dt/man:/usr/man:/usr/share/man:/opt/sfw/man:/usr/local/man
root@unixhost:> catman -w
```

Make sure `$MANPATH` incorporates all directories containing manpages and execute `catman -w`. The `-w` option instructs the program to create the so-called *windex* database. This database contains the index to search *within* (the contents of) the manpages.

Complicating matters further is the fact that there is no consistent location other than `/usr/share/man` and `/usr/local/share/man` where manpages are stored. There are different approaches, every one valid in its own right, but if you have different flavors of *NIX to administer, you want to make sure all directories containing manpages are searched accordingly. To illustrate the current situation, you can look at examples of how the `$MANPATH` might look like on GNU-Linux (Listing 1.6), Solaris (Listing 1.7), and OS X (Listing 1.8), all *NIX-based systems.

LISTING 1.6 `$MANPATH` on GNU-Linux

```
/usr/local/man:/usr/share/man:/usr/X11R6/man:/opt/gnome2/man:/opt/gnome
/man
```

LISTING 1.7 `$MANPATH` on Solaris

```
/opt/sfw/man:/usr/man:/usr/local/man:/usr/dt/man:/usr/openwin/man:/usr/
X/share/man
```

LISTING 1.8 `$MANPATH` on OS X

```
/sw/share/man:/sw/man:/usr/share/man:/usr/X11R6/man
```

That does not necessarily mean the settings shown in these listings are the way to go. You must adapt the settings to your systems. Some applications even store their manpages in the directory they were installed in.

The order in `$MANPATH` is equally significant. The manpages are searched on a per directory basis in the order they appear in the variable. In the Solaris example, `man` would find the GNU variant of `ls` first. This variant is able to display the directory listings color-coded, provided the right parameters are used. These parameters would not work with the SUN version of the command, though.

To find out about the situation on your system, use the command `echo $MANPATH`. If the output more or less resembles the examples shown earlier in this section, most of the available manpages should be searched and indexed. If time permits, take a look around on the machine and make sure there are no omissions. If the output is `/usr/share/man`, it might be a system that serves a certain purpose only and thus contains only a subset of the available software, or it might be that the person who ran the initial setup simply forgot to adjust the `$MANPATH` accordingly.

TIP

*Always install all manpages. Because hard disk space comes cheap these days, installing all available manpages, if possible, is advisable, provided, the *NIX in use allows to install the manpages separately. You might even find a program you weren't aware of, possibly enabling you to do your job faster, easier, etc.*

You should take the time to run `man intro` on all systems; the "intro" manpage installs automatically on every system and contains useful information about it. An interesting way to search the installed manpages is the `-k` option of `man`. This option allows you to search inside the available manpages. You just supply the word you are searching for, as shown in Listing 1.9.

LISTING 1.9 Searching the Manpages with `man -k`

```
root@unixhost:> man -k passwd
d_passwd          d_passwd (4)        - dial-up password file
getpw             getpw (3c)          - get passwd entry from UID
kpasswd           kpasswd (1)         - change a user's Kerberos
                                        password
nispasswd         nispasswd (1)       - change NIS+ password
                                        information
nispasswdd        rpc.nispasswdd (1m) - NIS+ password update daemon
pam_passwd_auth   pam_passwd_auth (5) - authentication module for
                                        password
passwd            passwd (1)          - change login password and
                                        password attributes
passwd            passwd (4)          - password file
```

By taking a closer look at the last two lines of the listing, you will notice two entries for the word `passwd`. The command `man passwd` will display only the information contained in section one, `passwd (1)`. To access the information contained in section four, `man` needs to be instructed to do so. The different calling conventions are shown in Listing 1.10.

LISTING 1.10 Accessing Information Contained in Sections with `man`

```
erikk@unixhost:> man passwd
Reformatting page.  Please Wait... done

User Commands                                          passwd(1)

NAME
     passwd - change login password and password attributes

SYNOPSIS
     passwd [-r files | -r ldap | -r nis | -r nisplus]  [name]

...

erikk@unixhost:> man -s 4 passwd
Reformatting page.  Please Wait... done<br/>

File Formats                                           passwd(4)
```

```
NAME
     passwd - password file

SYNOPSIS
     /etc/passwd

DESCRIPTION
     The file /etc/passwd is a local source of information  about
     users'  accounts.  The password file can be used in conjunc-
...
```

The manpages are the most up-to-date reference concerning every system. Every update supplied by the *NIX distributor will update the manpages as well. This information is an invaluable source to understand the inner workings of the system.

1.1.4 What Time Is It?

A *NIX system needs to know about date and time. If this information is not set up correctly, some internal workings will suffer. The machine uses the internal clock to keep track of date and time. Older systems, in terms of hardware age, tend to lose or gain a few seconds at times, partly due to the fact that the batteries used in these systems are exhausted, for example. If this happens, other means of keeping the date and time current have to be used. Another component of date and time is the `$TZ` variable. It stores the time zone the system is in and should be consistent among all systems in the same geographical location.

If you transfer a file from a system in Chicago to a system in Philadelphia without a correct `$TZ` value, the timestamp on the system in Philadelphia would differ from the one in Chicago by 60 minutes. This does not seem to be a big problem, but if a program bases the question to overwrite the file on the timestamp, problems could arise. If those systems have a correct `$TZ`, the right decision will be made. If not, you could lose valuable data. In this example, the system in Chicago should have `$TZ` set to `CST` or `US/Central`, and the one in Philadelphia either `US/Eastern` or `EST`.

1.1.4.1 How to Set the $TZ

The `$TZ` is set during install and can be changed for the user-context later with the help of the command `tzselect`. `tzselect` is used to get the valid `$TZ` value. It asks the user questions about the geographical location and outputs a string like `US/Eastern` suitable to be used. If the `$TZ` is not set on a system, make sure you understand why before setting it. The same goes for “wrong” `$TZ` settings; there could be a reason for them to be “wrong.”

Why should you check first for a wrong `$TZ`? *A couple of years ago, someone stopped the author a second before he could make a change to a seemingly "wrong"* `$TZ` *setting. As it turned out, the person originally installing the *NIX servers rolled out about 200 desktop machines configured with a false* `$TZ` *setting. So, instead of rectifying this on the individual machines, the decision was made to make up for the error by setting two of the servers to a different* `$TZ` *to compensate for the "lost" hour.*

Making the change the author was intending to make at the time would have confused a couple of tracking processes in the CMS (Content Management System), and thus six weekly magazines would have had to be checked by hand for missing content—for a print run about 90 minutes away! The correction took place during a rollout of upgrades to the desktop systems at a later date. Important here: ask first, understand the ramifications, calculate the risk, and then change!

1.1.4.2 How to Set the Time

There are two ways to set the time on a *NIX system, or on most computers for that matter. Option one, the preferable one, uses a so-called *time-host* to allow the systems to get the time on a regular basis. This means one or two computers on the network connect to some reliable source inside or outside the network, get the date and time, and provide this information as a service to the other systems. The advantage here is that all systems agree on the current time and date, and you have only one or two systems to maintain for that purpose.

The "old" way of setting the time using one time-host is the `rdate` protocol with a resolution of 1 second. The preferred way of setting the time uses the `xntp` package, available for almost all *NIX flavors. The latter has a drawback, though. To avoid tampering, security tokens have to be generated and exchanged between the time-host and the clients. On the upside, `ntp` is able to get the time from more than one host and calculates the latency, thus resulting in a more precise timing.

To find out what is used to set the time, check first for a file called `/etc/ntp.conf`. If it exists, chances are good `ntp` is used. Otherwise, scan the `root crontab` for entries containing `rdate` or `ntpdate`—the latter is a replacement for `rdate`. If neither exists and the system time seems to be off, read the earlier Note about checking first for a wrong `$TZ` again and proceed.

Though setting the time and date manually should be only an interim solution, the command `date` can be used to do so, as shown in Listing 1.11.

LISTING 1.11 Setting the Time Manually

```
root@unixhost:> date -s "15:40:00 02/09/04"
Mon Feb  9 15:40:00 CET 2004
root@unixhost:> date
Mon Feb  9 15:40:19 CET 2004
root@unixhost:>
```

If a GNU-Linux system is used, some more work is involved to convince the hardware to actually store the time and not use it just for the session. The command `hwclock` is used to write the time and date to the system-clock.

If you need to change the current time, make sure you do not "jump" too far ahead or back. There are programs that might react to jumps that are too "big," especially server processes, version control, or file services, such as `netatalk` and `smbd`; the former serves older Apple clients and the latter Windows clients. Background processes used to keep the time current in a network usually deploy a so-called *slew* mode. This means the change of time will happen gradually to avoid confusing programs relying on timestamps.

1.2 PREPARATIONS

If you *inherited* a system, you should try to get some overview of what was installed, where it was installed, and the current state of the configuration. You can use a couple of proven approaches to accomplish this.

1.2.1 `$HOME` of `root`

Assuming you are logged in as `root`, the first commands to use are `cd` followed by `pwd` to find out if the superuser has its own `$HOME` directory.

```
root@unixhost:> cd
root@unixhost:> pwd
/root
root@unixhost:>
```

The example shown is from a Linux host. `root` has its own home on Linux, but on Solaris the home of `root` is usually the `/` directory.

There are a couple of reasons for the setting on Solaris. One of them, probably the most important, is that `root` needs access to his home directory when logging in. If something is wrong with the system, not being able to access the directory `$HOME` points to can lead to problems, even a non-working login.

On the other hand, there should be a place owned by `root` to store confidential files. If needed, you should create a `root` home directory.

To create a home for `root` on a system where `root` has none, you must take some precautions. Listing 1.12 illustrates the necessary steps.

LISTING 1.12 Creating a `root $HOME`

```
root@unixhost:> cd /
root@unixhost:> mkdir .HO
root@unixhost:> ls -ld .HO
drwxr-xr-x    2 root     root          48 Feb  9 17:05 .HO
root@unixhost:> chmod -R g-r-x,o-r-x .HO
root@unixhost:> ls -ld .HO
drwx------    2 root     root          48 Feb  9 17:05 .HO

root@unixhost:>
```

This sequence of commands works like this: the first command, `cd /`, changes to the root of the filesystem. The next command, `mkdir .HO`, creates a directory that will be hidden from "normal" users. `ls -ld .HO` shows the permissions and tells us that all users have read and execute rights on the newly created directory. To rectify this, the command `chmod -R g-r-x,o-r-x .HO` is used. It revokes the read and execute permissions from the group, in that case `root` and the *others*, all valid logins not belonging to the group. The directory is not really "hidden", but won't show up if the `ls` command is not used with the option `-a`, which translates to show even the "hidden" files starting with ".".

NOTE

The `$HOME` *of* `root` *needs to be accessible.* `Root`*'s* `$HOME` *(the location pointed to in the file* `/etc/passwd`*) must be on a volume that will be mounted during all* `init` *states. If the system starts and cannot access* `root`*'s* `$HOME`*, the boot process will hang forever. The same goes for changing the* `init` *state; there could be problems doing so if* `root`*'s* `$HOME` *is inaccessible.*

1.2.2 More `roots`?

Some systems have more than one superuser. At first glance, this does not seem to make sense, but sometimes creating more than one user with `UID 0` is necessary. Imagine a data center, working in 24-hour shifts, 7 days a week. There should be more than one system administrator, obviously, but how can you differentiate who worked as superuser at what date and time?

Clearly the documentation should shed some light on the question, but to check whether there is more than one superuser, you can use the chain of commands shown in Listing 1.13.

LISTING 1.13 Finding Users with `UID 0`

```
root@unixhost:> awk -F: '{if($3==0) print $0}' /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
```

If the output looks like the one shown, there is only one root. Listing 1.14 demonstrates another scenario.

LISTING 1.14 Finding Users with `UID 0` (More Than One Superuser)

```
root:x:0:1:Super-User:/:/sbin/sh
ek_root:x:0:1:ek root account:/:/sbin/sh
mw_root:x:0:1:mw root account:/:/sbin/sh
fh_root:x:0:1:fh root account:/:/sbin/sh
```

On the machine shown in this example, you see three people able to become superuser. All accounts should have different passwords, and if one of those people changes the user-context to superuser, a log entry will be generated.

Needless to say, no one is supposed to use the `root` account other than the ones who match the initials in the example. Of course, how the multiple accounts are organized on the system you are working on depends on how the person who initially set up the system chose to differentiate between the system administrators.

A setup like the one described requires all system administrators to be aware of two things when changing to superuser:

- Never use the `root` account.
- Always log in with a normal user account and use `su` to switch to superuser. There is a command called `sudo` available on most GNU-Linux and BSD systems. If needed, one can install it on Solaris and most other *NIX variants, as well, but typing `sudo` repeatedly when executing commands can be annoying.

If these requirements are honored, all system administrators are able to see who worked as superuser and when.

NOTE *Solaris introduced the concept of roles a while ago; this concept is something completely different than the use of multiple superusers.*

A role contains permissions to do certain things on a system. These permissions are defined to fit certain job descriptions. This does not necessarily mean a user with certain granted rights is able to administer the system as a whole. It is possible to create a role containing all administrative rights, but this approach does not reflect the multiple `UID 0` *user concept.*

If a system runs 24/7, one superuser should always be able to access the system and take care of possible issues. Common sense tells us we cannot possibly define user rights in a way that covers all possibilities. Some issues to be resolved require drastic measures, and you couldn't anticipate all of them.

1.2.3 How Many Volumes?

Unlike other operating systems, *NIX always uses a tree-like structure for the connected hard disks and the filesystems stored on them. The tree starts on the `/` directory; everything else can be reached from there. It is possible to `mount` another filesystem "into" a directory to get more space; the actual contents of this directory will be hidden after the mount is complete, though.

To get an overview of how many volumes are currently mounted on a system, you can use the command shown in Listing 1.15.

LISTING 1.15 Displaying the Mounted Volumes

```
root@unixhost:> df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/dsk/c0t3d0s0     1.5G  1.1G  403M  73% /
/dev/dsk/c0t3d0s1     482M  311M  123M  72% /var
swap                  546M   40K  546M   1% /var/run
swap                  546M  312K  546M   1% /tmp
/dev/dsk/c0t3d0s7     1.5G  864M  550M  62% /export/home
```

The command `df`, easy to remember as *Disk Free*, run with the option `-h` for *human readable* shows how many volumes are there and how much space is available on them. The preceding example is from a small Solaris system. If the `-h` option is not available on your system, use `-k` instead. The resulting numbers will be in kilobytes instead of the default 512-byte blocks, which is even harder to grasp.

1.2.4 What Is Going On?

There are three commands to get an overview about what the system is doing right now. The commands are `ps`, `top`, and `sar`. Some of the commands will be covered in greater detail in Chapter 3.

1.2.4.1 ps

The `ps` command comes in handy if you have a need to determine whether a certain process is running on the system or how many instances of that process are active. The downside of `ps` is that few commands have more options; depending on incarnation of the command, you have between 20 and 50 single options, which can be combined to restrict the output or make it more talkative. The BSD version is used with the option `uawx` most of the time. To call the program with this option on Solaris, you should use `/usr/ucb/ps` to start the BSD version instead of using just `ps`. (The BSD binaries are still present on most Solaris machines; they are just hidden in `/usr/ucb`.) For example, to find the 10 processes using most of the CPU cycles, you can use the following commands, as shown in Listing 1.16.

LISTING 1.16 Running `ps`

```
root@unixhost:> /usr/ucb/ps uawx | head
ekeller    472  0.4  7.211176 8760 ?        S 10:06:14  0:05
                                            /usr/open win/bin/Xsun
root       632  0.2  0.7 1008  800 pts/5    O 10:17:54  0:00
                                            /usr/ucb/ ps uawx
ekeller    565  0.2  3.9 7160 4768 ??       S 10:06:48  0:00 /usr/dt/
                                            bin/dtterm-s
root         3  0.1  0.0    0    0 ?        S 10:05:34  0:00 fsflush
root       473  0.1  1.6 2320 1896 ?        S 10:06:15  0:00 mibiisa
                                            -r -p 32788
root         1  0.0  0.3 1232  368 ?        S 10:05:34  0:00 /etc/
                                            init -
ekeller    564  0.0  3.6 6920 4392 ??       S 10:06:48  0:00 /usr/
                                            dt/bin/dtterm-s
root         0  0.0  0.0    0    0 ?        T 10:05:31  0:00 sched
root         2  0.0  0.0    0    0 ?        S 10:05:34  0:00 pageout
```

The output of the preceding example is truncated at around 80 characters for clarity. The options modify the output in the following manner: `u` displays the user-related information among other things and sorts the processes by CPU usage, `a` shows the processes of all users instead of those of the calling user only, `w` uses 132 columns to display the information instead of the width of your terminal window, and `x` shows processes that are not bound to a terminal. The last option is important because processes not connected to a terminal are using CPU cycles, too. The pipe `|` at the end of the command relays the output to the program `head` to limit the number of lines to ten, the default, so calling `head` with `-10` is not necessary. If you want to find the “top ten” of your processes, this command is all you need.

If you get an error message when using the option `uawx`, then the other variant of `ps` is installed. You have to use the option `-ef` instead to get a similar output, but this version does not allow the easy sorting of processes based on their CPU usage.

Another use of `ps` is to determine how many processes of a certain program are running. This can be accomplished by chaining the output of `ps` to other commands at the users' disposal. To find out how many processes `sendmail` has spawned, for example, one possible chain of commands could be as shown in Listing 1.17.

LISTING 1.17 Count of Running `sendmail` Processes

```
root@unixhost:> /usr/ucb/ps uawx | grep sendmail | wc -l
3
```

It seems that there are three running `sendmail` processes. To verify that this result is correct, starting the command without relaying the output to `wc`, thus omitting the `| wc -l` part, displays the found number of lines for inspection (see Listing 1.18).

LISTING 1.18 Displaying Executing `sendmail` Processes

```
root@unixhost:> /usr/ucb/ps uawx | grep sendmail
root      740  0.1  0.6  992  696 pts/5    S 11:57:03  0:00 grep
sendmail
smmsp     592  0.0  1.2 4352 1448 ?        S 10:07:08  0:00 /usr/lib/
sendmail -Ac -q15m
root      593  0.0  1.7 4384 2000 ?        S 10:07:08  0:00 /usr/lib/
sendmail -bd -q15m
```

This command line reveals there are actually *two* running `sendmail` processes. `grep` listed itself because the string `sendmail` is part of the command as well. To make use of the count, the line containing `grep` has to be eliminated from the list that is relayed to `wc`. There is more than one way to achieve this; option one, see Listing 1.19, is a commonly used choice.

LISTING 1.19 Suppressing the Display of Lines Containing `grep`

```
root@unixhost:> /usr/ucb/ps uawx | grep sendmail | grep -v grep
smmsp      592  0.0  1.2 4352 1448 ?        S 10:07:08  0:00 /usr/lib
/sendmail -Ac -q15m
root       593  0.0  1.7 4384 2000 ?        S 10:07:08  0:00 /usr/lib
/sendmail -bd -q15m
```

Telling `grep` to ignore lines containing the word `grep` by using the option `-v` feeds the right information into `wc`, but another `grep` process is needed to do so. This third process, to get things right, won't bring the machine to its knees, to be sure, but option two (see Listing 1.20) eliminates the need for the third process completely.

LISTING 1.20 Suppressing the Display of Lines Containing `grep` without a Second Process

```
root@unixhost:> /usr/ucb/ps uawx | grep [s]endmail
smmsp      592  0.0  1.2 4352 1448 ?        S 10:07:08  0:00
/usr/lib/
sendmail -Ac -q15m
root       593  0.0  1.7 4384 2000 ?        S 10:07:08  0:00
/usr/lib/
sendmail -bd -q15m
```

The square brackets around the `s` prevent `grep` from finding another line containing `sendmail`. How so? `grep` uses so-called "regular expressions" and the square brackets denote a choice, e.g., `grep [Ss]endmail` would find lines containing `Sendmail` and `sendmail`, but to find itself again, one would have to search for the square brackets as well.

1.2.4.2 `top`

The `top` command acts like running `ps` permanently while allowing the user to change parameters or apply filters to the output. `top` is usually available on all *NIX distributions. After you start `top` via the command line, the screen looks like Listing 1.21.

LISTING 1.21 `top` Default Display

```
last pid:  1026;  load averages:  0.12,  0.19,  0.13
14:17:37
67 processes:  65 sleeping, 1 running, 1 on cpu
CPU: 97.4% idle,  1.8% user,  0.8% kernel,  0.0% iowait,  0.0% swap
Memory: 128M real, 34M free, 60M swap in use, 542M swap free

   PID USERNAME LWP PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
   874 root       1  59    0   11M 8808K sleep   0:10  0.60% Xsun
  1025 erikk      1  59    0 2344K 1616K cpu     0:00  0.23% top
   958 erikk      5  59    0 8912K 7144K sleep   0:05  0.15% dtwm
   472 root       7  59    0 2320K 1896K sleep   0:03  0.09% mibiisa
   988 erikk      1  59    0 7008K 4744K sleep   0:00  0.00% dtterm
   238 root       8  59    0   44M 9072K run     0:01  0.00% java
   959 erikk      1  49    0   10M 7720K sleep   0:01  0.00% dtfile
   461 root       3  59    0 2672K 2008K sleep   0:01  0.00% vold
   956 erikk      1  49    0 7720K 4464K sleep   0:00  0.00% dtsessio
   983 erikk      1  49    0   10M 1264K sleep   0:00  0.00% dtfile
  1010 erikk      1  59    0 6920K 4648K sleep   0:00  0.00% dtterm
   995 erikk      1  59    0 6912K 4648K sleep   0:00  0.00% dtterm
   962 erikk      1  59    0 7080K 4640K sleep   0:00  0.00% sdtperfm
   955 erikk      1  59    0 4696K 3384K sleep   0:00  0.00% ttsessio
   875 root       1  59    0 6840K 3104K sleep   0:00  0.00% dtlogin
```

Listing 1.21 shows a machine without load. The first line displays the `PID` of the process last started, load averages, and the current time. The next line shows the number of processes and how many are running at the moment. The third line contains the current CPU statistics and the last line of the header shows the amount of memory available and how much of it is actually used.

The remaining lines are the currently running processes sorted by their CPU usage. Typing `u` followed by a valid username filters the output to processes owned or run by a specific user (Listing 1.22 and Listing 1.23 illustrate the process).

LISTING 1.22 `top` **Filtering One User**

```
Memory: 128M real, 34M free, 60M swap in use, 542M swap free
Username to show: erikk
   PID USERNAME LWP PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
   874 root       1  59    0   11M 8808K sleep   0:10  0.60% Xsun
```

The screen changes to the output shown in Listing 1.23 after pressing the Enter Key.

LISTING 1.23 `top` **Filtered Display**

```
last pid:  1061;  load averages:  0.04,  0.04,  0.05
14:42:09
70 processes:  68 sleeping, 1 running, 1 on cpu
CPU: 99.6% idle,  0.2% user,  0.2% kernel,  0.0% iowait,  0.0% swap
Memory: 128M real, 33M free, 61M swap in use, 541M swap free

   PID USERNAME LWP PRI NICE  SIZE   RES STATE   TIME    CPU COMMAND
  1050 erikk      1  59    0 2336K 1600K cpu     0:01  0.23% top
```

```
 988 erikk      1  59    0 7008K 4744K sleep   0:00  0.01% dtterm
1060 erikk      1  59    0 1904K 1336K sleep   0:00  0.01% vi
 958 erikk      5  59    0 8912K 7144K sleep   0:05  0.01% dtwm
1010 erikk      1  59    0 6928K 4744K sleep   0:00  0.00% dtterm
 959 erikk      1  49    0   10M 7720K sleep   0:01  0.00% dtfile
 956 erikk      1  49    0 7720K 4464K sleep   0:00  0.00% dtsessio
 983 erikk      1  49    0   10M 1264K sleep   0:00  0.00% dtfile
 995 erikk      1  59    0 6912K 4648K sleep   0:00  0.00% dtterm
 962 erikk      1  59    0 7080K 4640K sleep   0:00  0.00% sdtperfm
 955 erikk      1  59    0 4696K 3384K sleep   0:00  0.00% ttsessio
 987 erikk      1  59    0 3168K 2304K sleep   0:00  0.00% dtexec
1009 erikk      1  59    0 3168K 2304K sleep   0:00  0.00% dtexec
 994 erikk      1  59    0 3168K 2304K sleep   0:00  0.00% dtexec
 939 erikk      1  59    0 4032K 2088K sleep   0:00  0.00% sdt_shel
```

By using this basic functionality, you can see why `top` is one of the most important tools on any given *NIX system. Other functions available are dependent on the version of `top` installed on the machine. Using the `?` key while `top` is running displays one or more help screens describing the capabilities of the running program.

`top` is available for almost every *NIX variant, but not necessarily installed by default or even available on the installation media. If `top` is not to be part of the *NIX distribution, you can download it from the locations given in Appendix B.

1.2.4.3 sar

The `sar` command, available on many *NIX systems, spells out to "system activity reporter" and covers a huge amount of measurements, ranging from CPU statistics to I/O on hard disks and terminal lines among others. The number of probes recorded varies between the different flavors of *NIX. The Solaris variant knows 15 groups, while the OS X supplied version is limited to 4.

The mode of operation is the same on all supported platforms—`sar` is either run if needed, without storing any data, or an entry in a `crontab` initiates the collection of the data and stores it somewhere in the `/var` hierarchy. The resulting data files will be recycled, if configured. On a Solaris system, the filenames consist of the string `sa` and the day of the month, e.g., `sa05` for the fifth day of the month. This mode of rotation allows to access "historical" data up to 30 days.

To calculate and display information about the CPU load, you can use the command in Listing 1.24.

LISTING 1.24 `sar` One Shot (1)

```
erikk@unixhost> sar 5 5

SunOS unixhost 5.9 Generic_112233-11 sun4u    07/05/2004

13:56:20    %usr    %sys    %wio   %idle
13:56:25       2       1       0      97
13:56:30       2       0       0      98
13:56:35       3       1       0      97
13:56:40       3       1       1      95
13:56:45       3       0       0      97

Average        2       1       0      97
erikk@unixhost>
```

The command used executes `sar`, telling it to get five samples in a 5-second interval. The machine in the preceding Listing 1.24 seems to be low on load; a system with a higher workload would appear more like the one shown in Listing 1.25.

LISTING 1.25 `sar` One Shot (2)

```
erikk@unixhost> sar 5 5

SunOS unixhost 5.9 Generic_112233-11 sun4u    07/05/2004

14:07:28    %usr    %sys    %wio   %idle
14:07:33      23      15      62       0
14:07:38      22      15      63       0
14:07:43      23      18      59       0
14:07:48      25      22      53       0
14:07:53      24      27      49       0

Average       24      19      57       0
erikk@unixhost>
```

This listing shows the same machine, but a little more taxed by processes running in the background. Analyzing the data in Listing 1.25 seems to indicate that the system spends most of its time waiting for I/O operations to complete. So the one-shot mode of `sar` can be used to diagnose a condition on the fly. Job one, in this case, would be to find out why the machine complains about slow I/O. One could use `top` to find the process(es) hogging the I/O subsystem or run `sar` repeatedly in concert with `ps` to find the reason for the high *wait I/O* percentage.

Another approach could be to analyze what happened earlier today by using the stored data collected by `sar`. If `sar` stores the data via an entry in a `crontab`, you could use the information to get a more general view of the load situation. Displaying the stored data is shown in Listing 1.26.

LISTING 1.26 Accessing Stored Data with `sar`

```
erikk@unixhost> sar -s 13:00 -e 14:30

SunOS unixhost 5.9 Generic_112233-11 sun4u    07/05/2004

13:00:01    %usr    %sys    %wio   %idle
13:10:00       1       0       0      98
13:20:00       0       0       0      99
13:30:00       0       0       0     100
13:40:01       0       0       0     100
13:50:00       1       0       0      99
14:00:00       0       0       0      99
14:10:01       3       2       5      90
14:20:00       0       0       0     100
14:30:00       0       0       0     100

Average        1       0       1      98
erikk@unixhost>
```

Examining the output in Listing 1.26 shows that there was just a case of heightened activity. Judging the cumulative values sampled over a period of ten minutes indicates nothing serious so far. The `-s` and `-e` options are used to narrow the window of data to be displayed to the time period of 1:00 P.M. to 2:30 P.M. on that day. You will find out more about `sar` in Section 3.1.6, "Monitoring the CPU."

1.2.5 Create a List

One of the first things that comes in handy to you when working with a system is a listing of all files on a specific machine. You have more than one way to accomplish getting that list, but nothing beats a text file, which can be "`greped`" in case of need. The command shown in Listing 1.27 can be used to create a list of files, spanning all mounted volumes.

LISTING 1.27 Creation of a List Containing All Files

```
root@unixhost:> cd /
root@unixhost:> find . > flist.txt
```

A word of caution, though—this command needs to be run as `root` and could slow down the system considerably. If the system is under heavy load, the creation of the list should be scheduled during either a planned downtime or whenever the job affects the least number of users.

Also, remember to move the resulting list into your private `root $HOME` as soon as possible. It contains sensitive information about the system.

If the system in question should be used as a file server or some other setup with huge disk space, then creating a catalog of all files on the shared volumes simply does not make sense. To just create a list of the files on the important volumes, you can use the `-xdev` parameter to the `find` command (assuming all relevant directories are on the root volume), as shown in Listing 1.28.

LISTING 1.28 Creation of a List of One Hard Disk Only

```
root@unixhost:> cd /
root@unixhost:> find . -xdev > flist.txt
```

This option tells `find` not to traverse *mount points*. If, for example, `/usr/local` is a mounted volume, then you need to repeat the command as shown in Listing 1.29.

LISTING 1.29 Adding Directories to the List

```
root@unixhost:> find /usr/local -xdev >> flist.txt
```

You must repeat this command for every mounted directory whose contents should be included in the list. Do not `cd` to the directory because you would lose the path that leads to the files.

This list (stored somewhere only `root` has access to) enables the administrator to find even the most exotic files and directories. This list especially comes in handy if a certain program complains about not being able to locate a specific file. In that case, the list can be used to search for the file, as demonstrated in Listing 1.30.

LISTING 1.30 Searching the Location of a File

```
root@unixhost:> grep Xdefs.h flist.txt
./usr/X11R6/include/X11/Xdefs.h
root@unixhost:>
```

This list can be used for other purposes as well. For example, it is used in Section 7.2.2, "A Simple File Checker," to create a tool to check for changes in vital directories.

1.2.6 Mail

Knowing what happens on the system regarding mail is crucial these days. This does not imply that an administrator should spy on the users, but he should keep an eye on the amount of mail that goes through the machine. The most common mailers found on a *NIX system are `sendmail` and `postfix`. The former comes with a function to create a statistics file that can be read with the command `mailstats`, as shown in Listing 1.31.

LISTING 1.31 Output of the Command `mailstats`

```
root@unixhost:> mailstats
Statistics from Wed Sep 17 16:03:39 2003
 M   msgsfr  bytes_from   msgsto    bytes_to  msgsrej msgsdis  Mailer
 3       14         21K       13         27K        0       0  local
=============================================================
 T       14         21K       13         27K        0       0
 C        3                    0                    0
```

If you should get an error message while running `mailstats`, you have to enable the statistics gathering by issuing the command used in Listing 1.32.

LISTING 1.32 Activating the Gathering of Mail Statistics

```
root@unixhost:> touch /etc/mail/statistics
```

Contrary to the `sar` mechanism, the mail stats are created on an ongoing basis. After 4 days, all you get is the combined data for these 4 days; you have no way to evaluate the data on a day-to-day basis.

The program could be used with the option `-p` to rectify this situation (see Listing 1.33). This option instructs the program to output the data in a format suitable for processing in scripts and to delete it afterwards. Because the statistical data will be deleted when displayed with the `–p` option, you have to gather the statistics in a file to allow later examination.

LISTING 1.33 Output for Further Processing

```
root@unixhost:> mailstats -p
1063807419 1090320821
 3       17          24       16          30        0       0  local
 T       17          24       16          30        0       0
 C        6       0       0
```

`postfix`, on the other hand, needs additional software to perform the same function, e.g., `isoqlog` to be found at *http://www.enderunix.org/isoqlog/*. Because installing this additional software requires that you to know how to compile it, a description is beyond the scope of this chapter.

1.2.7 Installed Software

Given the fact that there are no real rules where to install additional software on a *NIX system, just recommendations like the one provided by the "Filesystem Hierarchy Standard Group" at *http://www.pathname.com/fhs/*, most third-party software developers adhere to this standard or a scheme devised by themselves. The latter schemes are normally based on the FHS standard in one way or another. In the proposed standard, additional packages are to be installed in two locations:

- `/opt`
- `/usr/local`

As final step to gather information about a system not installed by yourself, you should browse through these locations for a while to get a better understanding of whether additional software has been installed and, if so, where. A potential caveat of this kind of custom install is the fact that additional configuration files might be hidden in these directories. If the programs installed in there should use these configuration files instead of the ones in `/etc`, being aware of this fact will save a lot of guesswork later. Using this approach is not wrong, but most system administrators are used to the fact that configuration files should reside in `/etc`. If the installed programs are compiled to ignore this directory and use their own hard-coded locations instead, you should put this information in the documentation accompanying the system. This is especially important if the programs in question usually do not exhibit this behavior or act differently on other systems that are part of the same network. For example, suppose program `foo` is installed on a variety of operating systems and reads the configuration files from `/etc/foo.conf`. If this program is installed as part of a package of programs to fulfill a certain need for the creator of the package, it might be the case that the creator of the package decided to make sure that another instance of the program would not interfere with its use in the package. Thus, the configuration for the program will reside in `/opt/package-name/etc/foo.conf` instead of `/etc/foo.conf`, a common case, given the versatility of many *NIX programs. If there is a necessary change of the configuration due to security reasons, the configuration in `/etc` will usually be changed according to the given information. If the configuration in `/opt/package-name/etc/foo.conf` should be changed as well, the system administrator should be aware of the fact that it exists in the first place, so he can act accordingly.

The directory hierarchy will be discussed further in Chapter 5.

1.3 SUMMARY

The information in this chapter should provide administrators with pointers of how to gather information about the configuration of environment variables and how to set the system time, if needed; how to investigate the number of superusers, should there be more than one; and how to create a secure location for `root` to store sensitive information concerning the system. Furthermore, this chapter discussed how to display statistics about running processes and the mail system and closed with a short discussion about where additional software should be installed.

This page intentionally left blank

2 Working on *NIX

In This Chapter

- Executables on *NIX
- Shell Scripts Step by Step
- Perl Scripts
- Automation

Compared to the "other" dominant operating system, *NIX uses a different approach regarding programming and system tasks. *NIX supplies the user with literally dozens of small programs capable of doing one or more things "right."

The programs talk to each other via defined interfaces, thus minimizing the risk of breaking something if one of them changes or gets expanded capabilities. You are able to choose your tools and chain them together to achieve what you have in mind, regardless of whether the small program in question is simply a script or a binary. Indeed, many commands are actually shell or Perl scripts, and, given the flexibility of this approach, you can study and re-use them or exchange them completely with something you write yourself. The latter approach is rarely necessary, but comes in handy if you need to take care of a very special case.

2.1 EXECUTABLES ON *NIX

How does the *NIX-system "know" which files are executable? Why isn't every user allowed to execute all programs on the system? How does the system make sure that programs don't overwrite data belonging to another process? These questions will be answered in the following sections.

2.1.1 Why Is a Program Executed?

Contrary to other operating systems, *NIX features a tight control system and runs a couple of checks before a program gets executed. The system checks first whether the user, or the calling process for that matter, has the right to execute the program, if it is marked as *executable* at all. If this check goes through and the program tries to create a file, the system makes sure the user has the right to create that file in this very location. The same holds true for creating files in random access memory (RAM). Given this chain of security checks, it is almost impossible to overwrite important files belonging to the system or another user. If you are logged in as `root`, some of the restrictions don't apply, though. So, if you execute a program or script while logged in as `root`, make sure the outcome is what you intended to do in the first place—especially with commands like `rm` (remove), which, when executed in the superuser context, rely on your judgment and proceed without asking.

NOTE *Even if some seasoned *NIX administrators will tell you otherwise, the author has seen this firsthand—some *NIX brands don't mind if someone tries to remove the kernel from the hard disk! To be precise, they don't mind as long as the system will not be rebooted. Admittedly, the author saw this happen on an older system, but still, there was no warning; the kernel was simply removed. Double-check while working as root.*

2.1.2 What Is #!?

When browsing executable files like shell or Perl scripts, you inevitably encounter lines like `#!/bin/sh` or `#!/usr/bin/perl -w`. Take a look at Listing 2.1.

LISTING 2.1 The First Line of an Executable Script

```
erikk@unixhost:> more netatalk-config
#!/bin/sh
# netatalk-config
af_libs=
af_cflags=
prefix=/usr
exec_prefix=${prefix}
...
```

These lines occur only in line one of the script and are called *shebangs*. If a file containing text is marked as executable, line one should contain the interpreter, including the path, needed to run the file as a program. If this prerequisite is fulfilled, the file can be executed either from the command line or launched from the window manager of choice. This could be a portability issue, though; for example, if the program `perl` is not in the `/usr/bin` directory on a system, the script will not execute, for obvious reasons. The shebang is a shortcut for typing `perl myscript.pl`, thus enabling scripts to behave like programs.

2.1.3 Why Use `./`?

If you want to make sure to launch the right program or script, it is always advisable to use either a full pathname like `/usr/bin/ls` or the shortcut `./<program>` if the current location of `$PWD` is already the directory the command or script resides in—for example, `./ls`. `ls` is obviously just an example, but just in case you try to create a better version of the command, it makes sense to control which one is executed. While you are testing the new program or script, chances are that the current directory is already the directory it was built or written in. The shortcut `./` implies using the current path (`$PWD`) to search for the program to be executed, instead of traversing the contents of `$PATH` to find the executable named on the command line. Another use of the shortcut is to make use of untainted binaries executed from another location like a CD-ROM while working on a compromised or malfunctioning system.

2.2 SHELL SCRIPTS STEP BY STEP

As mentioned earlier, *NIX consists of small programs chained together to achieve the desired goal. Many programs in the OS are actually scripts, not binaries. This is possible due to the "do one thing right, expose a defined interface" paradigm and enables the administrator to customize the system to fit certain needs. This section of the chapter deals with the process of writing scripts. It presents a simple starting point, which evolves over time. The beauty of the *NIX approach is to *start simple and go to sophisticated if needed*. This approach enables the writer of the script to test the basic functionality, thus laying a solid foundation before extending the capabilities of the script. The script grows in manageable steps, allowing the writer to tackle the task at hand based on existing experience and knowledge.

2.2.1 Creating the Script

Step one would be to create a directory for storing and testing the scripts to be created (see Listing 2.2). Depending on your needs, creating a tree of directories for your different projects would be a good idea. Again, you don't have to follow these examples verbatim; just create something that is usable for you and reflects your train of thought. Remember, though, a directory tree containing all the valuable work is easier to back up than countless directories scattered all over the hard disk. See Chapter 4 for scripts to automate the process.

TIP *One word of advice—use a "normal" account for development work, unless it is absolutely necessary to switch to superuser. An account with restricted rights on the system prevents possible damage to it. Some commands have destructive side effects, and it is better to get an error message than to re-install the whole system. Programs with the potential to take the computer down should be tested on a test system, which resembles the target platform as close as possible. If the test system needs to be reinstalled, it is a lesson learned; if a production server needs to be reinstalled, that's a far more serious matter.*

LISTING 2.2 A Directory Structure for Development 1

```
erikk@unixhost:> mkdir -p work/df_check
erikk@unixhost:> ls -lR work
work:
total 2
drwxr-xr-x   2 erikk    staff        512 Jun 29 10:49 df_check

work/df_check:
total 0
```

The structure shown in Listing 2.2 is rather simplistic. There is a directory called `work`, which contains a directory named `df_check`. Depending on the needs, this structure could be sufficient. A more structured approach is shown in Listing 2.3.

LISTING 2.3 Directory Structure for Development 2

```
erikk@unixhost:> mkdir -p work/df_check/RCS work/df_check/doc
erikk@unixhost:> touch work/df_check/ToDo.txt
erikk@unixhost:> ls -lR work
work:
total 2
```

```
drwxr-xr-x   4 erikk    staff        512 Jun 29 11:29 df_check

work/df_check:
total 6
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:05 RCS
-rw-r--r--   1 erikk    staff          0 Jun 29 11:29 ToDo.txt
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:05 doc

work/df_check/RCS:
total 0

work/df_check/doc:
total 0
```

Now there are two more directories, `doc` to hold the documentation if necessary and `RCS` for version control. We'll talk more about versioning later. The file `ToDo.txt` will be used as a scratch pad of things to be taken care of at a later time (the reason to use a capital `T` in the filename is simple; because the files are sorted alphabetically, if you use a capital letters, those files will show up on top of a listing created by `ls`).

Choose an Editor

Which text editor should you use? There is a simple answer: the one you feel comfortable with. A more complex answer would be as follows: the right tool for the job at hand that you feel comfortable with. *NIX, as always, puts you into the position to choose from more than one alternative. The more common choices would be `vi` or `emacs`, followed by a dozen other programs. Depending on who you ask, you'll be swamped with reasons why the favorite editor of the person asked is the best choice.

The question about the best editor can spark a lively discussion among programmers and system administrators. Simply try to imagine a group of physicists confronted with the sentence "It is possible to travel faster than light!" Their inevitable reaction is nothing compared to the vehemence with which people defend their editor.

We're using both `vi` and `emacs` for reasons that probably don't apply to you: `vi` is available on every *NIX-based system. If you are not a hundred percent sure `emacs`, or whatever editor you have chosen for life, is available on every system you are responsible for, try to at least maintain a certain knowledge of `vi`. You will need it in due time.

→

This book was written with `emacs`.
If your time permits, try to experiment with as many editors as you like and choose the best candidate. Just make sure your choice is available on every system you administer, regardless of operational status. And if you should be bored at a *NIX-related conference, mention your editor of choice.

Now, the file `ToDo.txt` will be put under version control, although it is not absolutely necessary. The file needs the contents shown in Listing 2.4 to enable versioning.

LISTING 2.4 Contents of `ToDo.txt`

```
erikk@unixhost:> head ToDo.txt
$Id$
- read in output of 'df -h' and process it
```

`ToDo.txt` is a simple text file, apart from the odd-looking first line. `$Id$` is used for version control; when checked in, it will be replaced with the version number and some other information. Although this first line is not a necessity to make use of version control, it allows the user to recognize the fact that it was put under version control and displays the current version. The necessary steps to put the file under version control are shown in Listing 2.5.

LISTING 2.5 Putting `ToDo.txt` under Version Control

```
erikk@unixhost:> ci -l ToDo.txt
RCS/ToDo.txt,v <-- ToDo.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> start
>> . initial revision: 1.1
done
erikk@unixhost:> head ToDo.txt
$Id: ToDo.txt,v 1.1 2004/06/29 09:44:31 erikk Exp erikk $
- read in output of 'df -h' and process it
erikk@unixhost:> ls -l RCS
total 2
-r--r--r--   1 erikk    staff        245 Jun 29 11:44 ToDo.txt,v
```

The command `ci -l ToDo.txt` puts the file under version control. `ci` is a moniker for "check in." The option `-l` tells `ci` to "lock" the file and keep it where it is; otherwise, it would create the file ToDo.txt,v in the directory RCS and delete the original. To get it back, you would have to execute the command co ToDo.txt.

The latter approach is fine while working with a team of programmers or fellow system administrators, but for the purpose of developing for yourself, it is easier to keep the files handy. The moment the file is "unlocked," it becomes writable for other programmers, provided they have access to the RCS directory (if one of those programmers chooses to work with the file, the file is properly locked again while that programmer works on it). This mechanism allows multiple programmers to work on the same files without overwriting changes introduced by others. If one is not able to obtain a lock on a file, that file is usually locked by another programmer for editing.

As shown in Listing 2.5, the string $Id$ is replaced with `$Id: To- Do.txt,v 1.1 2004/06/29 09:44:31 erikk Exp erikk $`. It contains the name of the RCS file, the current version number, 1.1, the date it was checked in last, and the username of the person executing the check-in.

Now you are ready to start the creation of the script. Insert the contents shown in Listing 2.6 into the file df_check.sh. Using an extension to the filename is not a real requirement in *NIX, but it can help you remember which interpreter is used to execute the file.

LISTING 2.6 Creating the First Version of the Script df_check.sh

```
#!/bin/sh
#
# $Id$
##########################################################
#
#       file:           df_check.sh
#
#       purpose:        script to monitor the status
#                       of mounted volumes
#
#       author:         Erik Keller
#
#       parameters:
#
##########################################################

df -h

# EOF
```

NOTE

Use a common language. This tip is important if your company operates in more than one country. All comments and messages in the scripts should be either in a language understood by all system administrators or in all the necessary languages to convey the information.

The author used to work for an American company in the Munich, Germany, office. The head office was located in Los Angeles, California; another branch was situated in Japan. The system administrators took rotating shifts to watch over the servers in the branch offices during the respective off-work hours of the other administrators. Usually all messages coming from the scripts were supposed to be in English, but once it took the author about an hour to decipher a message sent by a server in Japan. If your Japanese is rusty, try to avoid situations like this.

All characters after a pound sign (`#`) are considered comments in a shell script. Use this to your advantage; write comments about what the script is supposed to do and comment the code as well. If the programs are growing, their complexity level usually grows as well. You will thank yourself the next time—around 3:00 A.M.—that you wrote the comments; believe me. Whether you use the scheme presented in these examples or come up with your own does not matter. What matters is that you use whatever suits you and try to stick to it. The last line `# EOF` is not a requirement of `sh`. It is used as a marker, signaling that the file seems to be complete. So if you are using more than one programming language, you can choose to either start this habit or not.

The file is potentially executable, due to the use of the shebang. The OS just does not know about this, yet. A common mistake is to put the file in version control now. However, if you did, you would have to set the execute bit every time the file is checked out again. The version control mechanism restores the permissions of the file; thus, the file will be marked executable only if this were the case while it was checked in for the first time (you can change this at a later time, but setting it right from the beginning just makes more sense). Listing 2.7 shows the needed steps.

LISTING 2.7 Making the File Executable

```
erikk@unixhost> ls -l
total 8
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:44 RCS
-rw-r--r--   1 erikk    staff        103 Jun 29 11:44 ToDo.txt
-rw-r--r--   1 erikk    staff        276 Jun 29 12:07 df_check.sh
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:05 doc
erikk@unixhost> chmod u+x df_check.sh
erikk@unixhost> ls -l
```

```
total 8
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:44 RCS
-rw-r--r--   1 erikk    staff        103 Jun 29 11:44 ToDo.txt
-rwxr--r--   1 erikk    staff        276 Jun 29 12:07 df_check.sh
drwxr-xr-x   2 erikk    staff        512 Jun 29 11:05 doc
```

The first output of `ls` shows the file `df_check.sh` has read/write permissions for the owner, `erikk` and read-only permissions for the group `staff` and all other users on the system. To change that you use the command `chmod`. The parameter `u+x` reads as follows: set the execute bit for the user that owns the file. That means the owner of the file, `erikk`, will be allowed to execute it. To grant this right to the group and all other users, you would have to use `g+x,o+x`.

If the script is executed now, it will output a list of the mounted volumes.

```
erikk@unixhost:> ./df_check.sh
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0t3d0s0      1.5G   1.0G   402M    73%    /
/proc                    0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
fd                       0K     0K     0K     0%    /dev/fd
/dev/dsk/c0t3d0s1      481M   336M    97M    78%    /var
swap                   545M    40K   545M     1%    /var/run
swap                   545M   312K   545M     1%    /tmp
/dev/dsk/c0t3d0s7      1.4G   776M   637M    55%    /export/home
```

The output contains data you are not interested in. To filter the output, you could use the programs `grep` and `awk`. With a little help of `grep`, the first line, containing the column headers, disappears from the output, as shown in Listing 2.8.

LISTING 2.8 Filtering with `grep`

```
erikk@unixhost:> df -h | grep %
/dev/dsk/c0t3d0s0      1.5G   1.0G   402M    73%    /
/proc                    0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
fd                       0K     0K     0K     0%    /dev/fd
/dev/dsk/c0t3d0s1      481M   336M    97M    78%    /var
swap                   545M    40K   545M     1%    /var/run
swap                   545M   312K   545M     1%    /tmp
/dev/dsk/c0t3d0s7      1.4G   776M   637M    55%    /export/home
```

As shown in Listing 2.8, `grep` is used to eliminate all lines *not* containing the character `%`. Now your threshold for getting nervous about the capacity of your volumes needs to be defined. This example uses 70% for demonstration purposes; your threshold might be higher, but this value gives some leeway to act without being too much under pressure. The next example uses `awk` to further filter the output you got from `grep`, as shown in Listing 2.9.

LISTING 2.9 Filtering with `grep` and `awk`

```
erikk@unixhost:> df -h | grep % | awk '$5 >= 70 {print $6,$5}'
/ 73%
/var 78%
```

Now the names and the *percentage used* of all volumes greater than 70% are shown. `awk` is used to inspect the values in column five; because the value is greater than 70%, the contents of column six and five are printed. The program uses the so-called whitespace—meaning blanks or tab characters—to differentiate between the columns or fields. This chain of commands seems to do the job; let's put it in the script, as shown in Listing 2.10.

LISTING 2.10 Using the Commands in the Script

```
erikk@unixhost:> cat df_check.sh
#!/bin/sh
#
# $Id$
##########################################################
#
#       file:           df_check.sh
#
#       purpose:        script to monitor the status
#                       of mounted volumes
#
#       author:         Erik Keller
#
#       parameters:
#
##########################################################

df -h | grep % | awk '$5 >= 70 {print $6,": ", $5}'

# EOF

erikk@unixhost> ci -l df_check.sh
RCS/df_check.sh,v  <--  df_check.sh
enter description, terminated with single '.' or end of file:
```

```
NOTE: This is NOT the log message!
>> check df output for volumes filled > 70%
>> ^D
initial revision: 1.1
done
```

A colon is inserted to render the output more readable; because this script is already working, it is put under version control.

2.2.2 Using Variables

The script works without problems—on the system on which it was created and tested. It should work on similar systems and even on different flavors of *NIX, but you have to make sure that the needed programs are found via the `$PATH` variable. `awk` especially could cause some confusion, because there are two versions of the program on most modern systems: the original one, created a while ago, and a newer one called `nawk`. The latter knows how to define functions, for example. You shouldn't have any problems if the program is used in the aforementioned way, but still, you should be aware of the fact. To find out, simply try to start the program `nawk` and watch the output. If it looks like the following, the newer version is available on the system.

```
Usage: nawk [-f programfile | 'program'] [-Ffieldsep] [-v var=value]
[files]
```

To ensure the script uses the intended program, you should use the complete path. Typing `/usr/bin/awk` every time a program is used in a script is prone to error, though. So, to create a single entry that can be changed when needed, you can use a variable to store the path and the name of the program. Listing 2.11 shows how to do it.

LISTING 2.11 The `df_check.sh` Script Using Variables

```
#!/bin/sh
## $Id: df_check.sh,v 1.1 2004/08/03 10:25:34 erikk Exp erikk $
##########################################################
#
#       file:           df_check.sh
#
#       purpose:        script to monitor the status
#                       of mounted volumes
#
#       author:         Erik Keller
```

```
#
#       parameters:
#
##########################################################

# Variables
DF="/usr/bin/df"
GREP="/usr/bin/grep"
AWK="/usr/bin/awk"

$DF -h | $GREP % | $AWK '$5 >= 70 {print $6,": ", $5}'

# EOF
"df_check.sh" 23 lines, 451 characters

erikk@unixhost> ./df_check.sh
/ :  73%
/var :  78%
erikk@unixhost> ci -l df_check.sh
RCS/df_check.sh,v  <--  df_check.sh
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> added variables for executables
>> ^D
done
```

As shown in Listing 2.11, the variables for all used programs are created. When you are creating variables, you might encounter a couple of stumbling blocks. First, please note that the `$` character is not used while defining the variable—only when accessing the contents. Second, another stumbling block occurs if you mistakenly use blanks around `=` in the definition. If there are blanks, the variable won't contain any values ("use a blank to create blank variables" the author's teacher used to say). After a test, the program is checked into version control again to record the changes made to it. Be as descriptive as possible regarding the log message. You will appreciate this when trying to figure out what was changed and why at a later time. To list the existing versions, you use the command `rlog` , as shown in Listing 2.12.

LISTING 2.12 Using `rlog` to List the Existing Versions

```
erikk@unixhost:> rlog df_check.sh
RCS file: RCS/df_check.sh,v
Working file: df_check.sh
head: 1.2
```

```
branch:
locks: strict
        erikk: 1.2
access list:
symbolic names:
keyword substitution: kv
total revisions: 2;     selected revisions: 2
description: check df output for volumes filled > 70%
filtering complete
----------------------------
revision 1.2    locked by: erikk;
date: 2004/08/03 10:29:54;  author: erikk;  state: Exp;  lines: +7 -2
added variables for executables
----------------------------
revision 1.1
date: 2004/08/02 15:00:15;  author: erikk;  state: Exp;
Initial revision
=======================================================================
======
erikk@unixhost:>
```

`rlog` used without any options displays the revision history of the queried file and the data and settings used to maintain the revisions. Revisions are listed from "new" to "old" including who checked the revision in and when. The interesting parts here are the log messages. The latest revision (`1.2`) was checked in by `erikk` and the log entry (hopefully) explains what was changed and why. How elaborate these entries are is up to the writer of the comments; it is even possible to leave it blank. For practical reasons, the log entries should enable you to understand what happened—make sure you will get the gist with a glance.

If more than one person is allowed to modify the script or one is just interested in the last change, `rlog` can be used with the option `-r`. Listing 2.13 describes the process.

LISTING 2.13 Displaying the Last Modification Using `rlog`

```
erikk@unixhost:> rlog -r df_check.sh
RCS file: RCS/df_check.sh,v
Working file: df_check.sh
head: 1.2
branch:
locks: strict
        erikk: 1.2
access list:
```

```
symbolic names:
keyword substitution: kv
total revisions: 2;     selected revisions: 1
description: check df output for volumes filled > 70%
filtering complete
----------------------------
revision 1.2    locked by: erikk;
date: 2004/08/03 10:29:54;  author: erikk;  state: Exp;  lines: +7 -2
added variables for executables
=============================================================================
erikk@unixhost:>
```

The `-r` (revision) option is intended to give access to a certain revision, and if no specific revision is supplied, the information regarding the last check-in is displayed. To query a specific revision, for example, revision `1.1`, the command would be used like so: `rlog -r1.1`.

If you intend to use the script on another *NIX system, variables enable you to keep the needed changes to a minimum. This doesn't seem to be necessary for the script at hand, but scripts tend to grow in complexity over time. Utilizing variables is one way to enhance portability and maintainability. Nobody forces you to use variables to call the needed programs. It is just good practice; after all, then you have only one line to change if needed.

2.2.3 Extending the Script

The script is now ready to be deployed on all systems where its functionality is useful. The drawback is that you are required to log into every system, run the script, and act on the output. If the script would run automatically at predefined times and write the output to a file, the task would be definitely simpler. Listing 2.14 extends the functionality of the script by having the output written to a file.

LISTING 2.14 `df_check.sh` Output to File

```
#!/bin/sh
#
# $Id: df_check.sh,v 1.3 2004/08/05 08:18:18 erikk Exp erikk $
##########################################################
#
#       file:           df_check.sh
#
#       purpose:        script to monitor the status
#                       of mounted volumes
```

```
#
#       author:         Erik Keller
#
#       parameters:
#
##########################################################

# Variables
DF="/usr/bin/df"
GREP="/usr/bin/grep"
AWK="/usr/bin/awk"
SYSNAME=`/usr/bin/uname -n`
REPORT=/export/home/users/erikk/work/df_check/${SYSNAME}_dfrep.txt

echo Systemname: $SYSNAME > $REPORT

$DF -h | $GREP % | $AWK '$5 >= 70 {print $6,": ", $5}' >> $REPORT

# EOF
```

Obvious additions in Listing 2.14 are the variables `$SYSNAME` and `$REPORT`. `$SYSNAME` contains the name of the system the script is run on by redirecting the output of `uname -n` using back ticks. Its primary use is to create a descriptive name for the resulting file containing the report. That filename (including the path) is stored in the second variable aptly called `$REPORT`. To deploy the script on another system, changing the variables that point to the programs and adjusting the path for the file to be created is sufficient.

However, one issue lingers; the script is functional but not very readable. Shouldn't that be addressed, as well?

Listing 2.15 shows how the script could look when fit for deployment.

LISTING 2.15 Final `df_check.sh` Script

```
#!/bin/sh
#
# $Id: df_check.sh,v 1.3 2004/08/05 08:18:18 erikk Exp erikk $
##########################################################
#
#       file:           df_check.sh
#
#       purpose:        script to monitor the status
#                       of mounted volumes
#
```

```
#       author:         Erik Keller
#
#       parameters:
#
##########################################################

# Variables
# Used programs
DF="/usr/bin/df"
GREP="/usr/bin/grep"
AWK="/usr/bin/awk"

# OS info
SYSNAME=`uname -n`
OSNAME=`uname -s`
OSVERSION=`uname -r`
OSBUILD=`uname -v`

# destination
REPORTPATH=/export/home/users/erikk/work/df_check
REPORTFILE=${SYSNAME}_dfrep.txt
REPORTDEST=${REPORTPATH}/${REPORTFILE}

# Report header
echo Systemname: $SYSNAME > $REPORTDEST
echo Operatingsystem: $OSNAME \
                      $OSVERSION \
                      $OSBUILD >> $REPORTDEST
echo ------------------------- >> $REPORTDEST

# Diskfree report
$DF -h | $GREP % | \
    $AWK '$5 >= 70 {print $6,": ", $5}' \
    >> $REPORTDEST

# EOF
```

As you can see, the script is now easier to customize and easier to read. The path and the filename are separated to avoid mistakes made while changing one of them, and the commands are sliced into smaller chunks using the continuation character `\` (don't forget: there can't be any characters between `\` and the carriage return (CR); otherwise, the continuation will not work). The new information regarding the name of the operating system (`$OSNAME`), its version information (`$OSVERSION`), and the build details (`$OSBUILD`) are not essential to the deployment of the script, but could be of use in a more heterogeneous environment. Listing 2.16 shows the output on a Solaris system.

LISTING 2.16 Output of `df_check.sh`

```
Operatingsystem: SunOS 5.9 Generic_112233-11
------------------------
/ : 73%
/var : 79%
```

This script can be used as a template for monitoring other parts of a system because it can be easily extended. If you decide to run it once a day using the `cron` mechanism (see Section 2.4.2, "The `cron` Mechanism," later in this chapter for more information), the status of the filesystems, regarding available space, is always available.

2.3 PERL SCRIPTS

Perl is short for "Practical Extraction and Reporting Language." It is included with most of the *NIX distributions nowadays. Some of them actually rely on the existence of Perl to tackle administrative tasks. Perl was originally created to provide a means to extract and format information found in larger text or log files in an efficient manner. Regardless of its origins, Perl started to evolve into a full-blown programming language; even object-oriented programming techniques are supported.

One aspect that makes Perl unique is its simplicity; it allows you to build handy tools with unmatched ease. Perl feels different when compared to other scripting or even programming languages. Perl was created in a way that allows you to complete easy tasks in an easy way without preventing you from tackling more complicated jobs with it. The most important saying in Perl is this: if the program does its job, it is a good program. Using Perl requires only rudimentary experience in another programming language. If you worked your way through the shell-scripting example, you should be set. If you are a seasoned programmer and you are anxious to learn Perl, you will find many familiar functions in it.

2.3.1 Creating a Perl Script

The following Perl script will output a list of all users and their respective `UID`s. Start by creating another subdirectory in your project hierarchy; the one used for the examples is called `ulist1`. Inside this directory, create the `RCS` directory and a file named `ulist1.pl`, according to the following description in Listing 2.17.

LISTING 2.17 Creating the Structure for the Perl Example

```
erikk@unixhost:> pwd
/export/home/users/erikk/work
erikk@unixhost:> mkdir ulist1
erikk@unixhost:> mkdir ulist1/RCS
erikk@unixhost:> cd ulist1
erikk@unixhost:> ls -l
total 2
drwxr-xr-x   2 erikk    staff        512 Aug  9 16:56 RCS
erikk@unixhost:> touch ulist1.pl
erikk@unixhost:> chmod u+x ulist1.pl
erikk@unixhost:> ls -l
total 2
drwxr-xr-x   2 erikk    staff        512 Aug  9 16:56 RCS
-rwxr--r--   1 erikk    staff          0 Aug  9 16:57 ulist1.pl
erikk@unixhost:>
```

Up to now, the procedure is the same as in the shell example. However, in this case, you will have to determine the path to the Perl interpreter. To find the exact location of programs in the `$PATH`, you can use the `which` command. `which perl` should return something similar to `/usr/bin/perl`. The contents shown in Listing 2.18 need to be inserted into the file.

LISTING 2.18 First Version of the Perl Example

```
#!/usr/bin/perl
# $Id$
#
#################################################
#
#     file:          ulist1.pl
#
#     purpose:       lists all users
#                    and their respective IDs
#
#
#     author:        Erik Keller
#
#################################################
```

```
# Variables

# list passwd first
# open the file
open(MY_PWFILE, "/etc/passwd") or die
     "$0: couldn't open file: $!";

# loop through the file and print the records
while(<MY_PWFILE>)
{
     print $_;
}

# close the file
close(MY_PWFILE);

# EOF
```

If you have never seen a Perl program before, the following list contains a short explanation of what is going on.

- `open(MY_PWFILE, "/etc/passwd")` instructs the program to open the file `/etc/passwd` and attaches it to the `Handle MY_PWFILE`. A Handle points to the location in the computer's memory where the file is stored. The program will use it later to access the file. The continuation of the line, or die `"$0:` couldn't open file: `$!";`, tells the program to stop executing in case of an error and to print out the following information: $0 contains the name of the program and `$!` the description of the error.
- The instruction `<MY_PWFILE>` is the living proof of how to make things simple in Perl. It means, "as long as you didn't reach the end of the file the Handle is pointing to." So `while(<MY_PWFILE>)` steps through the file, line by line, until the end of it is reached and executes the commands between the curly braces (`{}`).
- `print $_`; outputs the current record without changing it. The variable `$_` is created automatically and contains the current record.
- `close(MY_PWFILE)`; does exactly what it says—it closes the Handle and the file it points to. Make sure to always close the opened files in the program that used them; relying on the operating system to do so could potentially damage the files or create hard to find error conditions.

If you execute the program, the output should display the lines contained in the `/etc/passwd` on your system (the output shown in Listing 2.19 is shortened for this example).

LISTING 2.19 Output of the Perl Script

```
erikk@unixhost:> ./ulist1.pl
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
smmsp:x:25:25:SendMail Message Submission Program:/:
listen:x:37:4:Network Admin:/usr/net/nls:
nobody:x:60001:60001:Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
erikk:x:101:10::/export/home/users/erikk:/bin/sh
```

The result in Listing 2.19 obviously lacks clarity. It contains information you don't need for the purpose of the program, for example, which shell will be started after the login of the respective user. A colon separates the fields, and the fields containing the information you are looking for are field 1 and field 3. To access the contents of these fields, Perl provides the function `split`. It requires two parameters: the field-separator, in this case:, and the name of the variable to be split up, `$_`, as shown in Listing 2.20.

LISTING 2.20 Splitting the Records into Fields

```
#!/usr/bin/perl
# $Id$
#
#################################################
#
#     file:         ulist1.pl
#
#     purpose:      lists all users
#                   and their respective IDs
#
#
#     author:       Erik Keller
```

```
#
#################################################

# Variables


# list passwd first
# open the file
open(MY_PWFILE, "/etc/passwd") or die
     "$0: couldn't open file: $!";

# loop through the file and print the records
while(<MY_PWFILE>)
{
     chop;
     @my_records = split(":",$_);
     print "@my_records\n";
}

# close the file
close(MY_PWFILE);

# EOF

erikk@unixhost:> ./ulist1.pl
root x 0 1 Super-User / /sbin/sh
daemon x 1 1  /
bin x 2 2  /usr/bin
sys x 3 3  /
adm x 4 4 Admin /var/adm
lp x 71 8 Line Printer Admin /usr/spool/lp
uucp x 5 5 uucp Admin /usr/lib/uucp
nuucp x 9 9 uucp Admin /var/spool/uucppublic /usr/lib/uucp/uucico
smmsp x 25 25 SendMail Message Submission Program /
listen x 37 4 Network Admin /usr/net/nls
nobody x 60001 60001 Nobody /
noaccess x 60002 60002 No Access User /
nobody4 x 65534 65534 SunOS 4.x Nobody /
erikk x 101 10  /export/home/users/erikk /bin/sh
```

The function `chop` removes the character signaling the end of the line. If called without parameters, `chop` acts on the contents of `$_`. Even if it isn't that obvious for now, `split` actually extracted the information contained in the fields. To read this information, the following construct is used: `$my_records[0]` accesses the contents of the first field. The numbering of the array fields starts with 0, just like in most other programming languages. This means the first field is on address 0, the second field on address 1, and so forth. Thus, the information you are looking for is in field 0 and field 2. Listing 2.21 presents the script and its output.

LISTING 2.21 Perl Script to Extract Usernames and IDs

```
#!/usr/bin/perl
# $Id: ulist1.pl,v 1.1 2004/08/10 10:57:03 erikk Exp erikk $
#
#################################################
#
#     file:          ulist1.pl
#
#     purpose:       lists all users
#                    and their respective IDs
#
#
#     author:        Erik Keller
#
#################################################

# Variables


# list passwd first
# open the file
open(MY_PWFILE, "/etc/passwd") or die
     "$0: couldn't open file: $!";

# loop through the file and print the records
while(<MY_PWFILE>)
{
     chop;
     @my_records = split(":",$_);
     print "$my_records[0]\t$my_records[2]\n";
}

# close the file
close(MY_PWFILE);

# EOF

erikk@unixhost:> ./ulist1.pl
```

```
root    0
daemon  1
bin     2
sys     3
adm     4
lp      71
uucp    5
nuucp   9
smmsp   25
listen  37
nobody  60001
noaccess        60002
nobody4 65534
erikk   101
```

The script works. To use the resulting list in any program capable of importing comma-separated values (CSV), you simply switch the tab characters (`\t`) to a comma and that does the trick. One use of the output could be to transfer the user information into a database system to check the consistency of the used names, for example.

2.3.2 Using the `Getopt` Perl Module

As soon as your Perl programs are starting to grow in complexity, you will find one question comes to mind: how do you process arguments given on the command line? A fair share of the Perl programs in circulation use the `@ARGV` array to evaluate the supplied parameters. There is nothing wrong with that approach, but the creators of those scripts are making their lives a little bit more complicated than necessary. The Perl Standard Library, part of every Perl installation, contains a wealth of functions you can use to extend the functionality of the scripts or simply tackle certain issues of a more general nature in a standard way. You can find ready-built functions to deal with parameters, among other things.

As the author started out creating his first Perl programs, he had the privilege to get to know a couple of well-versed Perl programmers. They took the time to look at the created code and listened to the author's complaints about how hard certain things are to realize in Perl. One remark they made to the author about a fair number of the programs was this: "But that is part of the standard library, isn't it?" The author actually created complex functions to achieve certain results, not knowing that Perl would provide a solution if asked correctly.

One of these timesavers is the package `Getopt`. Utilizing it makes the processing of command-line switches like `-p` or evaluation of parameters like `-d /export/home` almost an automatic process. Listing 2.22 shows the usage of the package.

LISTING 2.22 Using `Getopt`

```
#!/usr/bin/perl
# $Id: perlgetopt.pl,v 1.1 2004/08/29 18:44:49 erikk Exp erikk $
####################################################
#
#     file:          perlgetopt.pl
#
#     purpose:       demonstrate the use of
#                    Getopt
#
#     author:        Erik Keller
#
#     parameters:    -s <string>
#
####################################################

use Getopt::Std;

getopt('s');

if(length($opt_s) == 0)
{
        print "$0: usage:\n\n";
        print "$0 -s <string>\n\n";
}
else
{
        print "-s has value: $opt_s\n";
}

# EOF
```

This example is just an illustration how easy and, most of all, how consistent the handling of command-line arguments can be if done with `Getopt`. The instruction `use Getopt::Std;` loads the `Getopt` package, and `getopt('s');` checks for the existence of the option `-s` and, if successful, stores the following string in the variable `$opt_s`. If another option, for example, `-d`, should be present instead, the length of the variable `$opt_s` will be `0`. Because `Getopt` was not instructed to evaluate any other options than `-s` and the length of `$opt_s` is tested in the `if` clause following the evaluation, the presence of the option is mandatory. In the case of a zero length, a message of how to use the program is displayed. If the length of `$opt_s` is `> 0`, the string stored in `$opt_s` is printed. This example barely scratches the surface regarding the capabilities of `Getopt`, but it can be considered as an example of how to handle command-line parameters using Perl.

As shown, Perl can be used to extract information from a variety of text-based files. This capability alone makes it a versatile tool in every system administrator's tool chest. You always have more than one way to achieve something in Perl. The beauty of the language is this—Perl adapts to you, not the other way round. Investing time to learn more about Perl is always beneficial to everyone who has the need to deal with the management of log files and the like. In short, you will find many tasks in your daily routine that can be tackled more efficiently using Perl.

2.4 AUTOMATION

Most *NIX systems are rarely shut down; they are running 24 hours, 7 days a week (24/7). Based on this fact, it is sometimes necessary to instruct the system to run a certain command or chain of commands at a predefined time and/or date. Other programs need to run on a recurring basis, for example, every Sunday. *NIX provides two mechanisms to automate things, `at` and `cron`.

2.4.1 The `at` Command

The command `at` does exactly what it says by executing things *at* a certain time and date. It can be used for recurring commands, but `cron` is better suited for this purpose most of the time. The exception to this rule would be the following situation: execution based on return values from a previous run. For example, a program is started at a certain time, does its job, and returns a value to be converted again in a certain number of hours. This value is fed into `at` again to schedule the next execution. Reorganizing a database or copying a large amount of files for safekeeping, for example, need be handled this way, because you have almost no means to calculate the exact amount of time needed for the task.

The syntax of `at` is simple enough, comprised simply of `at` and when the execution should begin. The chain of commands shown in Listing 2.23 creates a listing of all files in the `/tmp` directory at 5:00 P.M.

LISTING 2.23 Execution of a Command `at 17:00` on the Same Day

```
root@unixhost:> at 17:00
at> ls -l /tmp > /.HO/documentation/contents_tmp.txt
at> <EOT>
commands will be executed using /bin/sh
job 1087311600.a at Tue Jun 15 17:00:00 2004
```

at is started with the time the execution should begin as the parameter. After pressing Enter, you are shown the secondary prompt; here you type the command to be executed . After pressing Enter again, you use the key combination Ctrl-D to terminate the session. at answers with the shell that will be used to execute the command and the *job ID* followed by the execution time.

To list the pending jobs in the at queue, you use the command atq, as shown in Listing 2.24.

LISTING 2.24 Displaying the at Queue

```
root@unixhost:> atq
 Rank     Execution Date     Owner     Job         Queue   Job Name
  1st   Jun 15, 2004 17:00   root    1087311600.a    a      stdin
  2nd   Jun 15, 2004 20:17   root    1087323435.a    a      stdin
```

The preceding listing, taken from a Solaris system, shows that there are two pending jobs, one starting at 5:00 P.M. and one at 8:17 P.M. at interprets time in the following manner:

1. If the time used as parameter is still in the future, the command will execute today at that time.
2. If the time has already passed, the execution will start tomorrow at that time.

Both jobs are in queue a, the default if the flavor of *NIX uses a queuing system for at jobs.

Besides specifying a time and a date, at is able to use shortcuts like now, midnight, and noon as time. These can be combined with so-called *increments*, like noon tomorrow or now + 2 days; even combinations like 4pm tuesday or now + 2 months are possible. To make sure that a certain shortcut works on your system, it is always advisable to check the manpages.

To remove a job, you can use the command atrm, as shown in Listing 2.25.

LISTING 2.25 Removal of a Scheduled Job from the Queue

```
root@unixhost:> atq
root@unixhost:> atq
 Rank     Execution Date     Owner     Job         Queue   Job Name
  3rd   Jun 23, 2004 14:00   root    1087992000.a    a      stdin
  5th   Aug 15, 2004 17:07   root    1092582446.a    a      stdin
root@unixhost:> atrm 1092582446.a
1092582446.a: removed
root@unixhost:> atq
```

```
 Rank     Execution Date     Owner     Job         Queue   Job Name
  3rd   Jun 23, 2004 14:00   root    1087992000.a     a     stdin
root@unixhost:>
```

One way to submit jobs to `at` is via `stdin`. Alternatively, you can feed a file containing the commands to be executed into `at`, as shown in Listing 2.26.

LISTING 2.26 Execution of `at` Using a File

```
root@unixhost:> cat my_cmds1.txt
cd /.HO/mytest1
ls -l /tmp > tmplist.lst

root@unixhost:> at -f my_cmds1.txt now + 2 minutes
commands will be executed using /sbin/sh
job 1087395867.a at Wed Jun 16 16:24:27 2004
root@unixhost:> atq
 Rank     Execution Date     Owner     Job         Queue   Job Name
  1st   Jun 16, 2004 16:24   root    1087395867.a     a    my_cmds1.txt
  2nd   Jun 18, 2004 14:00   root    1087560000.a     a    stdin
  3rd   Jun 18, 2004 17:11   root    1087571471.a     a    stdin
  4th   Jun 23, 2004 14:00   root    1087992000.a     a    stdin
  5th   Aug 15, 2004 17:07   root    1092582433.a     a    stdin
root@unixhost:>
```

The advantage of using a file containing the commands to be executed is obvious; instead of `stdin`, the name of the command file is displayed as *job name*. Jobs supposed to run in a distant or even a not-so-distant future should be submitted via a command file, thus saving the guesswork of what the job in question will do. By using a *very* descriptive name, you reveal the purpose of the chain of commands to be executed.

However, if the jobs either are scheduled via the command line or are given a command file name that is not descriptive enough, it is possible to check the contents of all `at` jobs. Depending on the flavor of *NIX, all information will be stored in a directory, for example, in `/var/spool/cron/atjobs` on Solaris. Listing 2.27 shows the Solaris job ticket.

LISTING 2.27 Contents of a Job Ticket (Solaris)

```
# pwd
/var/spool/cron/atjobs
# cat 1087560000.a
: at job
: jobname: stdin
: notify by mail: no
export DISPLAY; DISPLAY=':0.0'
export DTSOURCEPROFILE; DTSOURCEPROFILE='true'
export DTUSERSESSION; DTUSERSESSION='root-yoda-0'
export DTXSERVERLOCATION; DTXSERVERLOCATION='local'
export EDITOR; EDITOR='/usr/dt/bin/dtpad'
export HELPPATH;
HELPPATH='/usr/openwin/lib/locale:/usr/openwin/lib/help'
export HOME; HOME='/'
export LANG; LANG='C'
export LOGNAME; LOGNAME='root'
export MAIL; MAIL='/var/mail/root'
export MANPATH; MANPATH='/usr/dt/man:/usr/man:/usr/openwin/share/man'
export OPENWINHOME; OPENWINHOME='/usr/openwin'
export PWD; PWD='/'
export SHELL; SHELL='/sbin/sh'
export TERM; TERM='dtterm'
export TERMINAL_EMULATOR; TERMINAL_EMULATOR='dtterm'
export TZ; TZ='Europe/Berlin'
export USER; USER='root'
export WINDOWID; WINDOWID='8912905'
export _; _='/usr/dt/bin/sdt_shell'
export dtstart_sessionlogfile; dtstart_sessionlogfile='/dev/null'
$SHELL << '...the rest of this file is shell input'
#ident  "@(#).proto     1.6     00/05/01 SMI"   /* SVr4.0 1.2   */
cd /.HO/mytest1
umask 22
ulimit unlimited
ls -l /tmp > /.HO/mytest1/tmplist.lst

root@unixhost:>
```

If this listing looks confusing at first, consider that there is actually more information in a Solaris job ticket than is shown here, but some lines have been deleted for clarity. Again, the contents of the file are dependent on the *NIX distribution, as is the location of the file. The commands to be executed are usually at the bottom of the file. The information in the job ticket is rarely used, but if a certain command seems to behave erratically, the information regarding the environment the command will execute in is a great aid for debugging. To get the description pertaining your flavor of *NIX, use the command `man at`. It is advisable to create a simple job and to examine the job ticket on every system to be administered; doing so will save time later if a job needs to be checked for errors. The information about the location of the job tickets should be recorded in the documentation of either the specific system or in a documentation covering the release of the operating system.

2.4.2 The `cron` Mechanism

Unlike `at`, `cron` is used to execute commands or scripts in a recurring manner. It is ideally suited for things that should happen, for example, every day at 8:00 A.M. and 6:00 P.M. or once a week on a certain day.

Every user is allowed to maintain his own `crontab`, provided the username is either listed in the file `cron.allow` or not listed in the file `cron.deny`. The location of these files, if they exist, is normally inside the `/etc` tree. If neither of these files exists, access to the `cron` mechanism can be configured on a systemwide basis, either only the superuser is allowed to use it or all users are allowed to use it. Check the manpages of your *NIX distribution if you need to find out.

2.4.2.1 The `crontab`

`cron` uses a file called `crontab` for human interaction. This means if you issue the command `crontab -e`, it opens the editor that the variable `$EDITOR` points to and allows you to modify this file. You should make sure `$EDITOR` is set to `vi`; otherwise, you will encounter the not too user friendly `ex` editor.

The format of the `crontab` file is quite simple, but as always, a little documentation helps to remember.

Minute (0–59)
Hour (0–23)
Day of the month (1–31)
Month of the year (1–12)
Day of the week (0–6 0=Sunday)
The command to execute

The `#` sign is used to give a line a "comment" status, which means `cron` will ignore the line. Using comments always makes configuration files more readable. All parameters can be used as follows to span a period: `8-18` in the hour section means do it from 8:00 A.M. to 6:00 P.M. If the command should execute every 15 minutes, a comma-separated list, for example, `0,15,30,45`, will do the job. If the program should run every hour, the asterisk `*` is used. A template like the one shown in Listing 2.28 helps you concentrate on the job instead of memorizing the order of the entries.

LISTING 2.28 `crontab` Template

```
# $Id$
#Min    Hour    dom     moy     dow0=Su cmd
#------------------------------------------------------------------
```

Given the fact that a `crontab` is just a text file, all precautions against loss, like checking it into `rcs`, can be used. The latter especially should be done on every occasion the file needs to be changed. This approach allows keeping the comments in the `crontab` short and precise and moves the detailed explanation to the log file accompanying the revision.

NOTE *Take note that* `rcs` *is the name of the package, and RCS (uppercase) is an acronym for Revision Control System.*

Using the template, a command supposed to run once a day, every day at 11:00 A.M., regardless which day of the month, which month of the year, or day of the week, would be entered as shown in Listing 2.29.

LISTING 2.29 An Entry for a Recurring Job in the `crontab`

```
# $Id$
#Min    Hour    dom     moy     dow0=Su cmd
#------------------------------------------------------------------
0       11      *       *       *       /usr/local/bin/mydaily.sh
```

If a decision is made that it is sufficient for the command to run Monday to Friday, the `dow` "day of the week" parameter should be changed to `1-5`. One word of caution, though, `cron` does not know about public holidays and the like; if the result of the program requires some sort of user interaction, for example, like changing a tape, you have to make sure that either the program is disabled on that day or someone is there to do it. The same goes for Daylight Savings Time (DST)—if the system is in a country using it, be aware of the fact that the commands scheduled during the time shifts twice a year might not execute.

The preceding syntax will work even on the oldest incarnations of `cron`. There are newer versions of `cron` available, especially on Linux or BSD systems. This "other" `cron` uses the mentioned syntax and adds three things:

- The usage of / to indicate a step value. For example, `0-23/2` means execute every other hour. The V7 ("old") way would have been as follows: 0,2,4,6,8,10,12,14,16,18,20,22. To launch the program every 10 minutes */10 would be sufficient, instead of 0,10,20,30,40,50.

- Names instead of numbers for month and day. The first three letters of the name would be used, for example, `sun` for Sunday; ranges of names, like `sun-wen`, are *not* allowed, though.
- The use of environment settings. The V7 `cron` always executes commands within `/bin/sh` and sends e-mails to the owner of the `crontab`, if the called command returns something other than `0`. Here the environment can be modified by using a name = value pair; for example, `MAILTO = jdoe` would send all resulting messages to the user `jdoe` on the system.

2.4.2.2 The `cron` Background Process

Depending on the build of `cron`, either you have no options to consider ("old" `cron`) or you can set one of the two options `-s` or `-o` ("new" `cron`). `-s` enables `cron` to deal with the DST switches. Programs intended to run every hour will execute as expected, but should the hour of execution "vanish" due to the switch, execution of the program will take place at the scheduled time plus one hour. The `-o` option mimics the behavior of the "old" `cron`. The fact that this choice exists in the first place is another example of the *NIX philosophy not to break existing programs or scripts. If the programs are able to handle the switch to and from the Daylight Savings Time themselves, then the `–o` option enables them to continue to do so, instead of requiring the programmers to adapt all the scripts and programs to the new behavior of `cron`.

2.5 SUMMARY

This chapter discussed the topic of executable programs in *NIX, how and when a program or script is executed at all, and how to convey the needed interpreter to the operating system if one is needed to execute a script. It demonstrated the creation of shell and Perl scripts and how to maintain versions of those scripts. It talked about directory structures to store the scripts and accompanying files during the development process. It closed with a discussion about the two mechanisms provided by *NIX that allow the scheduling of scripts and programs to be executed at a later time and/or date.

This page intentionally left blank

3 Maintenance

In This Chapter

Maintenance is the ongoing process of keeping the system running smoothly to make sure the users get most out of the money spent for the machine in question. The task of *baselining* the various counters inside the system is explained in Section 3.1, "What Is Baselining?", the task of keeping the user accounts in good shape is covered in Section 3.2, "Users and Groups," and the actions that require constant care on a daily basis are described in Section 3.3, "Daily Tasks." This chapter lists the more common things a system administrator has to deal with regularly.

3.1 WHAT IS BASELINING?

Baselining is the process of sampling a variety of counters the system provides. The name stems from the idea that the samples taken during normal operation provide a *base* to decide if something might be wrong with the machine. Sampling this type of data should start right after the machine is set up and continue during the lifespan of the system.

3.1.1 Why Baselining?

There are many advantages to taking constant measurements of a productive system. Naturally, the most obvious is the fact that a deviation of one or more values from the "norm" points to a possible issue with the machine. Baselining provides the system administrator with the necessary foundation to discover the deviation in the first place. Every system tends to change over time, with higher workloads, more users, etc. If a solid foundation of baselines exists, the one in charge of keeping the system up and running (you) will be able to determine if the machine is just "loaded"; if the machine needs a hardware upgrade, for example, another CPU or more RAM; or if the system shows any erratic behavior that needs to be taken care of. Using the data over time even yields a good base to convince the person(s) in charge of the IT budget that the time has come to invest money in the system(s) to keep up with user demands or increased workloads.

Also, a nice side effect is that the system administrator is able to figure out if the system is really "slow" or if the user complaining about the "slowness" of the machine is simply under pressure to finish some work in time. In the latter case, try to find out how to help the user and rest assured the machine is doing what it is supposed to be doing right now.

One last thing to cover before we dive into the work: even if the sampling of the data imposes a slight performance hit on the system, you should try to generate the logs and graphs every time a sample is generated. If the machine should crash, it might still be possible to recover this data and get a clue what happened.

3.1.2 What to Baseline

Some people consider making the decision as to what needs to be baselined a "black art," information passed on by the ones in the know during strange rituals or by pure coincidence. The fact is that, depending on the tasks that a system needs to fulfill to justify its existence, the data that should be sampled in any given situation differs greatly. Some counters are needed for every computer; others are related to a special purpose or a certain business case. The values to be monitored discussed in the following sections are of general nature; this means they should be monitored regardless of the system's purpose.

3.1.3 How Long Should the Data Be Kept?

How long should you store the sampled data? The author's point of view is simple: it depends. Measurements taken before a significant hardware upgrade are of limited use after a while. Its lifespan usually lasts as long as it takes to justify the cost involved with the upgrade. Data regarding the amount of free space on the hard disks won't be needed after the available storage space is increased. To stay on the safe side, however, you should write the log files and graphs to a CD-ROM or DVD to make sure you are able to access them, just in case.

3.1.4 Directory Structure for Baselining

Because the following scripts are used for a common cause—gathering data to create statistics spanning a longer time frame—storing them in a directory structure suitable for easy maintenance and backup is a good idea. For the purposes of this book, the structure shown in Listing 3.1 will be used.

LISTING 3.1 Directory Structure to Store Baseline Scripts and Data

```
baselines
     base_cpu
     base_disk
     base_netconn
     ...
     var
          dirwatch
          ...
     html
          images
```

Below the main directory called `baselines`, directories for the different types of data gathering are created, for example, `base_cpu` contains the scripts used to sample CPU data. The directory `var` will contain the created log files; the log files created by the `dirwatch` script will be written to their own directory and another hierarchy of subdirectories below, if needed, because this script possibly creates a large number of them. The `html` directory will contain HTML files used for presentation of the gathered samples and the created statistics, and `images` will hold the generated graphs. Again, this is the structure used for the examples in this book; there is no right or wrong structure. The goal should be to create a structure that mimics the train of thought of the maintainer, the system administrator using it.

Some solutions in that respect generate more work than necessary, though. Some scripts will adhere to the latest paradigms propagated by computer science or some other approach fashionable at the moment, neither of which has been tested during a crisis. There is a principle that should be applied to all scripts and programs regarding administration of computers, *KISS*—keep it simple! The most elegant object-oriented Perl script is completely useless, if you need about five minutes to figure out what it will do if launched.

3.1.5 Monitoring Disk Space

Every *NIX system comes with the commands `df` and `du`. They are used to display the percentage of how much space is used on every partition (`df`) and, among other things, to find out about the current size of a directory (`du`).

The script in Listing 3.2 records the available space on filesystems periodically and stores the data to allow you to analyze it later.

LISTING 3.2 Registering Free Hard Disk Space

```
#!/usr/bin/perl
# $Id$
##################################################
#
#     file:           base_df.pl
#
#     purpose:        display filesystems and
#                     used space
#
#     author:         Erik Keller
#
#     parameters:     none
#
##################################################

#Variables
$varpath="/export/home/users/erikk/work/baselines/var";

# get output of df -k
open(MY_DF, " df -k |") or die "$0: couldn't get df: $!";
while(<MY_DF>)
{
        chop;
        if(/^\/dev\//)
        {
                @myrec = split(" ", $_);
                $free = int($myrec[3]/1024);
                #print "$myrec[5]: $myrec[4] $free MB free\n";
                $fsname = $myrec[5];
                $fsname =~ s/\//_/g;
                $logname = "df_".$fsname.".log";
                open(MY_LOG, ">>$varpath/$logname")
                        or die "$0: problem opening file: $!";
                print MY_LOG "$free\n";
                close(MY_LOG);
        }
}

close(MY_DF);

# EOF
```

This script analyzes `df`'s output and extracts all filesystems found on a device. A file for every filesystem found is created in the directory `$varpath` points to and named after the mount point the filesystem is located at, and the amount of available space is written to that file. The filename is constructed based on the following scheme: `/export/home` becomes `_export_home`, and the string `df_` precedes the name because filenames starting with / are not allowed in *NIX. If there is another hard disk mounted, all filesystems found on it will be evaluated as well.

The fact that a filesystem starts to fill up does not necessarily allow you to pinpoint the reason or the origin for that behavior. A script to record changes to single directories or directory trees is needed; a possible approach to this kind of script is shown in Listing 3.3.

LISTING 3.3 Script to Monitor Directories

```
#!/usr/bin/perl
# $Id: dirwatch.pl,v 1.4 2004/08/23 15:56:25 erikk Exp erikk $
##################################################
#
#     file:           dirwatch.pl
#
#     purpose:        display size of subdirectories
#                     script ignores directories
#                     if the user doesn't have the
#                     rights to scan them, this is
#                     intentional
#
#     author:         Erik Keller
#
#     parameters:     -d <directory to scan>
#                     -k use KB instead of MB
```

```
#
##################################################

use Getopt::Std;

# Variables
$varpath="/export/home/users/erikk/work/baselines/var/dirwatch";

# get directory to watch
getopt('d');

if($opt_d eq '')
{
        usage();
}

open(MY_DIRS, " du -sk $opt_d\/\* |")
        or die "$0: du failed: $!";
while(<MY_DIRS>)
{
        chop;
        @myrec = split(" ", $_);
        $dirname = "dw_".$myrec[1];
        $dirname =~ s/\//_/g;
        if($opt_k)
        {
                # use KB
                $mysize = $myrec[0];
        }
        else
        {
                # use MB
                $mysize = int($myrec[0]/1024);
        }
        open(MY_LOG, ">>$varpath/$dirname")
                or die "$0: error using file: $!";
        print MY_LOG "$mysize\n";
        close(MY_LOG);
}

close(MY_DIRS);

sub usage()
{
        print "\nScans the subdirectories and reports the\n";
        print "amount of space used by them.\n";
        print "Directories the executing user has no rights\n";
        print "to scan are ignored.\n\n";
        print "Usage:\n";
        print "$0 -d <dirname>\n\n";
        exit(1);
}
# EOF
```

The feature to switch the units to KB seems to be outdated regarding the available hard disk space, but small changes to the size take a long time to show up if the size is measured in MB. The same goes for using the `-h` option in `df` and `du`: to get a general overview, using MB and GB is sufficient; tracking subtle changes requires a finer unit of measurement, thus allowing you a closer look at what is happening. Feel free to input the main directory containing the directories to be scanned directly in the code. To do so, you could simply change the parameter supplied to `du -sk` or create a variable to hold the path.

Because this script potentially creates a huge number of log files, storing its output to the general `var` directory in the baselining hierarchy is not a sound option; one should use a subdirectory instead. As already mentioned in the script's comments, directories that cannot be accessed in the current user-context are ignored. Nevertheless, this script should be accessible only to the superuser because it can be used to easily map directories, thus displaying confidential information. This script can also be used to do comparisons (size-wise), before and after applying updates and patches or installing software. You just have to remember to store these scans in another directory; the directory used to hold the usual scans could become very cluttered, thus making it less usable for day to day use.

3.1.6 Monitoring the CPU

The load on the CPU or CPUs on an SMP system is a clear indicator of how taxed the machine is at the moment—or is it? The answer to that question is not as clear cut as you might like it to be, but recording this value at least gives you some idea about the load situation on the system.

The easiest way to get access to this data is by using the *System Activity Reporter*, `sar`. This package is available on most *NIX systems, but is not necessarily installed by default. More often than not, it "hides" itself in the *accounting* package or the *sysstat* package; `sar` is usually available after installation of that package. The command to output the data is `sar`. It displays the CPU load by default, sliced by *user*, *system*, *idle*, and, with the exception of GNU-Linux, *wio (wait I/O)*. If you start `sar` on a regular basis via your `crontab` mechanism, an overview of the last 30 days is always at hand because the `crontab` is usually modified to do so during the installation of the package containing `sar`. If not, you could, as always, check the manpage to learn how to enable the continuous gathering of data and fine-tune the settings. Listing 3.4 shows a script for gathering this data into separate log files.

LISTING 3.4 Creating Log Files from `sar` Output

```
#!/usr/bin/perl
# $Id: base_cpu.pl,v 1.2 2003/06/03 10:32:54 erikk Exp $
##################################################
#
#     file:          base_cpu.pl
#
#     purpose:       save the current sar values
#                    to files
#
#     author:        Erik Keller
#
#     parameters:    none
#
##################################################

# Variables
$WORKDIR = "/export/home/users/erikk/work/baselines/var";

# create the 4 outfiles for gnuplot
open(SG_USRH, ">$WORKDIR/sar_usr.log")
        or die "$0: couldn't create file: $!";
open(SG_SYSH, ">$WORKDIR/sar_sys.log")
        or die "$0: couldn't create file: $!";
open(SG_WIOH, ">$WORKDIR/sar_wio.log")
        or die "$0: couldn't create file: $!";
open(SG_IDLH, ">$WORKDIR/sar_idl.log")
        or die "$0: couldn't create file: $!";

# get the stored values from sar for CPU
open(THE_SAR, "sar |") || die "$0: couldn't run sar: $!";

while(<THE_SAR>)
{
        if(/^[0-9]/)
        {
                @myrec = split(" ", $_);
                if($myrec[1] ne "\%usr")
                {
                        print SG_USRH "$myrec[1]\n";
                        print SG_SYSH "$myrec[2]\n";
                        print SG_WIOH "$myrec[3]\n";
                        print SG_IDLH "$myrec[4]\n";
                }
        }
}

close(THE_SAR);

close(SG_IDLH);
close(SG_WIOH);
close(SG_SYSH);
close(SG_USRH);

# EOF
```

This script writes the values of *user*, *system*, *wait I/O*, and *idle* into four log files. It creates a temporary snapshot of the current data. Further processing of the data takes place in Section 3.1.11, "Creating the Graphs."

Another interesting counter in that respect is the `runq` size, the queue of processes waiting to be executed in main memory. Listing 3.5 demonstrates how to gather data for this counter into a log file.

LISTING 3.5 Script to Store the `runq` Size

```
#!/usr/bin/perl
# $Id: base_runq.pl,v 1.1 2004/08/27 03:44:21 erikk Exp erikk $
##################################################
#
#     file:           base_runq.pl
#
#     purpose:        save the current size of the
#                     run-queue to a file
#
#     author:         Erik Keller
#
#     parameters:     none
#
##################################################

# Variables
```

```
$WORKDIR = "/export/home/users/erikk/work/baselines/var";

# open the file
open(MY_RUNQ, ">$WORKDIR/sar_runq.log")
        or die "$0: couldn't create file: $!";

# get the stored runq sizes
open(THE_SAR, "sar -q |") || die "$0: couldn't run sar: $!";

while(<THE_SAR>)
{
        if(/^[0-9]/)
        {
                @myrec = split(" ", $_);
                if($myrec[1] ne "runq-sz")
                {
                        print MY_RUNQ "$myrec[1]\n";
                }
        }
}
close(THE_SAR);

close(MY_RUNQ);

# EOF
```

3.1.7 Monitoring Network Connections

The number of simultaneous, active network connections is something you should keep an eye on. The script in Listing 3.6 counts all connections in status `ESTABLISHED` while it runs.

LISTING 3.6 Script to Count Active Network Connections

```
#!/usr/bin/perl
# $Id: base_netconn.pl,v 1.2 2004/08/23 12:42:08 erikk Exp erikk $
##################################################
#
#     file:             base_netconn.pl
#
#     purpose:          count number of network
#                       connections
#
#     author:           Erik Keller
```

```
#
#     parameters:      -4     counts inet4
#                      -6     counts inet6
#                      none   counts all
#
##################################################

use Getopt::Std;

# Variables
# path and files  !!!!!! change for every system
$varpath="/export/home/users/erikk/work/baselines/var";
$thelogfile="base_netconns.log";

# connection counter
$active_conns = 0;

# check for options on cmdline
# netstat options work on Solaris and OS/X
# Linux needs -A <family>, check man-page
getopt('');
if($opt_4)
{
        $netstatcmd = "netstat -f inet";
}
elsif($opt_6)
{
        $netstatcmd = "netstat -f inet6";
}
else
{
        $netstatcmd = "netstat";
}

# open the input stream from netstat
open(MY_NETSTAT, "$netstatcmd |") or die "$0: couldn't run: $!";
while(<MY_NETSTAT>)
{
        chop;
        if(/ESTABLISHED/)
        {
                $active_conns++;
        }
}
```

```
close(MY_NETSTAT);

# write to file
open(MY_LOG,">>$varpath/$thelogfile")
        or die "$0: couldn't open log file: $!";
print MY_LOG "$active_conns\n";

close(MY_LOG);

# EOF
```

The script can be used on Solaris and OS X—after modifying the paths—right away. Other *NIX flavors need to be tested to determine if the `netstat` programs knows the `-f <Address-Family>` parameter. If the script is called without parameters, it should run almost anywhere. The easiest way to use the script is via the `crontab`. How often the script needs to be called is heavily dependent on the services provided by the system to be checked. Every 30 minutes should be sufficient to get a general idea about the network load on the system. If a finer resolution is required, then run the script every 10 minutes. Depending on the requirements, your last option is to start the script continuously by hand in the frequency needed and to check the resulting log files.

3.1.8 Network Retransmissions

TCP/IP is, for historical reasons, extremely fault-tolerant; problems regarding the network infrastructure are therefore prone to occur unnoticed, at least for some time. The protocol is self-healing; this means it is able to keep the services up and running by avoiding defective components in the physical structure of the network. One of the results of this behavior is explained best by the following scenario: assume there is a network consisting of a couple of routers. If one of the routers should fail, then the protocol tries to find alternate routes for the data packages, and the failure will go unnoticed at first. The failure might become visible when the users of the routed services start to complain about slow connections or bad response times.

TIP

The author used to work for a company with branch offices situated all around the world.

One day the number of retransmissions suddenly started to grow exponentially. Minutes later the phone rang, and the internal users started to complain about slow response times. A quick run of `tcpdump` *displayed that packages supposed to go* *from London, England, to Los Angeles, California, were routed via Munich, Germany. The U.K. branch office used to have the faster connection to the United States; nevertheless, all Web and mail traffic was suddenly routed via the Munich branch office. The router in London seemed to route everything via another route instead of using its own, faster, connection. A check of the router revealed the reason for this behavior—one user in the U.K. obviously had started an update of a system there, using a machine in Los Angeles as source, and another user in the U.K. had transferred a substantial amount of data to a system in Los Angeles. All this took place inside the Frame Relay network of the company; the routers had simply decided that the available speed for the route via Munich was faster than the direct link and began to reroute all traffic over this "faster" connection—slowing down the company network in Germany.*

The current configuration of the routers should have prevented that. A phone call to the U.K. office revealed the source of the problem: someone had applied an update to the router's software and had forgotten to activate the routing rules again. That router had just created its own rules based on the speed and availability of the network connections. After the rules had been activated again, everything went back to normal. The time to react and rectify the configuration error was too short in this case to be noticed by the majority of the users, but the growth of retransmissions had been a clear indicator that something was wrong inside the network.

The ratio between the number of sent packages and retransmitted packages is a counter that should be watched all the time. Every *NIX system provides the counters used to calculate the ratio via the `netstat –s` command. Unfortunately, the output of `netstat` is not consistent on each platform. Listings 3.7, 3.8, and 3.9 show the output on Solaris, OS X, and GNU-Linux, respectively.

LISTING 3.7 Transmissions and Retransmissions on Solaris

```
...
        tcpOutDataSegs      =  7997     tcpOutDat...
        tcpRetransSegs      =     0     tcpRetran...
...
```

LISTING 3.8 Transmissions and Retransmissions on OS X

```
tcp:
        124691 packets sent
                91326 data packets (85129699 bytes)
                10 data packets (6483 bytes) retransmitted
...
```

LISTING 3.9 Transmissions and Retransmissions on GNU-Linux

```
...
    2249 segments send out
    6 segments retransmited
...
```

It is rather complicated to create a common script for all platforms to extract this vital data, because of the inconsistent output on the various platforms. If one should have to administer more than one platform, the usual case nowadays, two approaches present themselves.

- You could create a script containing only the base functionality and add the code for the respective operating system on a case-by-case basis. The advantage would be code that is easier to read and understand; the downside of this approach is the fact that there are different versions of the script—to be maintained separately.
- The script tries to guess the operating system it is deployed on and calls the specific code. The advantage in this case would be having only one script to maintain; the downside is scripts that try to do too many things at a time tend to become unreadable and thus harder to maintain over time.

However, there is also a "secret" option three—creating an object-oriented script that uses an abstraction to process the gathered data. That option is not only beyond the scope of this book, but also the design of such a script should always be tested via simple, streamlined scripts that do one thing right. You should always keep in mind that the KISS principle is the only way to go regarding system administration. Also, if you try to achieve results that could make the monitoring systems of the common software distributors obsolete, then the system you are responsible for is not the platform for the necessary experiments to achieve that goal.

Listing 3.10 shows a script for Solaris that takes a slight glitch into account.

LISTING 3.10 Recording the Retransmission Ratio (Solaris)

```
#!/usr/bin/perl
# $Id: base_retrans.pl,v 1.1 2004/08/24 15:22:03 erikk Exp erikk $
##################################################
#
#     file:             base_retrans.pl
#
#     purpose:          calculate the ratio between
#                      transmitted and retransmitted
#                      segments and write it to a file
#                      SOLARIS
#
#     author:          Erik Keller
#
#     parameters:      none
#
#
##################################################

# Variables
$varpath = "/export/home/users/erikk/work/baselines/var";
$thelogfile = "base_retrans.log";

# open netstat -s Solaris version
open(MY_NETSTAT, "netstat -s | head -19 |")
        or die "$0: problems running netstat: $!";
while(<MY_NETSTAT>)
{
        chop;
        if(/tcpOutDataSegs/)
        {
                @tcpout = split(" ", $_);
                if(length($tcpout[1]) >= 2)
                {
                        $tcpout[1] =~ s/=//;
                        $tcpoutsegs = $tcpout[1];
                }
                else
                {
                        $tcpoutsegs = $tcpout[2];
                }
        }
        if(/tcpRetransSegs/)
        {
                @tcpret = split(" ", $_);
                if(length($tcpret[1]) >= 2)
                {
                        $tcpret[1] =~ s/=//;
                        $tcpretsegs = $tcpret[1];
                }
                else
                {
                        $tcpretsegs = $tcpret[2];
                }
        }
}

close(MY_NETSTAT);

# calculate ratio
$retransratio = (($tcpretsegs * 100)/$tcpoutsegs);

# write to file
open(MY_LOG, ">>$varpath/$thelogfile")
        or die "$0: couln't open log file: $!";
print MY_LOG "$retransratio\n";

close(MY_LOG);

# EOF
```

If you take a closer look at the `netstat` call, you will notice that the output will be truncated after 19 lines. It is even possible to extract only the lines of interest for the task at hand, but the more complicated the call gets, the harder it is to debug if something does not seem to work the way you intended it to work. The mentioned display glitch on Solaris results in the following—if the value increases over a certain number of digits, the space between the equal sign and the value disappears, thus resulting in a value that looks like this: `=4803517`. To ensure that the value in `$tcpout[1]` is valid, the script counts the number of digits stored in that variable; if the number of digits is `>2`, then the space character is missing and the value you are interested in is in the second field of the array. The instruction `$tcpout[1] =~ s/=//;` replaces (`s` = substitute) the `=` character with the value contained between the second pair of slashes, thus removing it from the string, leaving only the numbers. The `=~` instructs Perl to modify the object on the left-hand side. The display format used by this variant of `netstat` is two columns, or seen from a programmer's point of view six columns. The third field of the array would contain a real string and not a number, in case the mentioned overflow would occur. Because Perl always tries to execute the calculation by converting the datatype, the calculation involving a string would always return zero. This behavior can be tested easily.

```
erikk@unixhost> perl -e '$mytest = (("blahblahblahsdk"*100)/1024);\
print "$mytest\n"'
0
```

Nevertheless, you should always check the results of the scripts you create very thoroughly by using the values that the script would gather while running. The Solaris display glitch was found during a test involving values gathered from a file server. The workstation used for development interestingly never showed the overflow because of reduced network traffic. If all necessary tests are completed, then the script is ready for deployment. You should verify the results of a newly deployed script for at least another week.

Listing 3.11 shows the same script with the necessary code to obtain the values on GNU-Linux.

LISTING 3.11 Recording the Retransmission Ratio (GNU-Linux)

```
#!/usr/bin/perl
# $Id: base_retrans.pl,v 1.1 2004/08/24 15:22:03 erikk Exp erikk $
##################################################
#
#     file:             base_retrans.pl
#
#     purpose:          calculate the ratio between
#                       transmitted and retransmitted
#                       segments and write it to a file
#                       LINUX
#
#     author:           Erik Keller
#
#     parameters:       none
#
#
##################################################

# Variables
$varpath = "/home/erikk/work/scripts/baselines/var";
$thelogfile = "base_retrans.log";

# get output of netstat -s and find retransmissions
# linux specific, code needs rework for other *NIX
open(MY_NETSTAT, "netstat -s -t | head -9 |")
        or die "$0: netstat problem: $!";

while(<MY_NETSTAT>)
{
        chop;
        if(/segments send out/)
        {
                @myrec = split(" ", $_);
                $tcpoutsegs = $myrec[0];
        }
        if(/segments retransmited/)
        {
                @myrec = split(" ", $_);
                $tcpretsegs = $myrec[0];
        }

}

close(MY_NETSTAT);

# calculate ratio
$retransratio = (($tcpretsegs * 100)/$tcpoutsegs);

# write to file
open(MY_LOG, ">>$varpath/$thelogfile")
        or die "$0: couln't open log file: $!";
print MY_LOG "$retransratio\n";

close(MY_LOG);

# EOF
```

The GNU-Linux variant used (SuSE 8.2) allows you to filter the output of `netstat` for TCP statistics via the `–t` option. The resulting lines are searched for the strings `segments send out` and `segments retransmited` (that's not a misspelling) to obtain the values.

With these two scripts as a base, you should be able to customize as necessary and obtain the values needed for the calculation of the retransmission ratio on almost every machine.

3.1.9 What Does `TIME_WAIT` Mean?

A TCP network connection in the `TIME_WAIT` state signals that the connection was terminated and that the system will wait for a predefined span of time for incoming packages from the other system or, respectively, that it will wait for the connection partner to confirm the termination of the connection. The TCP protocol was designed this way to minimize the loss of packages.

So far so good—if the requested confirmation for the termination of the connection is not received within the predefined time span, the system will drop the connection anyway. Usually, this design choice doesn't pose a problem at all, but systems with a huge number of open connections, for example, a Web server, could tie up precious resources this way. Even worse, some clients using a TCP/IP stack of questionable quality might let the number of connections in `TIME_WAIT` state grow exponentially (even if this statement covers the better part of Windows, there are other systems showing the same behavior, too). Changes to the time-out value of `TIME_WAIT` should be executed with extreme care, though.

The script in Listing 3.12 enables you to obtain the number of connections in `TIME_WAIT` state (HP-UX sometimes uses a `TIME_WAIT2` state as well).

LISTING 3.12 Recording the Number of Connections in `TIME_WAIT`

```
#!/usr/bin/perl
# $Id: base_timewait.pl,v 1.1 2004/08/24 09:02:03 erikk Exp erikk $
##################################################
#
#     file:            base_timewait.pl
#
#     purpose:         number of network connections
#                      in TIME_WAIT state
#
#     author:          Erik Keller
#
#     parameters:      -4     counts inet4
#                      -6     counts inet6
#                      none   counts all
#
##################################################

use Getopt::Std;

# Variables
# path and files  !!!!!! change for every system
$varpath="/export/home/users/erikk/work/baselines/var";
$thelogfile="base_timewait.log";

# connection counter
$timewait_conns = 0;

# check for options on cmdline
# netstat options work on Solaris and OS/X
# Linux needs -A <family>, check man-page
getopt('');
if($opt_4)
{
        $netstatcmd = "netstat -f inet";
}
elsif($opt_6)
{
        $netstatcmd = "netstat -f inet6";
}
else
{
        $netstatcmd = "netstat";
}

# open the input stream from netstat
open(MY_NETSTAT, "$netstatcmd |") or die "$0: couldn't run: $!";
while(<MY_NETSTAT>)
{
        chop;
        if(/TIME_WAIT/)
        {
                $timewait_conns++;
        }
}

close(MY_NETSTAT);

# write to file
open(MY_LOG,">>$varpath/$thelogfile")
        or die "$0: couldn't open log file: $!";
print MY_LOG "$timewait_conns\n";

close(MY_LOG);

# EOF
```

The script resembles, more or less, the one used to count the active network connections. As already mentioned, if this script is needed for the purpose of the system to be monitored, it depends heavily on the intended use of that system.

3.1.10 Monitoring SYN Packages

The `SYN_SENT` and `SYN_RECEIVED` states of network connections translate into either the system trying to establish a connection (`SYN_SENT`) or another system trying to establish a connection to it (`SYN_RECEIVED`). This behavior is quite normal, but because not every system connected to the Internet is to be trusted, you should be aware of so-called `SYN` attacks. These attacks are used, among other things, to find out about the open ports on your systems or to render the services on your systems useless. This is called a *DOS* attack (pun intended), spelled out denial-of-service.

If an IDS system is active inside the network (see Chapter 7 for more on Intrusion Detection Systems), then such attempts should not go unnoticed. But beyond being a pointer that an attack is going on, the growth of connections in SYN state could also mean you have a malfunctioning device connected to your network, slowing down the established connections. The proper use of baselining statistics enables you to visualize these changes (the same holds true for connections in TIME_WAIT state) and react accordingly. So monitoring these counters is not only about defending yourself, but also about being notified of changes regarding the behavior of your networks, thus allowing you to make assumptions about the state of those networks.

Listing 3.13 demonstrates a script used to record the number of connections in the mentioned SYN states.

LISTING 3.13 Recording the Number of Connections Regarding the SYN State

```
#!/usr/bin/perl
# $Id: base_syn.pl,v 1.1 2004/08/24 10:08:50 erikk Exp erikk $
##################################################
#
#     file:             base_syn.pl
#
#     purpose:          count number of network
#                       connections in SYN state
#
#     author:           Erik Keller
#
#     parameters:       -4     counts inet4
#                       -6     counts inet6
#                       -i     incomming connections
#                       -o     outgoing connections
#                       none   counts all
#
##################################################

use Getopt::Std;

# Variables
# path and files  !!!!!! change for every system
$varpath="/export/home/users/erikk/work/baselines/var";
$thelogfile="base_syn.log";
```

```
# connection counter
$syn_conns = 0;

# check for options on cmdline
# netstat options work on Solaris and OS/X
# Linux needs -A <family>, check man-page
getopt('');
if($opt_4)
{
        $netstatcmd = "netstat -f inet";
}
elsif($opt_6)
{
        $netstatcmd = "netstat -f inet6";
}
else
{
        $netstatcmd = "netstat";
}
# check if all, incoming or outgoing
if($opt_i)
{
        # incoming
        $nstate = "SYN_RECEIVED";
}
elsif($opt_o)
{
        # outgoing
        $nstate = "SYN_SENT";
}
else
{
        $nstate = "SYN";
}

# open the input stream from netstat
open(MY_NETSTAT, "$netstatcmd |") or die "$0: couldn't run: $!";
while(<MY_NETSTAT>)
{
        chop;
        if(/$nstate/)
        {
                $syn_conns++;
        }
}

close(MY_NETSTAT);

# write to file
open(MY_LOG,">>$varpath/$thelogfile")
        or die "$0: couldn't open log file: $!";
print MY_LOG "$syn_conns\n";

close(MY_LOG);

# EOF
```

The script is able to record only the number of either incoming (option `-i`) or outgoing (option `-o`) connections if needed. There should not be a value greater than zero during normal operations.

3.1.11 Creating the Graphs

The numbers recorded in the log files are of limited use. Great deviations might become visible, but only if the deviation is big enough; it is almost impossible to spot evolving trends. To visualize data gathered by `sar`, you could use the program `sag`, but this program is far from user friendly; sometimes it requires being run inside an `x-term` using a special Tektronix-emulation mode. The other issue here is the fact that it cannot be used to visualize the data recorded by the other scripts.

Taking this into account, two options remain: either you use a spreadsheet program (like OpenOffice Calc or MS Excel, for example) capable of transforming your columns of numbers into meaningful graphics, or you use the program `gnuplot`. The drawback of `gnuplot`—and the same goes for OpenOffice Calc—is that, with the exception of GNU-Linux and Solaris, the program needs to be obtained and installed first (to obtain `gnuplot` for other *NIX brands, you can use the Internet addresses in Appendix C of this book). The program is usually available as binary, ready to be installed and used; there is no need to compile it first.

There is no real need to explain the use of a spreadsheet program, but a couple of remarks regarding `gnuplot` are appropriate. The program is free software; it runs on almost all operating systems available and enables you to plot functions and—more interesting for the task at hand—data sets. The name has no connection whatsoever to GNU; in fact, pronounced correctly the name should sound like "newplot." If you are interested in the history of the program, then *http://www.gnuplot.info/faq/faq.html* is the place to learn more about how the program came to life.

After the program is installed on a system, it can be started by executing `gnuplot` on the command line. Listing 3.14 shows the result.

LISTING 3.14 Starting `gnuplot`

```
erikk@unixhost> gnuplot

        G N U P L O T
        Version 4.0 patchlevel 0
        last modified Thu Apr 15 14:44:22 CEST 2004
        System: SunOS 5.9

        Copyright (C) 1986 - 1993, 1998, 2004
        Thomas Williams, Colin Kelley and many others

        This is gnuplot version 4.0. Please refer to the documentation
        for command syntax changes. The old syntax will be accepted
        throughout the 4.0 series, but all save files use the new syntax.

        Type `help` to access the on-line reference manual.
        The gnuplot FAQ is available from
                http://www.gnuplot.info/faq/

        Send comments and requests for help to
                <gnuplot-info@lists.sourceforge.net>
        Send bugs, suggestions and mods to
                <gnuplot-bugs@lists.sourceforge.net>


Terminal type set to 'x11'
gnuplot> plot sin(x)
```

If you issue the command `plot sin(x)` at the `gnuplot>` prompt, another window opens and should contain the plot shown in Figure 3.1.

The simple sinus plot is not that interesting, but it is a confirmation that the program works. If you change into the directory containing the log files created by your scripts (or prepend the filename in the following command with the complete path to them), things will get a lot more interesting. The command used would be `plot "base_netconns.log" with linespoints`; after hitting Enter, you are presented with a plot similar to the one shown in Figure 3.2.

As brilliant as this display of the gathered data might be, typing the command for every log file gets cumbersome soon. Therefore, `gnuplot` enables you to use so-called *command files*. You are able to type the necessary command into a plain text file and ask `gnuplot` to execute the chain of commands for us. This mechanism even enables you to generate the graphs via `crontab` as soon as fresh data was recorded. Listing 3.15 shows the command file used to generate the graph in Figure 3.2.

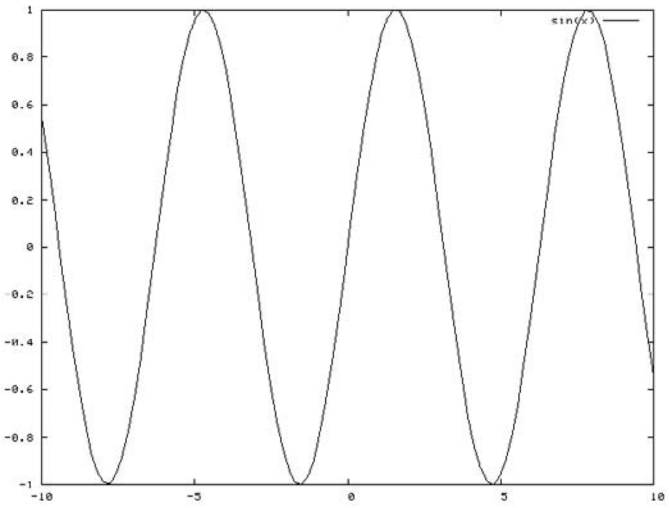


FIGURE 3.1 Starting `gnuplot`.

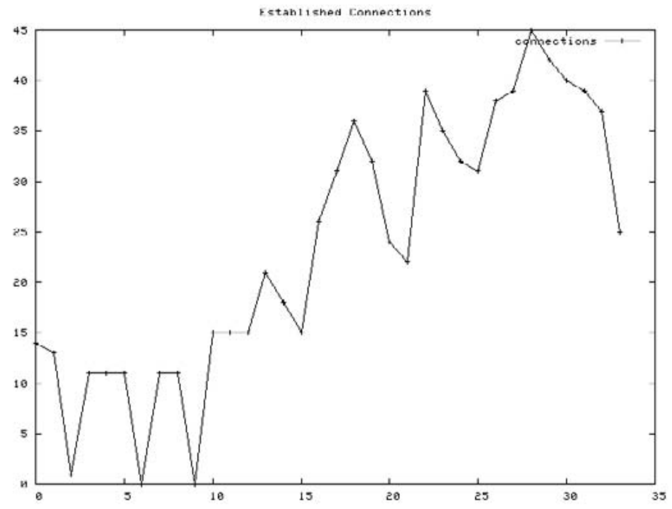


FIGURE 3.2 Plotting the data.

LISTING 3.15 The Command File for `base_netconns.png`

```
set title "Established Connections"
set terminal png monochrome
set output "../html/images/base_netconns.png"
plot "../var/base_netconns.log" with linespoints title "connections"
```

You are able to type the commands at the prompt of gnuplot one by one; this enables you to test command files and get a general feeling how the program reacts to certain commands. If the parameter monochrome is omitted, gnuplot will create a colored PNG file. Because gnuplot contains a large assortment of commands and options, you should use the built-in help function to learn more about the program. Listing 3.16 shows an example.

LISTING 3.16 The help Function in gnuplot

```
gnuplot> help set terminal
 `gnuplot` supports many different graphics devices.  Use `set termi-
nal` to
 tell `gnuplot` what kind of output to generate. Use `set output` to
redirect
 that output to a file or device.

 Syntax:
       set terminal {<terminal-type> | push | pop}
       show terminal

 If <terminal-type> is omitted, `gnuplot` will list the available ter-
minal
 types.  <terminal-type> may be abbreviated.

 If both `set terminal` and `set output` are used together, it is
safest to
 give `set terminal` first, because some terminals set a flag which is
needed
 in some operating systems.
...
```

The help function is organized in a tree structure. If you use the command help set, you will get a list of all options set understands. There is a list of all supported terminals in the example shown after about a screen full of text; the term terminal is somewhat misleading in that respect, though. A graphic format like PNG, JPG, or PDF is also considered to be of terminal type. Listing 3.17 shows the command file to output the recorded CPU counters extracted from sar via script.

LISTING 3.17 A Command File to Output Datasets

```
set terminal png monochrome
set output
"/export/home/users/erikk/work/baselines/html/images/base_cpu.png"
```

```
set grid\
set yrange [0:100]
set title "CPU Counters"
plot "/export/home/users/erikk/work/baselines/var/sar_usr.log"\
with linespoints title "User",\
"/export/home/users/erikk/work/baselines/var/sar_sys.log"\
with linespoints title "System",\
"/export/home/users/erikk/work/baselines/var/sar_wio.log"\
with linespoints title "Wait I/O",\
"/export/home/users/erikk/work/baselines/var/sar_idl.log"\
with linespoints title "Idle"
```

If you want to display more than one set of data in a graph, you just have to separate the specific datasets with a comma. If the lines exceed a certain length, thus becoming unreadable, then you can break these lines with the continuation character `\`, thus rendering them readable again. Using the continuation character works the same way as it does in scripts; there cannot be another character after it.

As shown in Listing 3.17, command files can be quite complex and typing long pathnames is error prone at best. To ease this situation somewhat, you could use Perl to generate the command files for us. An added bonus of this approach is the fact that you can react to certain conditions in the log files more easily. Listing 3.18 shows an example that keeps the y-axis at a certain level unless this threshold is exceeded.

LISTING 3.18 Generating a Command File with Perl

```
#!/usr/bin/perl
# $Id: graph_base_netconns.pl,v 1.1 2004/08/25 15:34:21 erikk Exp erikk
$
##################################################
#
#     file:            graph_base_netconn.pl
#
#     purpose:         create command-file to plot
#                      the data in the file
#                      base_netconns.log using
#                      gnuplot
#
#     author:          Erik Keller
#
#     parameters:      none
#
##################################################
```

```
# Variables

# paths and files
$bbasedir = "/export/home/users/erikk/work/baselines";
$datafile ="$bbasedir/var/base_netconns.log";
$imagepath = "$bbasedir/html/images";
$imagename = "base_netconns.png";
$cmdfilepath = "$bbasedir/plotcmds";
$cmdfilename = "base_netconns.plot";

# gnuplot vars
$title = "Established Connections";
$terminal = "png monochrome";
$outfile = "$imagepath/$imagename";
$yrange = 100;
$lineform = "linespoints";
$linetitle = "connections";

# test if the number of connections is greater than yrange
open(MY_LOG, "$datafile")
        or die "$0: problems opening data file: $!";
while(<MY_LOG>)
{
        chop;
        if($_ >= $yrange)
        {
                $yrange = $_+10;
                # increase the y-axis
        }
}

close(MY_LOG);

# write the command file
open(MY_CMDFILE, ">$cmdfilepath/$cmdfilename")
        or die "$0: Problem opening file: $!";
print MY_CMDFILE "set terminal $terminal\n";
print MY_CMDFILE "set output \"$outfile\"\n";
print MY_CMDFILE "set grid\n";
print MY_CMDFILE "set yrange [0:$yrange]\n";
print MY_CMDFILE "set title \"$title\"\n";
print MY_CMDFILE "plot \"$datafile\" with $lineform ";
print MY_CMDFILE "title \"$linetitle\"\n";

close(MY_CMDFILE);

# EOF
```

The code shown in **bold** is an example of how to deal with changes inside the datasets. Assume the value of 100 is sufficient regarding the y-axis; if there are more than 100 connections, then the script scales the y-axis to the highest value `+ 10` and stores this value in `max`. As long as there are no values > 99, `max` stays at 100.

Figure 3.3 shows a graph with a scaled y-axis and Figure 3.4 with the default y-axis.

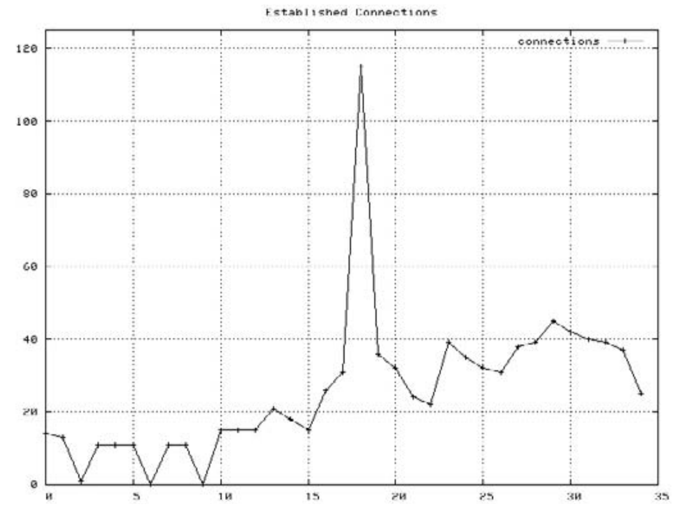


FIGURE 3.3 Graph with a scaled axis.

At the moment, the script automates only the creation of the command file. If you want to create the graphic at the same time, you have to make a choice between two approaches. Either you add two lines to the script, `$gnuplot. = "/usr/local/bin/gnuplot";` in the `#Variables` section and `exec("$gnuplot $cmdfilepath/$cmdfilename");`, or you create another script that takes care of the creation of the graphs. Listing 3.19 shows a script doing just that.

LISTING 3.19 Script to Create the Graphs

```
#!/usr/bin/sh
```

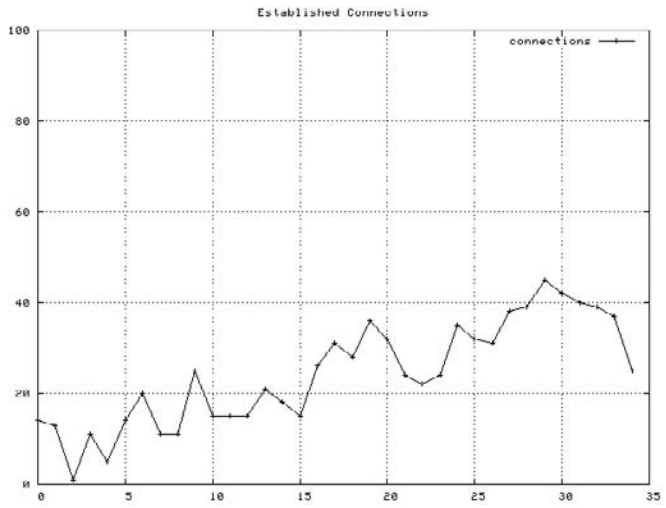


FIGURE 3.4 Graph with the default axis.

```
# $Id: graphcreate.sh,v 1.1 2004/08/26 10:36:57 erikk Exp erikk $
##################################################
#
#     file:          graphcreate.sh
#
#     purpose:       tests if there are files with
#                    the extension .plot and feeds
#                    them to gnuplot
#
#     author:        Erik Keller
#
#     parameters:    none
#
##################################################

# Variables
MY_DIR=/export/home/users/erikk/work/baselines/plotcmds
GNUPLOT=/usr/local/bin/gnuplot

for THE_PLOTC in $MY_DIR/*.plot; do
        if [ -f $THE_PLOTC ]
        then
                $GNUPLOT $THE_PLOTC
        fi
done
```

```
logger -p user.crit executed plot commands in $MY_DIR

# EOF
```

This script searches the given directory for files ending in `.plot` and hands them over to `gnuplot`. The advantage of this approach is if you should create new command files, then these new files just have to be copied into the specific directory and the script will process them automatically during the next run.

3.1.12 Using a Browser to Display the Graphs

Using any program to display the created graphs is possible, but getting a view of the bigger picture, in other words *what is happening on the system*, is still hard. Using a Web browser to view the generated graphs has two advantages. First, you are able to view the graphs on any device capable of displaying them, and second, provided the files are published on a Web server, you are able to publish them either internally or even externally for consumption. This means you can monitor the systems while away from the office.

Listing 3.20 shows a simple HTML page to display the generated data and Listing 3.21 the accompanying *stylesheet*.

LISTING 3.20 HTML Code to Display the Graphs

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Baselines for unixhost</title>
<link rel="stylesheet" title="baseline_style (default)"
        href="baselining.css" type="text/css" media="screen">
</head>

<body>
<h1>Baselines for unixhost</h1>
<h2>CPU</h2>
<img src="images/base_cpu.png">
<img src="images/base_runq.png">

<h2>Disks</h3>
<h3>/ (root)</h3>
<img src="images/df__.png">

<h3>/var</h3>
<img src="images/df__var.png">
```

```
<h3>/export/home</h3>
<img src="images/df__export_home.png">

<h2>Network Counters</h2>
<h3>Established Connections</h3>
<img src="images/base_netconns.png">

<h3>Retransmission Ratio</h3>
<img src="images/base_retrans.png">

<h3>TIME_WAIT Connections</h3>
<img src="images/base_timewait.png">

<h3>SYN Connections</h3>
<img src="images/base_syn.png">


</body>
</html>
```

LISTING 3.21 The Stylesheet

```
/* $Id$ */
/* stylesheet for baseline display */
/* screen output */

body {
        background-color: white;
        font-size: 12px;
        font-family: sans-serif;
}

h1 {
        font-size: 140%;
}

h2 {
        font-size: 130%;
        color: white;
        background-color: black;
}

h3 {
        font-size: 120%;
}

/* EOF */
```

The HTML code and the stylesheet are simple and, in case of the HTML code, not the latest standard in use for a reason. They will display correctly even on a browser like Microsoft Internet Explorer. That program is still not capable of understanding the standards of W3C (the consortium that recommends the standards every other Web browser adheres to) regarding the positioning of objects.

3.1.13 Managing Log Files

The growth of the log files created by the shown scripts is not limited at the moment, so you have to take care that they do not fill up all available disk space. The directory `baselines/housekeeping` contains the script in Listing 3.22, which is run shortly before midnight to take care of this issue.

LISTING 3.22 Script to Rotate the Log Files

```
#!/usr/bin/sh
# $Id: logrotate.sh,v 1.1 2004/08/26 09:17:04 erikk Exp erikk $
################################################
#
#     file:           logrotate.sh
#
#     purpose:        rotates the generated logfiles
#                     for baselining
#
#     author:         Erik Keller
#
#     parameters:     none
################################################

# Variables
LOGPATH=/export/home/users/erikk/work/baselines/var

if test -d $LOGPATH
then
        cd $LOGPATH
        for LOG in *.log; do
                test -f $LOG.2 && mv $LOG.2 $LOG.3
                test -f $LOG.1 && mv $LOG.1 $LOG.2
                test -f $LOG.0 && mv $LOG.0 $LOG.1
                test -f $LOG && mv $LOG $LOG.0
        done
fi

logger -p user.crit baseline logs rotated

# EOF
```

The script searches the given directory for files with the extension `.log`. If one is found, the script searches for a file with the same name and the extension `.2`. If the latter is found, it will be copied to `.3`. After that, the script searches for a file with the found name and the extension `.1` and so forth. As soon as all older files are copied over, the current file is moved to the extension `.0`. Because the scripts will create the log files if they do not exist, all log files are created again after midnight as soon as the scripts are run. This approach leaves you with the log files that cover the last four days. If you should need a longer history, then either you simply need to extend the `for` loop, or you could use the script in Listing 3.23.

LISTING 3.23 Saving the Log Files with a Date

```
#!/usr/bin/sh
# $Id: logrotatedate.sh,v 1.1 2004/08/26 10:26:16 erikk Exp erikk $
################################################
#
#     file:          logrotatedate.sh
#
#     purpose:       rotates the generated logfiles
#                    for baselining and saves the
#                    copies with a date tag
#
#     author:        Erik Keller
#
#     parameters:    none
################################################

# Variables
LOGPATH=/export/home/users/erikk/work/baselines/var
DATE=`date +%y%m%d`
if test -d $LOGPATH
then
        cd $LOGPATH
        for LOG in *.log; do
                test -f $LOG && mv $LOG $LOG.$DATE
        done
fi
```

```
logger -p user.crit baseline logs rotated

# EOF
```

The log files will now be rotated by renaming them to `base_netconns.log.050822`, for example. The only drawback to this approach is the fact that you need to make sure that there is either enough room for the next generation of log files or move them to another location to free up some space.

3.2 USERS AND GROUPS

Regardless of the purpose of a system or even a network, the computers need to know who is allowed to do what, where, and sometimes when. An operating system differentiates between the people allowed to perform certain actions on it by the so-called User IDentification Number or UID for short. To ease the administration of a number of users with related rights, the concept of bundling them together as a group is used. To ease the interaction of the system with the computer's administrator and users, both have names assigned to them. The following sections explain the concepts and related tasks to administer both users and groups.

3.2.1 What Is a User?

A user is to a *NIX system what an IP address is to a computer, just a number with some kind of meaning. To make sense for the user, the IP address can be mapped to a name like *slashdot.org*, which is easier to memorize than 66.35.250.150. Similarly, the user number is what the computer actually uses and the username is what the human in front of the machine uses to identify certain users, representing the same kind of number-to-name mapping.

3.2.2 What Is the `passwd` File?

To create the mapping of the user number to the user name, you store a file called `passwd` in the `/etc` directory. All programs are allowed to read this file, but only `root` is allowed to write to it. The structure of the file comes from the very first incarnations of *NIX systems. It used to contain the scrambled passwords as well. This arrangement proved to be too volatile after a while. Programs to "crack" the stored passwords in the `passwd` file started circulating in the network that would become the Internet one day, and to increase security, the scrambled passwords were stored in another file called `shadow` (see Section 3.2.3, "What Is the `shadow` File?", for more information). The structure of a user record in the `/etc/passwd` file is shown in Listing 3.24.

LISTING 3.24 A User Record in the `passwd` File

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

The delimiter is the colon ":", the first field is the username, the second field is where the scrambled password used to be, the third field is the user number, the fourth field is the number of the primary group, the fifth field contains the "real name" of the user, the sixth field the home directory of the user, and the seventh field the default shell to be used for interactions with the system. If the user record describes not a real user but the user-context of a process, for example, `lp`, responsible for managing the print queues, then the field that stores the standard shell contains `/usr/bin/false`, `/bin/false`, or simply nothing.

The term *user-context* refers to processes that are running in the context of a certain user number in the system, in the user space to be precise. A user has certain usage rights granted by the administrator regarding reading and writing from and to files and execution rights for certain programs. On a well-administered system, a user is never able to exceed the given rights, thus making the system secure.

One question remains—why is there still a field where the scrambled password used to be stored? This is a question of continuity; removing the field would require the possible rewrite of almost every program inside the *NIX system. At the very least every program would have to be checked to see if it still made use of the file. The two "wasted" bytes in every user record are a small price to pay compared to the effort needed to test every part and every program in a whole system.

3.2.3 What Is the `shadow` File?

As already mentioned, the file named `shadow` in `/etc` stores the scrambled passwords used by the system. Only `root` has read permission on this file; no one else. Listing 3.25 shows the permissions on `passwd` and `shadow`.

LISTING 3.25 Access Permissions of `passwd` and `shadow`

```
root@unixhost> ls -l /etc/passwd
-r--r--r--   1 root     sys          568 Jun 29 10:40 /etc/passwd
root@unixhost> ls -l /etc/shadow
-r--------   1 root     sys          332 Aug  2 15:38 /etc/shadow
```

The structure of a user record in `shadow` is shown in Listing 3.26.

LISTING 3.26 A User Record in the shadow File

```
root:UkE56lKUsPeSQ:6445::::::
```

The structure is as follows: the first field contains the username, the second field the ciphered password, the third field the number of days since the password was last changed, the fourth field the number of days the password cannot be changed, the fifth field the number of days the password will be valid, the sixth field the number of days before the warning about a password becoming invalid is issued, the seventh field the number of days within which the account has to be used (it will be locked otherwise), the eighth field the expiration of the account as an absolute date, and the ninth field is reserved for further extensions.

HP-UX introduced the `shadow` concept in Release 11.11i v1.6; this means in the Intel version. In your specific case, you should check your documentation to make sure all services are able to make use of the `shadow` file. Older HP-UX releases use the file `/tcb/files/auth/r/root` for the mentioned concept, and AIX stores the ciphered password in the file `/etc/security/passwd`.

3.2.4 What Is a Skeleton File?

The so-called *skeleton* directory contains files and directories that are used for all user accounts. Many *NIX systems use the directory `/etc/skel` for that purpose; it contains files like `.profile`, `.login`, and `.cshrc`, for example. These files will be copied in the `$HOME` directory of every new user, provided the program to create a new user makes use of the skeleton directory. Every system administrator should make use of this directory, store all files and directories in it that should be part of every home directory, and if necessary, create the missing files in accordance with the IT policy of the company. This way, all new users will find a consistent environment when logged into their respective accounts.

As already mentioned, as system administrator, you are free to create new files and directories in `/etc/skel`. If the program used to create new users on the system supports an alternate path to be used as a skeleton directory, it is even possible for you to create different directories for different types of users. Listing 3.27 shows an example for a "normal" user account.

LISTING 3.27 A Skeleton Directory for Users

```
root@unixhost> ls -alR std_user
std_user:
total 8
drwxr-xr-x    5 root     sys            512 Aug 19 11:08 .
drwxr-xr-x    4 root     sys            512 Aug 19 11:08 ..
```

```
-rw-r--r--    1 root     sys           136 Aug 19 11:07 .cshrc
-rw-r--r--    1 root     sys           157 Aug 19 11:07 .login
-rw-r--r--    1 root     sys           174 Aug 19 11:07 .profile
drwxr-xr-x    2 root     sys           512 Aug 19 11:07 documents
drwxr-xr-x    2 root     sys           512 Aug 19 11:08 to_be_backed_up
drwxr-xr-x    2 root     sys           512 Aug 19 11:07 work

std_user/documents:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:07 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:08 ..

std_user/to_be_backed_up:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:08 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:08 ..

std_user/work:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:07 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:08 ..
```

The structure in Listing 3.27 contains all default configuration files for the various shells and three directories that should be available in every home directory, according to the IT policy. This structure is supposed to be an example that can be customized according to the needs of the company, and doing so eases the creation of a bigger number of new users considerably.

Listing 3.28 shows an example for development accounts.

LISTING 3.28 A Skeleton Directory for Developers

```
root@unixhost> ls -alR developer
developer:
total 8
drwxr-xr-x    5 root     sys           512 Aug 19 11:14 .
drwxr-xr-x    4 root     sys           512 Aug 19 11:08 ..
-rw-r--r--    1 root     sys           136 Aug 19 11:09 .cshrc
-rw-r--r--    1 root     sys             0 Aug 19 11:14 .emacs
-rw-r--r--    1 root     sys             0 Aug 19 11:14 .exrc
-rw-r--r--    1 root     sys           157 Aug 19 11:09 .login
-rw-r--r--    1 root     sys           174 Aug 19 11:09 .profile
drwxr-xr-x    3 root     sys           512 Aug 19 11:13 documents
drwxr-xr-x    2 root     sys           512 Aug 19 11:09 projects
```

```
drwxr-xr-x    2 root     sys           512 Aug 19 11:09 to_be_backed_up

developer/documents:
total 3
drwxr-xr-x    3 root     sys           512 Aug 19 11:13 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:14 ..
drwxr-xr-x    2 root     sys           512 Aug 19 11:28 coding_guide-
lines

developer/documents/coding_guidelines:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:28 .
drwxr-xr-x    3 root     sys           512 Aug 19 11:13 ..
-rw-r--r--    1 root     other     2354755 Aug 19 11:28 C++.pdf
-rw-r--r--    1 root     other     3748271 Aug 19 11:28 Java.pdf
-rw-r--r--    1 root     other     2895422 Aug 19 11:28 Perl.pdf

developer/projects:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:09 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:14 ..

developer/to_be_backed_up:
total 2
drwxr-xr-x    2 root     sys           512 Aug 19 11:09 .
drwxr-xr-x    5 root     sys           512 Aug 19 11:14 ..
```

The structure in Listing 3.28 could be used for developers with no permanent connection to the company network, for example.

3.2.5 What Is a Group?

A group in *NIX fulfills the same purpose as in the real world. It contains people or user accounts in *NIX sharing a common goal or purpose. If a user needs to access a certain file, but is not the owner of that file, then it is sufficient for that user to be a member of a group that has the necessary access right to the file. Every user is a member of at least one group. Which group a user is a member of is a decision made by the administrator during the creation of the account. This group is the so-called primary group. Membership in additional groups is optional and heavily dependent on the activities a user should perform while using the account, for example, being a member of the group with execution rights for the database maintenance software.

3.2.6 What Is the `group` File for?

Listing 3.29 shows the structure of a group record in `/etc/group`.

LISTING 3.29 Structure of a Group Record

```
staff::10:erikk,silvied,ralphm
```

The first field contains the name of the group, the second field could store a ciphered password (normally unused), the third field contains the number of the group, and the fourth field the members of the group, separated by a comma.

HP-UX uses another file to manage groups, `/etc/logingroup`; AIX uses the additional file `/etc/security/group`. Administrators on those systems should not edit the group file with an editor, but use the programs `smit` (AIX) or `SAM` (HP-UX), respectively.

3.2.7 How to Create a User

Most of the time, the system administrator has the choice of how a user is created. Usually you have a GUI (Graphical User Interface) to maintain the user database or a program or script you can use via the command line. The drawback of the former is usually the fact that the GUI works only with one user record at a time, whereas the command-line program normally supports the use of a list of names to create the accounts accordingly. Another option would be the editing of `passwd` with an editor like `vi`, but this approach is a risky business. Some *NIX variants allow this type of editing with special binaries of `vi`, usually called `vipw`. Again, this is a very risky approach and not for the faint at heart. Use the provided scripts for that purpose—usually called `useradd` or `mkuser` in AIX—or the respective GUI application.

3.2.8 How to Move a User

Assume you have extended the capacity of the system's hard disks and want to move certain users to the newly created filesystems. To perform this move successfully, you have to meet the following conditions. All permissions must be preserved, ownership, group, and access control lists (ACLs) if used. Some *NIX variants change the ownership and permissions based on the user account actually performing the move; this means the files and directories will end up with the `UID` and `GID` of the user-context launching the program that performs the movement. `root` is usually exempted from this restriction; it is applied only to a user-context with `UID` other than `0`. If you have to administer more than one type of *NIX, you should always use the necessary options for the commands to ensure that the move does not change the ownership and the permissions on the moved files or directories.

To actually move or copy the directory structure of certain user account, you could use the commands `cp`, `tar`, and `cpio`. Actually you have some more commands that could be used for the task at hand, but the ones mentioned should be sufficient and are available on almost every *NIX system. Which command is the fastest and best suited for the task is the source of debate among system administrators and very well suited to start a lively discussion during a break at a conference. All the mentioned commands support options to make sure the permissions and ownership settings will not be changed during the transfer.

All programs are started from within the directory that contains the directories of the respective user accounts. Let's have a look at the contenders. Listing 3.30 shows the use of `cp`.

LISTING 3.30 The Use of `cp`

```
root@unixhost> cp -pR silvied /export/adminusers/
```

The `$HOME` directory of the user account `silvied` will be copied to `/export/adminusers`; the option `–r` instructs the program to recursively copy all enclosed directories as well.

Rumor has it, the program `cpio` is supposed to be the fastest of all three. Its real purpose in life is the creation of archives, similar to `tar`. Listing 3.31 shows the usage.

LISTING 3.31 Moving a Directory with `cpio`

```
root@unixhost> find silvied -print | cpio -pudm /export/adminusers/
```

Here the output of `find` is piped to `cpio`. The options are as follows: `p` is the so-called pass mode; the program reads from `stdin`, also known as the *pipe*. `u` is just a precaution (with a potential to create some confusion); the program will overwrite a newer file if it exists. `d` instructs the program to create directories if needed. `m` ensures that the permissions and the ownership will be retained.

The use of `tar` for this task is the traditional method taught to all SysAdmins since the beginning of *NIX. Listing 3.32 shows how.

LISTING 3.32 Moving a Directory with `tar`

```
root@unixhost> tar cf - ./silvied | ( cd /export/adminusers/; tar xBpf
- )
```

This variant looks a bit confusing at first. The upside to this approach is that it will work on almost every *NIX system, regardless of age or make. The first `tar` command reads as follows: `cf` is create a file instead of using a tape drive. `-` means create the file on `stdout`. `./silvied` is a relative path; archives using relative paths can be extracted anywhere. Because `tar` writes the archive to `stdout`, it can be piped to another shell (this is done via the brackets), which changes into another directory using `cd` and executes another copy of `tar`. The options of the second `tar` are as follows: `x` is for extract. `B` is block mode; this means the program reads incoming data until a `tar` block is filled, even if the data comes in really slow. `p` means preserve all permissions, ownership, and ACLs if used. `f` again means use a file instead of a tape drive.

Every method shown has its pros and cons. The common thinking is that `cp` is slow, `cpio` is the fastest method, and `tar` is somewhere in the middle, but works almost everywhere.

3.2.9 How to Move a User to a Different System

If both systems use the same operating system and you want the transition to go as smoothly as possible, you should create the account with the same UID, make sure that all needed groups are using the same GIDs as on the other system, and create a temporary password for the new account.

```
root@unixhost> useradd -c "Silvie" -d /export/home/users/silvied \
-g adminmgt -m -u 632 silvied
64 blocks
root@unixhost> passwd silvied
New Password:
Re-enter new Password:
passwd: password successfully changed for silvied
```

After the account is created, you need to change to the old machine, go into the `$HOME` of the user, and copy the contents via `scp` to the new machine in the new `$HOME` of the account.

```
root@unixhost> cd /export/home/users/silvied
root@unixhost> ls -al
total 21
drwxr-xr-x   12 silvied  adminmgt      512 Aug 20 18:57 .
drwxr-xr-x    7 root     other         512 Aug 20 18:09 ..
-rw-------    1 silvied  adminmgt       73 Aug 20 18:25 .TTauthority
-rw-------    1 silvied  adminmgt       98 Aug 20 18:25 .Xauthority
-rw-r--r--    1 silvied  adminmgt      136 Aug 20 18:23 .cshrc
drwxr-xr-x   11 silvied  adminmgt      512 Aug 20 18:51 .dt
-rwxr-xr-x    1 silvied  adminmgt     5111 Aug 20 18:25 .dtprofile
drwxr-xr-x    3 silvied  adminmgt      512 Aug 20 18:25 .java
drwx------    5 silvied  adminmgt      512 Aug 20 18:51 .netscape
```

```
-rw-r--r--    1 silvied  adminmgt      210 Aug 20 18:15 .profile
drwx------    2 silvied  adminmgt      512 Aug 20 18:25 .solregis
drwxr-xr-x    3 silvied  adminmgt      512 Aug 20 18:23 docs
drwx------    2 silvied  adminmgt      512 Aug 20 18:49 nsmail
drwxr-xr-x    2 silvied  adminmgt      512 Aug 20 18:56 templates_DE
drwxr-xr-x    2 silvied  adminmgt      512 Aug 20 18:57 templates_GB
drwxr-xr-x    2 silvied  adminmgt      512 Aug 20 18:56 templates_US
drwxr-xr-x    2 silvied  adminmgt      512 Aug 20 18:19 to_be_backed_up
root@unixhost> scp -rp * silvied@192.168.0.2:.
silvied@192.168.0.2's password:
root@unixhost>
```

If the operating system on the new machine uses `/etc/shadow` to manage the passwords, you could copy the ciphered password from the old to the new machine. Other than the new IP address or the new name of the machine, everything will be the same. The author tested the copying of the ciphered password successfully on Solaris. If the copying of the password is not possible, just tell the user the new password, and the transition is done.

3.3 DAILY TASKS

The next sections describe actions that require constant care on a daily basis.

3.3.1 Check the Uptime

One thing to check every day is the output of the command `uptime`. `uptime` reports the current time, the time the machine has spent up and running, the number of users currently logged in, and the load average. The time up and running is of interest here; machines supposed to run 24/7 usually show an uptime spanning weeks or even years (the uptime that impressed the author the most was 4 years and 6 months). A quick look at uptime tells you if the machine was rebooted, and if the reboot was not planned, an investigation of what happened is in order (see Section 3.3.2, "What `syslog` Tells You"). Most servers start up automatically after, for example, a loss of power, and `uptime` is the easiest way to check if anything happened. Listing 3.33 shows an example of `uptime`.

LISTING 3.33 The Output of `uptime`

```
root@unixhost:>  uptime
```

```
10:47am  up 27 day(s), 50 mins,  1 user,  load average: 0.18, 0.26,
0.20
```

3.3.2 What `syslog` Tells You

A well-configured `syslog` is the most important source of information on any *NIX system (see Section 5.2.2 "Configuring `syslog.conf`"). As long as you can rely on the fact that every process in need is able to send a message to the administrator of the system, the files written by `syslog` are the premier place to look for hints if you suspect problems with the machine.

The directory the files are stored in is usually `/var/log`. Solaris uses two directories for that purpose, `/var/adm` and `/var/log`. Most messages should end up in a file called either `messages` or `system.log`, depending on the configuration. You should make it a habit to periodically issue the command shown in Listing 3.34.

LISTING 3.34 Checking the `syslog` Main File for Error Messages

```
erikk@unixhost> grep -i error /var/adm/messages
Aug 24 14:15:19 unixhost efs_timed[1068]: [ID 923199 daemon.error]
SunScreen time daemon waking up on signal 1 (SIGHUP)
erikk@unixhost>
```

The error message shown in Listing 3.34 is the result of a shutdown of the firewall (this happened on our test system). No other messages contain the string error, regardless of capitalization. Everything seems to work fine.

If the firewall of the system is configured to write its messages into another file, this file needs to be checked as well. Listing 3.35 shows the output on a Solaris machine.

LISTING 3.35 Firewall Log (Solaris)

```
# ssadm log get | ssadm logdump -i - -t a logwhy 256
  1     hme0 (256: deny rule or no pass rule)2004/08/26 17:49:47.524365
      osxhost -> unixhost         TCP D=317 S=54566 Syn Seq=809212372
Len=0
 Win=3072
  2     hme0 (256: deny rule or no pass rule)2004/08/26 17:49:53.528728
      osxhost -> unixhost         CSNET-NS C port=54567
  3     hme0 (256: deny rule or no pass rule)2004/08/26 17:49:53.529017
      osxhost -> unixhost         TCP D=605 S=54567 Syn Seq=2134825783
Len=0
 Win=3072
  4     hme0 (256: deny rule or no pass rule)2004/08/26 17:51:21.841873
```

```
        osxhost -> unixhost          TELNET C port=50296
     5     hme0 (256: deny rule or no pass rule)2004/08/26 17:51:24.629654
        osxhost -> unixhost          TELNET C port=50296
```

The author executed a quick `nmap` scan of the system in question to create the entries.

3.3.3 Check the Number of Mails

As already mentioned in Chapter 1, you need to have a quick look at the mail statistics as part of your daily routine, as shown in Listing 3.36.

LISTING 3.36 Checking the `mailstats` Output

```
erikk@unixhost> mailstats
Statistics from Wed Aug  4 23:02:01 2004
 M   msgsfr  bytes_from   msgsto    bytes_to  msgsrej msgsdis  Mailer
 3       19         23K       19         27K        0       0  local
=============================================================
 T       19         23K       19         27K        0       0
 C       15                    0                    0
```

3.3.4 Checking the Baselines

If the system in question is equipped with some of the scripts presented in the beginning of this chapter, then you should have a look on the graphs as well. If the log files are rotated before midnight, then the existing data should provide a good picture of how the system has behaved since then. If there are no specific spikes in the counters, all should be well. Here a couple of hints what to look for, though.

If the machine should run an extensive backup after midnight or the database reorganization is started in this timeframe, heightened activity on the wait I/O counters is quite normal. If you should encounter a constant noise on these counters, then something needs to be done. You should try to find out why the system is complaining about nonresponsive hardware; the disks especially need to be checked.

Another counter you should watch is the `runq-sz` (runqueue size). Continuous spikes in this graph indicate that processes are waiting to be executed. A certain number of waiting processes are normal. However, if there are bigger spikes, then you need to evaluate the reason for that reading. As long as you have enough resources available, you should check if the load on the system has increased by using the older graphs or datasets. If the increase is a trend over the last couple of days, then either new software was started, taxing the system considerably, or other processes spawn faster at the moment. If this is the case, you need to find out which processes are responsible by using either `top` or `ps`.

This approach is called *preventive maintenance*. The name stems from the fact that all problems that can possibly occur on a computer system usually start small. This means there are indicators that something is wrong, and it is your job to find out what. Using this technique, supported by the baseline data, enables you to react to problems before the users of the system notice anything (see also Section 3.3.6, "Customer Relations," later in the chapter).

3.3.5 Security Patches

No operating system is perfect. This revelation may be shocking, but there you go. The *NIX architecture usually prevents the worst surprises, but some threats still require the patching of the system. Your only line of defense is to make sure your systems are current.

Make it a habit to check the relevant lists or vendor Web sites at least once a day. If there are two to four *NIX brands in production, the time required should be something in the line of 30 minutes. These 30 minutes are well spent compared to the effort to repair a system that has crashed because of a hardware error that could have been prevented if the necessary patches had been installed. The same holds true for a security breach that occurrs after the manufacturer had released the specific patches.

If there are patches to install, do not rush to install them without testing, though. If possible, test the patches on a system that will not interrupt the flow of work inside the company. Try to read all the documentation that accompanied the patches, especially the readme files; they usually contain the necessary information about possible side effects. Test the patches with all software packages that are installed on the production systems. If a patch does not work as expected on the test system or breaks a program you are relying on, no harm is done. One should contact the manufacturer of the software to try to find out if there is a patch or upgrade for the program in question. Most third-party software developers work in a close relationship with the manufacturer of the operating system, so they should be aware of any problems. The two parties usually consult before they release any new patches, but it does not hurt to ask.

As already mentioned, you should use a test system to test the patches if possible. It is advisable to maintain such a system; see Section 6.1, "A System for Testing," for more information on test systems. Using a test system is especially important if the software used on the productions systems was customized, altered in one way or the other, and does not completely resemble the binary on the installation media. The test system will be covered later in the book, but the vital information you need to know for now is this: the system does not have to be an exact mirror of the production system. It only needs to be able to run the operating system used and the software in production, not necessarily at the same speed, though.

MAINTENANCE CONTRACTS ARE IMPORTANT

If possible at all, you should try to get a maintenance contract for all production systems with the manufacturer of that system. Most manufacturers of *NIX systems limit the information available to their customers if they do not have a maintenance contract. This is not an act of discrimination, but common sense. If the need arises to install a certain set of patches with known side effects, then the availability of support by the manufacturer is an absolute necessity. If needed, a trained technician can be present in case something goes wrong.

The author's experiences with that kind of support are really good. One case required the patching of a five-year-old Solaris system. While applying the patches, it became obvious that they triggered an error inside the used RAID controller. Applying the patches was paramount, though. To keep an old database application up and running, the application required an outdated release of the database system, not certified for the system's patch level. The maintenance contract helped to solve the issue. A technician sent by the manufacturer changed a faulty controller over the weekend. Then the patches were applied, and the system was up and running on Monday again.

This does not necessarily mean every system administrator will encounter things like the one described, but always remember Murphy (the pilot not the scientist): "If something can go wrong, it will, at the most inconvenient time." Installed applications on *NIX systems become more and more interwoven with the rest of the system and other programs; make sure to get every support option available if the system is mission-critical.

Your best bet is a maintenance window, a predefined time every week or day, if necessary, in which all involved persons expect a downtime of the systems. If you should not need the window, that's even better. The superior architecture of a *NIX system allows you to apply most patches while the systems are doing their job, contrary to the "other" OS that requires you to reboot after every applied patch. Still, applying patches while other users are trying to work might put a noticeable strain on the performance of the system. Another advantage of the scheduled downtime is the fact that you are able to watch the system while nothing but the OS is running, thus allowing a valid assumption about the state of the system. If something shows up during that time, you are free to investigate and take precautions to keep the systems healthy; you are even able to switch to single-user mode without disrupting any user processes to take a closer look. Keep your users informed about changes to the systems; they rely on you to keep them up and running. That does not mean you should "swamp" your non-technical users with acronyms they do not understand, but do keep them informed in non-technical terms of what is going on.

3.3.6 Customer Relations

Avoid the "Thou shall not bother the *NIX guru" attitude. These people are your customers; gain their trust. Drop by now and then, ask them if everything is working as expected, and take notes if it does not. This information might point to an error condition that will not show up for some time. If you are positive that there is nothing to worry about (see Section 3.1.1, "Why Baselining?", earlier in the chapter), try to help your customer anyway.

We live in interesting times; all departments try to shield their budgets from cuts and work on deflecting them to some other place in the company. One of the reasons why the IT budget could come under fire is the fact that there are no "known" faces behind the department. IT enables the company to do its work, but if the company cannot do what it is supposed to do, they would not need an IT department in the first place. So, repeat with me: "The users are our customers; if we help them to do a good job, our budget is a little more secure."

This idea may seem to overdo it a little. It sounds like some kind of marketing stunt, actually. But truthfully, the users and the IT department are in a symbiotic relationship; they need each other.

3.3.7 Management Relations

The other side of the coin is the relation between the IT department and upper management. You need their trust, too. They are supposed to allow you to spend money on upgrades or new equipment. Make sure they understand that there is a valid business case behind your demands. Keep them informed about what is going on as well. You will have to change the perspective, though. Your users (see the preceding Section 3.3.6, "Customer Relations") are interested in getting faster connections, more disk space, and faster workstations; upper management is interested in keeping the running costs as low as possible. If you need to convince them to increase your budget, you have to prove that business will run smoother and they will gain in the long run. Of course, gaining in the "short run" is even better. To argue along those lines, you should arm yourself with solid numbers (refer back to Section 3.1.1, "Why Baselining?", earlier in the chapter).

3.4 SUMMARY

This chapter discussed aspects of the big topic called maintenance. It started with a discussion about the baselining technique and how the gathering of this data is required to start the preventive maintenance cycle. It explained why the gathering of baseline information is vital for the system administrator to make informed decisions about the health of the respective systems. The chapter continued with the topic of users and groups; it explained the underlying ratio behind the management of user and groups on *NIX systems and how to move users around on the system without disrupting their work. It closed with a discussion about the daily tasks a system administrator should perform and the importance of the relations between the users of the systems and the IT department.

This page intentionally left blank

4 Backup

In This Chapter

- Backup Strategies
- Tips Regarding Backup

Let's face it—hardware tends to fail eventually, and the types of failure can be mind-bogglingly complex. Accepting this fact is the first step toward a more secure system—*secure* in the sense of availability and not in the sense of computer security. The range of possible hardware accidents spans from a real hardware error to things like flooding, fires, and worse. This is not to imply that your building will collapse sooner or later, but it is the responsibility of a system administrator to be aware of the fact that it *could* happen. The data stored on the computers represents a substantial value to the company you are working for; at times, a loss of this data means the whole operation is out of business. A system administrator is supposed to take precautions, test them, implement them, and refine them on a recurring basis. This chapter deals with certain aspects regarding working with backups.

4.1 BACKUP STRATEGIES

You have to have some kind of strategy—*what* will be backed up, *when* will it be backed up, and *where* should it be backed up. You also have to have a way of testing *if* the backup was successful. A sound backup strategy should contain layers, thus building multiple levels of security. One important aspect is the *where to* back up (see Section 4.1.6 later in the chapter for further discussion). This aspect alone requires careful planning and the inclusion of a vast array of contingencies. Based on your geographical location, access to a full backup of your systems could be a necessity as well.

"We spent so much money for the new RAID system that the data is most certainly secure. Why do we need to spend money on a tape drive, anyway?" A statement like this is no exaggeration; When you hear it, one way to answer is to explain the *GIGO* principle to the cost-conscious members of your organization; GIGO means "garbage in, garbage out." In other words, if someone overwrites the valuable files on the shiny new RAID system with garbage, then that garbage is stored safely, and the contents of the original file are still lost. The main issue here is the fact that many marketing brochures try to sell the advantage that data stored on their respective product is stored that safely, so safely that the prospective buyer can reduce the number of backup cycles. Depending on the wording used in a brochure, someone with a casual knowledge searching for ways to cut costs might fall for it. This is not to say that there are outright lies in theses brochures, nor that controllers on their ongoing crusade to maximize profits don't care, but the one who takes the blast if something goes horribly wrong is you—you already guessed that, didn't you? It is *your* job to "sell" a sound backup strategy to your managers (see Chapter 3 for more on relations with management).

The backup strategy and the needed expenditures for hardware and media should be based on solid numbers. The time you invest to make a convincing case during the presentation of your ideas gives you the chance to evaluate your ideas thoroughly and represents time invested in a better quality of sleep on your side.

4.1.1 Daily Backup

Try to identify all files on your system that change on a daily basis and that you cannot afford to lose at any given time. Talk to the departments in your company and ask the relevant persons to estimate the value of those files and the required effort to recreate them in case of a loss. Let them also give you a good estimate about the timeframe these files would need to be recovered within given a worst-case scenario.

If the accumulated size of the files exceeds a certain limit, for example, the files of the company's database system or something similar in size, try to evaluate with the manufacturer of the application or the database if there are recommendations regarding how to deal with backups. Huge amounts of data are usually backed up in an incremental manner; this means only the changed portions of the files are written to the backup media. Try to estimate the required time for a recovery if the worst happens. Is it still possible to do a recovery in the timeframe given to you by the respective department?

Another very important point is the so-called "retention period," the time required by law or otherwise for which the backup has to be retained. The answer to this question tells you when you are able to "recycle" the backup media, but make sure you don't exceed the recommendations given by the manufacturer. Another aspect concerning the retention period to keep in mind is the fact that the backups do not only have to be available, but *readable* as well. This means that there has to be a device to read the backups back from the respective media. Often the author has been confronted with situations in which the media is stored safely somewhere, but the devices required to read the media did not exist anymore. This situation could pose a serious problem if the backup contains financial data required to prove that a certain transaction actually took place, for example.

4.1.2 Weekly Backup

A full backup should be created once a week if possible, especially if incremental backups are created during the week. Recovery of an incremental backup means restoring one day after the other until the latest state is available again. A worst case scenario would be as follows: the last full backup was done on Sunday. It is Friday, and the file in question was modified every day; all tapes created during the week, including the full backup, have to be used to recreate the file. Calculating the time, depending on the speed of the tape drives and the fill grade of the tapes, is definitely an exercise you have to complete. You have to make sure you are still able to stay in the given timeframe regarding a recovery.

Information that needs to be available all the time is no candidate for incremental backups. Examples of this sort of information might be drawings from the CAD/CAM system the company uses that are required by the engineers to be able to work at all; configuration files that are used by a communications system, if the company works in this area; data required to interact with the customers of the company; and the like. You should try to back up these files as soon as they are created. If this approach should not be feasible, then the backup should be created as soon as possible. Run periodic checks; if this kind of data creeps into the incremental backup, then try to move it to the dailies. Again, there is data crucial to the daily operations of a company, and this data requires special approaches regarding backup.

4.1.3 Monthly Backup

Creating a monthly backup—while not replacing the last weekly backup in the month—could be an advantage, as well, provided it can be written without disrupting the daily business. It should be considered a *backup of the backup*.

Most manufacturers of backup-related software offer the means to create a copy of a backup, if you are willing to pay for this privilege. This feature usually gets unlocked after the fee is paid. Should you choose this feature, you should not forget to add the cost of another tape drive or whatever is used to write the backups. The creation of a copy of a full backup usually requires additional hardware, because the size of the backup obviously exceeds the amount of free storage space.

4.1.4 Off-Site Backups

A teacher the author knew used to say, "The fireproof closet is of no use at all if it is buried under the remainder of the building." Keep this statement in mind, especially if the backup media is stored in the same building in which the systems are located. This is not meant to make your paranoid, but just to recommend you try to store the backup media as far as possible from the systems inside the building. If a fire destroys the data center, chances are you will still have the backup and some means to use a replacement system to keep the company working. An even better solution is to store the backups in a bank vault or a branch office of the company (the branch office should store their backups at another branch office).

How far away you store the backups depends on the surroundings of the company's location. Companies in San Francisco or Los Angeles, California, usually store the backups in a building on the other side of the San Andreas Fault. If your area is geologically safer, at least try placing the backups in another part of the city. Even if you think that you have no reason to worry about things like earthquakes or fires, nobody has gotten fired, so far, for providing a sound backup strategy. The company relies on the judgment of the system administrators to create this strategy.

4.1.5 The "Backup Window"

As already mentioned, the timeframe you are able to create the backups in, the *backup window*, plays an important role regarding the backup strategy. If the backup run should slow data processing in the company down to a crawl, the backup strategy needs to be adjusted accordingly. You have to find a backup window to minimize the disruption of the daily business while still allowing the creation of a usable set of backups. As a rule of thumb, expect that the faster the backup, the more expensive it will be. It's part of your job description to find a solution all sides can agree on.

Many *NIX systems know how to create so-called *snapshots*. This means the file system will be frozen at a predefined point in time, allowing the user to create a consistent image of it. All modifications of the files during the time the backup runs are written into a buffer and will be applied as soon as the filesystem is unlocked again. The users won't notice anything, provided there is enough space dedicated to the buffer used to record the changes. You should make sure to create the backup in the fastest possible way, though. All modifications have to be applied after the unfreezing, thus increasing the load on the disk subsystem. Some RAID systems allow you to do this in hardware, provided you are willing to pay for this feature.

4.1.6 Backup Media

Another important thing to consider is the media to which the backup is written. Depending on the strategy, the so-called retention period needs attention. The question here would be as follows: "How long do we need to keep the backup?" Different media have different storage requirements, regarding temperature, space, or accessibility.

4.1.6.1 CD-R/CD-RW/DVD-R/DVD-RW

In theory at least, any CD/DVD-type media can be read back on almost every computer system. That does not necessarily mean that the data can be processed on that system, but it can nevertheless be accessed and copied to a machine that is able to work with it. For example, you could use a Macintosh to read crucial data stored on a CD/DVD and copy it back to the system where the data would be processed. The only thing to keep in mind is that some operating systems cannot deal with every possible combination of filenames and paths (some Windows variants cannot read pathnames longer than 256 characters) or so-called 2-byte characters in filenames, for example, Chinese or Japanese characters. Regarding *NIX, most systems exceed the 256 characters in a path either during install or after a couple of weeks. There is no use in trying to read a CD/DVD on a system not capable of dealing with 2-byte characters if the filenames contain non-ASCII strings. Apart from these mentioned downsides, storing the backups on CDs/DVDs seems to be a good way to handle them.

The downside of CDs/DVDs is the fact that they don't "live as long" as tapes. Depending on manufacturer specifications, the lifespan of a CD-R/CD-RW/DVD-R/DVD-RW is between 15 and 30 years, provided they are stored standing, with no direct sunlight, in a case that doesn't put to much strain on the plastic surface of the disk. The lifespan can be reduced dramatically if the wrong type of marker pen is used on the surface and then there is "CD/DVD rot." Not that the author has ever witnessed a "rotting" CD/DVD, but you do not want to find it eating away the only copy of a file, so don't take chances.

Some specialists suggest that creating a copy of the CD in question every five years is the best bet. Again, your mileage may vary, but a little paranoia might save the day.

4.1.6.2 Tapes

Tapes on the other hand should last about 30 years, and the capacity is growing every year. Based on the retention period, it might be feasible to copy the "old" backups to a new type of tape and check the results to make use of the higher capacity when upgrading the hardware. Depending on the software that was used to write to the tape, the only prerequisite is to have the software and the hardware to read it back in available. If a backup software package like "Legato NetWorker®," for example, was used to create the backup, a copy of the software able to run on the existing hardware is needed to access the data again. So it all boils down to the questions: "How long do you need to access the backups?" and "Are you still able to access them with the soft-/hardware at hand?" The purchase of a new tape drive, possibly using a new type of tape or format to write to it, requires careful planning. Will the old hardware be accessible in the future? Is there a plan to make sure the old tape drive will still work in a couple of years? Sometimes, if the backup needs to be retained, copying the data to a CD or DVD will be the best bet to make certain you can read it in the future. You just have to make sure to use a format that is readable with the current (and probably future) software and/or operating system. The same holds true for hardware. If the manufacturer declares the "End of Life" for a certain piece of hardware, this means the system won't be supported in the not-too-distant future. You have to make sure either to provide the means to be able to still read the media or to copy the backups to another media type that can be used after the manufacturer has stopped supporting the old hardware.

NOTE

Which type of tape should you use? There are many different types of tapes and tape drives on the market. Make sure the manufacturer of the ones you use to back up your valuable data will still exist in a couple of years.

*Given the current state of technology, DLT or AIT seems to be the best bet. There is one drawback regarding the use of DLT tapes, though: the data must be written in a continuous stream. Otherwise, the drive mechanism will suffer. There shouldn't be any problems while using *NIX as an operating system, though. To quote a technician: every DLT drive connected to a Windows system is another drive sold. This quote holds some value, given certain technical observations the author has made, but there isn't any positive proof. The same technician said the incentive to create the AIT system was that certain operating systems are not able to create a continuous stream of data and the involved companies were fed up with the complaints they had to deal with. The author is still trying to get a confirmation for that statement.*

4.2 TIPS REGARDING BACKUP

Certain files and directories should be backed up independent of the backup strategy used. They are not necessarily written to another backup system, but are stored somewhere accessible if the need arises.

4.2.1 `tar` Important Files

Every *NIX system uses configuration files that change over the lifespan of the system. These files must be copied back if the system needs to be reinstalled. If one of these files gets corrupted or overwritten, it needs to be replaced by a working copy as soon as possible.

Some of these files are used for special purposes on certain systems; others are equally important on every *NIX system. The script shown in Listing 4.1 stores a copy of these files in a directory, and that directory is written to a `tar` archive for storage purposes.

LISTING 4.1 Script to Back Up Important Files

```
#!/usr/bin/sh
#$Id: backup_conf.sh,v 1.1 2004/08/18 11:15:26 root Exp root $
###################################################
#
#       file:           backup_conf.sh
#
#       purpose:        backup important files to
#                       directory and tar it
#
#       author:         Erik Keller
#
#       parameters:     none
#
##################################################

# Variables
SYSNAME=`uname -n`
DESTDIR=/.HO/backupconf
TODAY=`date +%y_%m_%d`

# get the right tar (needed in Solaris only)
if [ -x /opt/sfw/bin/tar ]
then
        TAR=/opt/sfw/bin/tar
        TAR_OPTS=czf
        TAR_EXT=tar.gz
else
        TAR=/usr/bin/tar
        TAR_OPTS=cf
        TAR_EXT=tar
fi

# Files to backup
FILES="
/etc/resolv.conf
/etc/nsswitch.conf
/etc/group
/etc/inet/hosts
/etc/inet/inetd.conf
/etc/logadm.conf
/etc/syslog.conf
/etc/system
/etc/vfstab
"

# Directories to backup
DIRS="
/etc/init.d
/etc/rc0.d
/etc/rc1.d
/etc/rc2.d
/etc/rc3.d
/etc/rcS.d
/etc/default
/etc/ssh
"

# function to copy the files in $FILES
copy_files()
{
        for FILE in $FILES
        do
                if [ -f $FILE ]
                then
                        DESTFILE=`echo $FILE | tr '/' '_'`
                        cp $FILE $DESTDIR/$SYSNAME/$SYSNAME.$DESTFILE
                        #echo $DESTDIR/$SYSNAME/$SYSNAME.$DESTFILE
                fi
        done
}

# function to copy the directories in $DIRS
copy_directories()
{
        for DIR in $DIRS
        do
                if [ -d $DIR ]
                then
                        DESTFILE=`echo $DIR | tr '/' '_'`
                        cp -r $DIR $DESTDIR/$SYSNAME/$DESTFILE
                fi
        done
}

# function to create the tar archive
make_tar()
{
        cd $DESTDIR
        $TAR $TAR_OPTS $SYSNAME.$TODAY.$TAR_EXT $SYSNAME
}

# function to check if the directory structure is in place
dircheck()
{
        if [ ! -d $DESTDIR ]
        then
                mkdir -p $DESTDIR
        fi

        if [ ! -d $DESTDIR/$SYSNAME ]
        then
                mkdir $DESTDIR/$SYSNAME
        fi
}

# main program
dircheck
copy_files
copy_directories
make_tar

# EOF
```

The script defines the variables $SYSNAME, $DESTDIR, and $TODAY. $SYSNAME contains the name of the system and $DESTDIR the path of the directory to which the files should be copied. The latter should ideally point to a directory on another partition or, better yet, another hard disk. You should keep in mind that the files contain valuable information, enabling a possible intruder to compromise the system. They should be stored in a location only superusers have access to. $TODAY contains the current date, used as part of the name of the resulting tar archive. The archives won't overwrite themselves this way, allowing you to create some kind of configuration history.

The choice of which tar program to use is Solaris specific. If the script is deployed on another *NIX system, you could use the path and name of the tar program installed there. If the system contains more than one tar program, customize the example function to make sure to create consistent archives.

The variables $FILES and $DIRS contain all files and directories to be backed up. They vary from system to system. If the apache Web server is used, for example, you have to add the directory containing its configuration files to the $DIRS variable. The same goes for other configuration files important for the specific system.

The copy_files() function tests whether the respective file exists. If it exists, it copies the file and changes the filename in the following manner: /etc/syslog.conf, for example, becomes unixhost_etc_syslog.conf and so forth. This provides you with the means to identify the system the file was originally used on. Testing for the existence of a file allows you to deploy the script on similar systems without the need to customize it extensively. If the file does not exist, it won't be copied, and there will be no error message. One requirement for this system to work as intended is no typos in the list of files. This shouldn't be a problem because scripts should be tested thoroughly before deployment, anyway.

The copy_directories() function more or less mimics copy_files(), but does not add the name of the system to the directory. Saved directories in $DESTDIR/$SYSNAME start with an underscore, _.

The make_tar function creates the tar archive. It does not seem to be necessary to wrap the creation of the archive in a function, but doing so allows an easier extension of the script later.

The dircheck() function tests for the existence of the used directory structure in $DESTDIR and creates the directory should the test fail.

The real script starts after the comment # main program; all functions are executed in order of appearance. If the script is deployed by executing it via cron, writing an entry into a log file is a good idea. Most *NIX systems use the program logger for that purpose. It is used to send messages to the syslogd process. You could simply add the following line at the end of the script:

```
logger -p user.crit $0 done.
```

This command executes the following steps: the string `backup_conf.sh done.` is sent to `syslogd` with the priority `user.crit`, and `syslogd` acts on it according to the configuration found in `/etc/syslog.conf`. In the case of `user.crit` this is usually by writing it in the main log file for the system (see Section 5.2.2 "Configuring `syslog.conf`"). The `$0` variable is set automatically to the name of the script.

4.2.2 Backing Up Index Files

Every commercial backup software stores an index to the contents of the backup media in an index file on the backup server. Legato NetWorker uses `/nsr/index`, for example. It is advisable to store theses directories on a CD/DVD or another machine, as well. Provided there is enough storage space, the latter approach is the most efficient. You should simply create a `tar` archive of the directory and should not forget to include the configuration files. If the worst should happen—you need to reinstall the backup server—you could copy the contents of the archive back on the newly installed system, and the system should be up and running again. If you don't have access to this kind of backup, the index files will need to be restored from the backups themselves. This means the process could take a long time to complete. As long as the setup of the new backup server is an exact mirror of the old machine, the copy on the other system should be all that is needed.

Considering the amounts of money a full disaster recovery will cost the company (services unavailable, billable working hours of the technicians, lost working hours of the employees, etc.), even the investment in a dedicated system (see Section 4.2.5 "Back Up to Another System" later in the chapter) or an external hard disk could pay off within a reasonable timeframe.

4.2.3 Redundant Backup Hardware

The topic of this section is between a tip and a backup strategy. Assume there are one or more tape drives built into a *jukebox*, a device containing one or more tape drives, and a mechanism in the jukebox is feeding the tapes to those drives. If there is no other jukebox of the same make, you have a *single point of failure*. There is no way of performing either a backup or recovery if a mechanical error occurs in the mechanism feeding the tapes to the drives. Of course, you could get access to the tapes in another way. For example, the author has witnessed another system administrator using a crowbar to open the housing of a jukebox; the housing of the mechanism was very expensive, but the company came to a standstill without access to the data on the tapes. This means you can still use a single drive, similar to the ones in the jukebox and less expensive for that matter, to do backups or recoveries by hand until the defective unit is serviced.

If you think we are exaggerating here, keep this in mind: backup is expensive, but losing your data for good is much more expensive. Being in a position to exchange a defective tape drive inside the jukebox with the external unit is just an added bonus. Even if the maintenance contract covers a replacement of the defective unit within 4 to 8 hours, would anyone risk losing the data over the availability of a piece of hardware in two to three years time?

4.2.4 Backup Using `rcs`

You can use the *Revision Control System*, part of most *NIX distributions, on the installation media, to check in important files. This is no real backup, but is a good way to give you some peace of mind.

There is more than one way to use `rcs` for the task at hand. Either you could create a directory to hold all configuration files put under revision control, modify the files in there, and copy them to their desired location after changing them, or you could create `RCS` directories in all places they are needed, for example, `/etc`.

Do as you prefer, but make sure all files get checked in after every change and write a descriptive log message. Everyone will be able to figure out who did what and the reason for the change. You can use `rlog` to display the revision history, create a living documentation in that respect. Listing 4.2 demonstrates the use of `rcs` with a configuration file.

LISTING 4.2 Check In Hosts

```
root@unixhost> ci -l hosts
RCS/hosts,v  <--  hosts
new revision: 1.37; previous revision: 1.36
enter log message, terminated with single '.' or end of file:
>> added alias linuxhost for 192.168.0.2 (Erik)
>> .
done

root@unixhost> rlog hosts
...
----------------------------
revision 1.37   locked by: root;
date: 2004/08/27 04:35:36;  author: root;  state: Exp;  lines: +3 -1
added alias linuxhost for 192.168.0.2 (Erik)
----------------------------
...
```

If the change doesn't work out the way it should have, you can get the former working version out of `rcs` with `co -r <revision> <filename>`. The newer version is still available from `rcs`, so you can find out what went wrong and rectify the configuration.

4.2.5 Back Up to Another System

If you compare the price of a jukebox, a device containing one or more tape drives and a mechanism to feed the tapes to them, with a dedicated *NIX system including a disk array, chances are the difference is not that big. The dedicated system wins often, actually. Backing up to hard drives has a downside, though. They are not as resilient as DLT or AIT tapes, and they usually stay in the system.

This is not to say that it is cheaper to do all backups to hard disks in the long run, but if you have files with retention periods measured in weeks, or sometimes a month, hard disks might be an option. If that data cannot exceed a certain size, or at least shouldn't, then backing it up to a hard disk could be financially viable.

This approach could prove useful in another respect depending on the type of data. Imagine you copy the files of a file server to a smaller, less powerful system with the same capacity regarding hard disks on a periodic basis. If something bad should happen to the real file server, you could switch to the smaller system. It won't be as fast, but at least you will be able to keep the service up and running. The data is only as current as it was during the last copy run, but the users can at least access their files. This approach buys some time to recover the real system.

If you are considering this approach, make sure the system you intend to copy to is able to deal with the filenames used, 2-byte characters, length of the paths, etc. There are more potential candidates than just file servers, though: the index files written by the backup software (see Section 4.2.2 earlier in the chapter), `$CVSROOT` on the development server, or the data files used by the database software. The data files used by the database system can be especially interesting. The size of many databases would allow you to store them on much cheaper systems; only the lack of processing power and network interfaces prohibits their use. One thing cannot be stressed enough, though: this approach enhances the *availability* of the data; it is *not* a replacement for a real backup. Remember the GIGO principle (garbage in, garbage out)?

This is an example utilizing the concept for enhancing the availability of an Oracle database. The database runs on a really big *NIX box; the backup window is 4 hours, starting 5 minutes after midnight. You have to create a script to mount the directory tree the data files will be saved to via NFS and launch `RMAN` (Recovery Manager), a part of Oracle. `RMAN` uses its own configuration to run the backup and write the resulting files to the NFS mount. The script should unmount the backup directories after a successful completion.

In case of a database error, the directories can be mounted again, and `RMAN` can be used to, for example, recover a corrupt tablespace (A *tablespace* is kind of a virtual filesystem Oracle uses to store tables and indices, usually distributed over a number of so-called data files. These data files are regular files in the *NIX filesystem).

```
#!/usr/bin/sh
```

```
#################################################
#
#     file:          oraback1.sh
#
#     purpose:       mount nfs-volume and start
#                    run_rman.sh to perform backup
#
#     author:        Erik Keller
#
#     parameters:    none
#
#################################################

# Variables

mount osxbu1:/Volumes/backups/ora_1 /data1/db_bu

/.HO/scripts/run_rman.sh

umount /data1/db_bu

# EOF
```

The script connects the database server with the system `osxbu1` by mounting the NFS share `/Volumes/backups/ora_1` and executes another script to launch `RMAN`. It disconnects the NFS share after completion. The NFS mount could be permanent as well, but because it is used only during backup, the connection gets terminated for security reasons. This allows for easier maintenance of the backup system; even exchanging it is possible without disrupting the operation of the database server, because the backup system's presence is required only during the time after midnight. The replacement backup system just needs the same or bigger size of NFS share, exposed by the same name.

Engaging in a deeper discussion about `RMAN` is beyond the scope of this book. If you are using an Oracle database greater than 9i R2 and don't use `RMAN` for recovery management, take a look at the program. Don't believe the rumors; the program is not what it used to be in earlier releases. It is usable and is an indispensable tool if your job is to keep the database up and running.

4.2.6 Cloning the Boot Disk

There is no way of booting a system if the hard disk containing the operating system is damaged. If the data is stored on a separate hard disk, you could replace the defective operating system disk and reinstall and restore the contents of the OS.

This is another reason to separate the operating system and the data created on the machine, by the way. However, swapping out the defective hard disk and reinstalling the OS is time consuming. Wouldn't it be nice to have access to a copy of the boot disk—not a mirrored disk, but a real copy? If you have the means to mirror the disks, you should do so, but you should always keep in mind that a corrupt file on one of the mirrors is just stored safer in its corrupt state, a deleted file is deleted on all mirrors. Remember: GIGO.

Almost all *NIX-based systems provide the command `dd`. This program allows you to create an exact *clone* of a hard disk, provided the target hard disk is either the same type or at least of the same size. One of the best approaches to increase the availability of a system is to mirror the boot disk *and* to have a duplicate of it. If you are not able to use this approach, for example, because of budgetary constraints, then you should at least try to have a duplicate of the boot disk available. The cost of hard disk space is going down constantly; the expenditure for another hard disk should fit into even the tightest budget. After the system is installed and configured, take the following steps to create a copy of the boot disk:

1. Find out which device points to the boot disk. `df` gives you a listing of the mounted partitions. Pretend the boot disk is on `/dev/dsk/c0t3d0`. The target disk is not mounted; thus you have to know which controller and which ID to use. In Solaris, you could use format to display a list of all known and responding hard disks. The first part of the listing displays all available hard disks. You assume there is a responding hard disk at `/dev/dsk/c1t1d0`. You have to make sure that there is nothing of value on the disk because the next command overwrites the whole disk!
2. To copy the contents of the boot disk bit by bit, you need access to the disk as a whole. *NIX usually uses the slice 2 tag=backup for that purpose. Please check the documentation for your kind of *NIX to see if your OS uses the same approach. Switch the system into single user mode (init S) to minimize the activity on the disk. To access the disks, the so-called raw device is used, /dev/rdsk/. So the command would be as follows: `dd id=/dev/rdsk/c0t3d0s2 of=/dev/rdsk/c1t1d0s2`. When the command returns, the following will have taken place: a complete copy of the hard disk on controller 0, target 3, device 0 will have been written to the disk on controller 1, target 1, device 0.
3. The next step varies depending on your decision either to leave the copy in the system or to remove the disk and swap it in case of an error. If the latter is the case, you are more or less set; the copy contains everything you need to boot the system. If the copy should remain in the system, to avoid swapping hard disks, entries in `/etc/vfstab` or `/etc/fstab`, depending on the OS, need to be changed. Mount the copy into a directory using `mkdir`

/tmp/mycopy; mount /dev/dsk/c1t1d0s0 /tmp/mycopy, open the respective file with vi or another editor using the path /tmp/mycopy/etc/, and change all occurrences of /dev/dsk/c0t3d0 and /dev/rdsk/c0t3d0 to /dev/dsk/c1t1d0 and /dev/rdsk/c1t1d0. The entry /dev/dsk/c0t3d0s7 has to be changed to /dev/dsk/c1t1d0s7, for example. This way all partitions will be mounted from the right device.

4. If you intend to swap disks in case of need, use the same command to boot the system as always. If the copy remains in the system, check the hardware documentation for which commands you need to boot from the copy.
5. Now shut down the system and test your copy. If you need a special command to boot from the copy remaining in the system, take a note and put it into the documentation of the system.

If you have changes applied to the boot disk, you must repeat the procedure. If you should encounter problems with the boot disk, you could boot from the copy and rest assured that everything is back to normal while the other disk gets the necessary repairs. The same procedure works for data disks as well. It can be used to transfer a large amount of data from one system to a similar system. It does not work that way with the boot disk, though. *NIX usually stores the host ID in the operating system during install; this means you cannot transfer the boot disk of one system to another system of the exact same make. Trying to boot a Solaris system with the boot disk of another Sun box will result in an error message. One has to make sure to have a way to differentiate between all the boot disks in the storage facility.

Another advantage is to apply patches or upgrades to the copy first. If something does not work, you can reboot with the untainted system. Just make sure you don't interrupt your users by doing that.

CAUTION

If the system has been compromised, resist the temptation to boot from the copy; there is no way of knowing when the break in happened or what happened before. If you have the shadow of a doubt, ***don't do it!*** *If you are 101 percent sure the copy is clean, think really hard if you want to use it. The tool used to compromise your system could be on the data partition as well. The only way to ensure that a system is clean is to reinstall and, if data needs to be restored from a backup, to check it thoroughly.*

4.2.7 Why Use a Dedicated Backup Server?

If you work in a setup where the server providing the services and the backup server run on the same hardware, take a moment to understand the implications. Many IT environments "grew" in a more organic manner and were not scaled to fit the needs. Somewhere in the past, the decision was made to install the backup service on a machine that seemed to have room for that task. If this machine dies, it doesn't just take the data, or whatever it used to do besides running the backup, with it, but the backup system is gone as well. You have no way to start a recovery session. As simple as the last statement sounds, many IT-related decisions remain pending because of daily stress and overworked administrators, with the inherent promise to "look into that, as soon as time permits."

If you are confronted with a situation like the one described, you should try to find a solution as soon as possible. A system capable of running a backup service for a couple of clients (a client in that respect is every machine, workstation, or server having data that needs to be backed up) in a satisfactory way needs to fulfill only four requirements:

1. There should be a reliable backup device attached to it.
2. The system needs a certain amount of memory and hard disk space.
3. The used operating system should be stable and secure.
4. The system should have more than one network interface.

The backup system should not run additional services. These services would suffer when the system is busy running a recovery session. Configurations using a customized *runlevel* (see Chapter 5 for more on init states and runlevels) to run the backup software are not usable during normal operation hours without shutting down running services. The quote "you only need the backup software during the night" works as long as you do not have to recover a crashed system during daytime.

The operating system to run the backup software on should be as stable and secure as possible. Because backups are usually performed via the network, the OS should be able to deal with more than one network interface and make use of the presence of more than one network interface.

There is one exception to the rule. If you are using clustered systems—this means more than one system is clustered to the others and is not visible to the outside world as an individual system, but as a single system providing services to the network, a so-called virtual system—then running the backup server on it should not be a problem. You just have to make sure to be aware of the impact on the running services and that you have enough capacity to start the backup software during normal operation hours as well.

4.2.8 Check the Backups!

Test periodically whether it is possible to read in old backups. If the read/write head of a tape drive starts moving, it will be a gradual process, and it will take a long time before a noticeable change occurs. If the unit fails to read in an older tape, you should take precautions immediately. Compensating for a misaligned read/write head requires either great skill or money. If you desperately need to get back an older backup written with a misaligned head, brace yourself for the bill to be paid. It can be done, but the companies capable of doing it charge accordingly. If you have to try to misalign the head of a replacement unit just to get to the contents of the tape, the warranty of this device will be voided.

Make it a habit to restore backups older than a month or even older to an empty directory on a periodic basis. If you encounter any errors, the unit should be replaced. If you followed the author's advice and bought an external unit, test it as well. The capability to restore important information is at stake here.

4.3 SUMMARY

This chapter discussed strategies regarding backup and provided information that needs to be taken into account while creating a strategy for doing daily, weekly, and monthly backups. It talked about the necessity of storing copies of the backups off-site. It continued with tips and a script to create backups of files vital to the operation of the system. Some other tips included the creation of a duplicate boot disk and the necessary assessment of the existing backup hardware and infrastructure.

5 Setting Up a System

In This Chapter

- Installation
- How to Ready a System for Production
- Booting a System

The installation of a new system seems to be a simple, if not mundane task, according to the distributors of the operating system software, that is. It's just something you have to do before using your shiny new hardware. There is some truth in that. It is possible to install the OS on a new system in a very short period of time; whether or not the system is fit for the tasks you bought it for is a different matter. If you intend to replace an existing system, then you already know the requirements regarding disk space and probably know what the software to be used needs. Issues regarding the scheme of hard disk partitions actually used should point you to the necessary changes in that respect, and you know the software and are aware of useful deviations from the current configuration. But things look very different if the system is to be used for new services. The lack of first-hand experiences needs to be accounted for. As for the hard disk requirements, you have to rely on the specs provided by the manufacturer. After the new system is installed, it is far from ready to be released in the wild. Its state needs to be changed from "newly installed" to "ready for production."

5.1 INSTALLATION

The new hardware is right in front of you, waiting to be of service—almost. Installing the operating system is job one. Contrary to the claims you might have heard from the manufacturer's marketing department, this process needs proper planning! If you already have experience with the *NIX derivate you want to use, then you know what to look for. If not, you *have* to read the supplied documentation. Taking this time out of your busy schedule is something that you are expected to do. Even if you should know an earlier version of the OS, take at least the time to study the *release notes*; they should contain the changes with respect to earlier releases of the software. Because the actual process of installing a *NIX system differs from one brand to the other, this chapter just contains some remarks of a more general nature.

5.1.1 Planning an Installation

If you intend to install new (and thus unknown) hardware, even if it is some variation of something you already know, then you have to take a look in the documentation. If there is something you need to take care of before the actual installation, you can learn about it there. The system requirements are in the installation documentation. If you need to install additional software, for example, a database application, then you have to consult its documentation as well. Once you have gathered all the necessary information, then you are about to start the *planning phase*.

There is a saying that planning replaces coincidence by misapprehension. If you have never heard it, then take a moment to reflect. If you never used the operating system before, then you should check the documentation—hardware and software—and install it the way it suggests there. Answer all questions asked by the installation program with the provided defaults, but do *not* connect the system to a network, yet. You could use a minimal setup consisting of another machine and a hub but no connection to the outside world, including your internal network, if necessary. If you should find anything unusual or something that needs to be investigated further, take notes and leave the system running after the installation completed without errors. Use *another* system to download the current patches for the newly installed machine. You need to read the documentation that comes with the patches and check for problems regarding the intended use of the new system. Then apply the patches by following the supplied instructions to the letter.

If something should break, no system in production is harmed, but you then know what *might* happen. If the system behaves differently, then you have to document the changes.

Locating New Files?

If you need to know which files have been replaced or added by a patch, the following code might be useful:

```
root@unixhost> touch /tmp/timemark.ek
root@unixhost> unzip /export/downloads/118191-01.zip
Archive:  /export/downloads/118191-01.zip
   creating: 118191-01/
  inflating: 118191-01/.diPatch
  inflating: 118191-01/patchinfo
   creating: 118191-01/SUNWgtar/
  inflating: 118191-01/SUNWgtar/pkgmap
  inflating: 118191-01/SUNWgtar/pkginfo
   creating: 118191-01/SUNWgtar/install/
  inflating: 118191-01/SUNWgtar/install/checkinstall
  inflating: 118191-01/SUNWgtar/install/copyright
  inflating: 118191-01/SUNWgtar/install/i.none
  inflating: 118191-01/SUNWgtar/install/patch_checkinstall
  inflating: 118191-01/SUNWgtar/install/patch_postinstall
  inflating: 118191-01/SUNWgtar/install/postinstall
  inflating: 118191-01/SUNWgtar/install/preinstall
   creating: 118191-01/SUNWgtar/reloc/
   creating: 118191-01/SUNWgtar/reloc/usr/
   creating: 118191-01/SUNWgtar/reloc/usr/sfw/
   creating: 118191-01/SUNWgtar/reloc/usr/sfw/bin/
  inflating: 118191-01/SUNWgtar/reloc/usr/sfw/bin/tar
   creating: 118191-01/SUNWgtar/reloc/usr/sfw/libexec/
  inflating: 118191-01/SUNWgtar/reloc/usr/sfw/libexec/rmt
  inflating: 118191-01/README.118191-01
root@unixhost> patchadd /var/spool/patch/118191-01

Checking installed patches...
Verifying sufficient filesystem capacity (dry run method)...
Installing patch packages...

Patch number 118191-01 has been successfully installed.
```

```
See /var/sadm/patch/118191-01/log for details

Patch packages installed:
  SUNWgtar

root@unixhost> cd /
root@unixhost> find . -newer /tmp/timemark.ek -print >
gtar118191.files
root@unixhost> more gtar118191.files
.
./var/sadm
./var/sadm/install
./var/sadm/install/logs
./var/sadm/install/contents
./var/sadm/install/.lockfile
./var/sadm/pkg
./var/sadm/pkg/SUNWgtar
./var/sadm/pkg/SUNWgtar/install
./var/sadm/pkg/SUNWgtar/save
./var/sadm/pkg/SUNWgtar/save/118191-01
./var/sadm/pkg/SUNWgtar/save/118191-01/undo.Z
./var/sadm/pkg/SUNWgtar/pkginfo
./var/sadm/patch
./var/sadm/patch/118191-01
./var/sadm/patch/118191-01/log
./usr/sfw/bin
./usr/sfw/bin/gtar
./usr/sfw/libexec
./usr/sfw/libexec/grmt
...
```

The example shows the installation of a patch on a Solaris system. Before applying the patch, the file `/tmp/timemark.ek` was created using the command `touch`. This file is used as a template for the `find` command, using the `-newer` option. This way all files newer than `/tmp/timemark.ek` are written to a list for later examination. This technique should work on almost every *NIX-based system. If you are using some kind of GUI or window manager like `gnome`, make sure to create the marker file just before applying the patch; otherwise, all files created by the GUI (temporary or others) will be recorded in the list as well.

You should re-run the installation a couple of times if necessary and try to gradually achieve the desired system state suitable for the intended use. The easiest approach is to install a lean system, just the bare essentials, and then check which features are missing. Continue with the installation of features until the requirements are met. This approach allows you to see how the system reacts when you modify it; sometimes features do not work as easily as advertised. Remember to add the patches for the newly installed features and functionality. If you should encounter errors while adding features and functionality or while patching them, check the manufacturer's Web site for newer *builds* of the operating system. Sun, for example, periodically releases new builds of Solaris as ISO images. These files are supposed to be written to CDs or DVDs, thus supplying you with the latest installation media. If all problems are solved, or there were no problems at all, you need to write all necessary patches to whatever medium is to be used during the installation. Make sure to be aware of the ramifications regarding patches being dependent on other patches or installed packages; some patches cannot be installed before certain other patches or packages are installed first.

Now you should find it possible, armed with the gained knowledge, to estimate the length of the installation process. If you need to install application software, then you need to include it in the tests, applying the same methods used for the installation tests of the operating system. Now add the estimates; the calculated time needed to do a full install, including all applications, gives you a solid estimate of how much time you would have to spend to do it again in case of a hardware failure or other problems.

5.1.2 Partitioning the Disks

Every *NIX system uses one or more hard disks to store the operating system and the created data. Partitioning these hard disks and the distribution of the parts the operating system consists of could have a significant influence on the performance and stability of a system.

The root partition, `/`, is the base of the whole filesystem; everything else, all other connected hard disks and the respective partitions, is accessible from here. The root partition should contain all necessary programs and configuration files to start the system. Software required to repair a malfunctioning or defective system should be stored here as well.

Recommendations concerning which parts of the system should not be located on the root partition but stored somewhere else—either on a different partition or another hard disk—are usually part of the installation process. Usually they differ between the various *NIX brands.

A well-known candidate is the `swap` partition. It is used to store parts of the main memory not used at the moment or paged out to make room for memory-intensive operations. *NIX does not page the main memory without a reason, as opposed to other operating systems that start swapping if some part of the main memory is not used for a certain period of time; paging takes place as soon as the need arises. Locating the `swap` partition on another hard disk usually influences the speed of the system significantly in a positive way. The system will not slow down to a crawl if the `swap` partition is not on a dedicated disk or on a disk used for other purposes, but if it is possible to put it on a separate disk, you should do so.

The filesystem `/var` is the next candidate. It is used to store all variable files like logs, mails, temporary files for printing, and the like. This directory should always be separated from the root partition. If a program should run wild and fill up the `/var` directory, then the system could be in trouble. If the same should happen on the partition containing the root file system, then you are in real trouble. However, the amount of hard disk space available to you on newer systems, seems to invalidate this argument, even a fast system should have problems filling up that much space. On the other hand, one big partition containing all of the system is harder to maintain and to monitor. So why take chances? A dedicated `/var` partition will show up in the baselining reports much faster, and its status is easier to spot (see Chapter 3 for more on baselining).

More often than not the `/usr` and `/usr/bin` directories are stored on separate partitions as well. The advantage of this approach is that it is possible to mount these partitions as read-only. This prevents modifications or worse, manipulations, of the stored programs during runtime. The downside of this approach is that it is not possible to apply patches or upgrades without restarting the machine, thus interrupting the running services. There is no clear-cut answer to this problem; what to do depends heavily on the services provided by the system and the implied security needs. Depending on the exposure to the outside world or how sensitive the data stored on the system is, protecting the programs running on the machine might become a necessity.

Many *NIX flavors separate additional software stored in `/usr/local` or `/opt`, as well. These directories are likely candidates because estimating their growth ratio is not that easy. If you have underestimated their size during the initial setup, you could still get a new hard disk, connect it to the system, and move their contents to it. Afterwards, you just `mount` the new partition into the old directory, and everything is back to normal. Just be aware of the fact that the contents of the directories the new partitions are `mount`ed into are inaccessible until you *unmount* them again.

Which *devices*, partitions on hard disks in this case, are mounted where is defined in the files `/etc/fstab` or `/etc/vfstab`. Some releases of AIX use `/etc/filesystems` as well. You have to check the manpages for the `mount` family of commands where the information is stored on a specific system.

5.1.3 An Installation Checklist

If you are finished with planning the installation and are ready to get started, then you need to create a document similar to the one shown in Listing 5.1.

LISTING 5.1 An Installation Questionnaire

```
!!! This document is supposed to be processed BEFORE the
    installation of the new system !!!

System-Location:_______________

System-Name:___________________
Domain:________________________
MAC-Address:___________________
(should be on the packaging. If there is
more than one NIC, all MAC addresses are
to be recorded.)

System:
Operating System:______________
Release:_______________________
Patches:_______________________
Remarks:_______________________

Networking:
IP-Address:____________________
Netmask:_______________________
Gateway-Router:________________
DNS1:__________________________
DNS2:__________________________
NTP-Server:____________________

Hardware:
CPU:___________________________
Number of CPUs:________________
RAM:___________________________

Hard Disk(s):
Type:__________________________
Count:_________________________
Type:__________________________
Count:_________________________
```

```
Partitions:
/:
Size:__________________________
Device:________________________

swap:
Size:__________________________
Device:________________________

/var:
Size:__________________________
Device:________________________

/opt:
Size:__________________________
Device:________________________

/home:
Size:__________________________
Device:________________________

/data1:
Size:__________________________
Device:________________________

Software:
Name:__________________________
Release:_______________________
Patches:_______________________
Application Path:______________
Remarks:_______________________
...
```

You can modify the example to fit to your needs and fill in the blanks. That way all needed information is available, and it is much harder to forget something. If you get everything you need, you should be able to install the new system.

5.1.4 Copy the Important Commands

To make sure you have access to untainted versions of the vital commands to analyze and repair a system after a break-in, copy the directories `/bin` and `/sbin`—after applying all necessary patches—to a CD. The copy of the directories should be completed *before* the system is connected to the network. An alternate solution would be a copy of the directories on a system not permanently connected to the network. If this system uses another processor architecture and a different kind of *NIX, then the commands would be even more secure. This system or the CD can be used to supply the commands to analysis scripts like `chkrootkit` or `aide`.

5.1.5 `ntp` Setup

The importance of a consistent time and date inside a network was already mentioned. The `ntp` protocol was created to gather this information and distribute it inside the network. If you want to keep things simple, then you just have to configure two machines inside the network either to get the time from a reliable time source inside your network or to connect to a reliable time source outside of your local area network (LAN). An internal time source could be a device that is either a reliable clock by itself, for example, an atomic clock, or a device receiving a time signal via the airwaves. The first solution could be expensive, though. Both solutions require some kind of device driver to access the current time and date. If you don't want to deal with attaching hardware to one of your servers, the other solution would be getting the time from a time server. A list of time servers is available at *http://www.ntp.org*.

COMMON RULES REGARDING THE USE OF TIME SERVERS

There are two types of time servers (actually, there are more, but for the sake of clarity the author assumes that there are two), primary time servers are called stratum 1, and secondary time servers called stratum 2. Some rules of thumb for using a stratum 1 server are as follows:

1. Your internal server serves at least 100 other computers inside your network.
2. Your internal server supplies the information to two to three other internal time servers as a failsafe measure (including routers equipped to function as time server).
3. You provide ntp access as a service for other networks in your geographical region to minimize network traffic.

These rules count on the fact that all users of the Internet try to save bandwidth; it is impossible, though, to enforce these rules. The fact that a significant number of stratum 1 servers that used to be available to the public have switched from unmonitored access to monitored access—this means to get data from the time host, you are supposed to register with the administrators of that system—should give you a clear indication that misusing the services provided by others isn't a good idea, after all. Those servers went to their knees due to the onslaught of requests. Granted, all administrators need precise timing for their networks, but do you really need the accuracy of a stratum 1 server? The lag occurring from a stratum 1 to a stratum 2 server is almost nonexistent; you should have a very good reason to access a stratum 1 time-host. Exceptions from these rules are medical services, nuclear plants, or security services. You are the judge; decide if half a second, sometimes one second—this actually almost never happens, though—is really that crucial for the purpose of your network.

Depending on the size of your network, you should try to fetch the time from two or even better three machines. The protocol uses the differences between the samples to calculate the actual time and thus compensates for the lag. If the gathered information originating from a time server deviates too much from the other samples, that information will be disregarded and checked again during the next cycle. Using three time hosts to get the data into your network provides added security. If two values differ, who is to decide? If there are three values, the majority vote wins. To set the internal clock of a system, you use the command `ntpdate`, as shown in Listing 5.2.

LISTING 5.2 Getting the Time

```
root@unixhost> ntpdate server1 server2 server3
```

This command tries to get the time from three servers and adjusts the internal clock accordingly. If you need to adjust the system time on a recurring basis, then the configuration file `ntp.conf` has to be changed (see Listing 5.3).

LISTING 5.3 `ntp.conf`

```
## servers
## check every 12^12 seconds
## max check 17^17 seconds
server server1 minpoll 12 maxpoll 17
server server2 minpoll 12 maxpoll 17
server server3 minpoll 12 maxpoll 17
```

These settings in `ntp.conf` activate the sampling of time every `12^12` seconds. If you want to share the gathered information with other systems in your network, then the internal clock of the system needs to be in the configuration as well (see Listing 5.4).

LISTING 5.4 `ntp.conf` Internal Server

```
## servers
## check every 12^12 seconds
## max check 17^17 seconds
server server1 minpoll 12 maxpoll 17
server server2 minpoll 12 maxpoll 17
server server3 minpoll 12 maxpoll 17

# configure local clock to act as server
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

The entry `server 127.127.1.0` points at the internal clock of the system. `127.127.1.0` represents the clock of the machine. A `13` instead of `1` would describe a *Leitch CSD 5300 Master Clock Controller*, depending on the type of *NIX, of course. The supplied reference implementations of `xntp` vary between the different *NIX flavors. Denoting your internal clock with a *bad* stratum level like `10` enables your other systems to use it in times of need, for example, if the external or internal time sources are not available. At least the time used inside the network stays consistent that way. The command to degrade the internal clock to stratum level 10 is `fudge 127.127.1.0 stratum 10`.

Now you are able to choose between three ways of providing the time service to your internal systems, or to mix and match as needed. One: you modify the `ntp.conf` on the systems to query your internal machine (see Listing 5.5). Two: configure the `crontab` on the machines to use `ntpdate` on a recurring basis (see Listing 5.6). Three: configure your time server to broadcast the fact that it is indeed a time-host.

LISTING 5.5 `ntp.conf` Internal System

```
server ip.address.of.your.time-host
```

LISTING 5.6 `crontab` entry

```
13 * * * * ntpdate ip.address.of.your.time-host
```

Using broadcasts seems to be the easiest solution, but there is a downside to that approach. The method is not *that* secure. If, for some reason—for example, a configuration error or even malicious intent—another time-host should appear on the internal network, broadcasting different values compared to the regular time servers, a situation that is really hard to debug arises. The file `ntp.conf` on the internal machines needs only one entry, `broadcastclient`. The file on the time server uses the entry `broadcast your.ip.range.255`. As already mentioned, this is not a solution, but a temporary fix, at best. If you want to use the broadcasts, you should deploy the authentication facilities of the protocol. `ntp` version 3 requires you to transfer the generated keys to every system; version 4 is able to distribute the keys from a central location. You just have to keep an eye on the address range(s) you are distributing your keys to.

The keys are stored in either `/etc/ntp.keys` or `/etc/inet/ntp.keys`. You have various encryption methods at your disposal. Type `M` means encryption using `MD5`. Listing 5.7 presents an example.

LISTING 5.7 `ntp.keys`

```
1 M coolpassphrase1
5 M anothercoolphrase
10 M evenbetterphrase
```

You should use `root`-only read/write permissions on this file. The `ntp.conf` on the time server needs the entries shown in Listing 5.8.

LISTING 5.8 `ntp.conf` Using Keys (Server)

```
## servers
## check every 12^12 seconds
## max check 17^17 seconds
server server1 minpoll 12 maxpoll 17
server server2 minpoll 12 maxpoll 17
server server3 minpoll 12 maxpoll 17

# configure local clock to act as server
server 127.127.1.0
fudge 127.127.1.0 stratum 10
# keys
 enable auth
keys /etc/inet/ntp.keys
trustedkey 1 5 10
broadcast your.ip.address.Range.255 key 5
```

The configuration file on the client system needs the entries shown in Listing 5.9.

LISTING 5.9 `ntp.conf` Using Keys (Client)

```
enable auth
keys /etc/inet/ntp.keys
broadcastclient key 5
```

It is possible to use these keys for the queries as well, if you need to make sure your systems know to whom they are talking. You have to change the client entry as shown in Listing 5.10 to make use of the facility.

LISTING 5.10 Query Using Keys

```
enable auth
keys /etc/inet/ntp.keys
trustedkey 10
server address.of.your.internal.time-server key 10
```

You can achieve a lot more by using `ntp`, but the information presented so far should be sufficient to set up a basic network of servers and clients.

5.2 HOW TO READY A SYSTEM FOR PRODUCTION

Installing a system and getting it *ready for production* are two sides of the same coin. Installing a system is step one. Step two, equally important—some say even more important—is to modify the system until all requirements regarding the network it is connected to and the other systems in it are fulfilled. The `$PATH` needs to be configured to match the approved standards in your infrastructure, catering for additional software that is installed on the new machine. `syslog` needs to know what to report and to do so in a way consistent with the rest of your systems. Important files like `/etc/hosts` should contain all the necessary entries to make the new system a well-behaved citizen in the LAN. If all these steps are completed and tested, you should still take a look at Chapter 7 on security before releasing the system in the wild.

Another aspect worthy of reevaluation comprises the running services on the new system. Are there active services *not* needed for the intended use of the system? Does a file server really need a running Web server or an active mail server process? Many services that are not necessary for the purpose of the system can be turned off during the installation or were not installed in the first place. Sometimes one or two of them sneak back on the system; getting rid of unwanted services depends heavily on the manufacturer of the operating system. It never hurts to check again, just to ensure that only needed services are running on a production system.

5.2.1 Setting the `$PATH`

As already mentioned in the Chapter 1 discussion of the `$PATH` variable, the order of the paths stored in this type of variable is significant. You have to make sure that the relevant paths are searched in a specific order to produce deterministic results regarding scripts and programs. You should check the given order first; it consists more often than not of `/bin:/usr/bin`. You have to make sure that the current directory is not part of the `$PATH` variable; its representation is a "`.`". Most systems use a set of files to preconfigure the `$PATH` during the startup of the various shells; these files are usually stored in the `/etc` tree and are named `/etc/default/login`, `/etc/login.default`, or `/etc/.login`. You should check the manpages of the various shells for which files are actually read. Which shell a user uses for the daily work is dictated either by necessity or simply by a matter of taste and preference. Because shells are considered a productivity tool, you should make sure that all available shells are configured to match the policy inside the network. The next part of the `$PATH` variable is usually the programs in `/usr/local/bin`, if the directory exists. Some *NIX brands "hide" programs like `ping` in `/usr/sbin` (usually not part of the `$PATH` of a "normal" user account) because the program is not considered necessary for standard system use. The last part should consist of directories needed to access installed software like a database, applications, monitoring software, etc. It is important to document the relevant paths regarding the executables and the libraries needed by the additional software, thus enabling you to add the necessary components to the default `$PATH` variable. This information is equally important to ensure that all relevant directories are visited during a backup run and to add them to the monitoring scripts (see Chapter 7 for more information on this sort of security). Some applications require the initialization of specific environment variables to enable the execution of the programs. One example would be the Oracle RDBMS. Certain programs that come with the system need variables like `$ORACLE_BASE` and `$ORA_NLS33`; the former stores the base directory of the RDBMS, the latter the encoding of the used character set. Most other application packages require similar variables to be initialized before being able to function properly. You have to make the decision whether variables are set by default or are set by a script that is used to start the program(s). Another option would be the use of skeleton directories and the differentiation of users into certain "classes" or "roles." This last approach will work only for new users, though. The best approach to solve this question is to consult the manuals of the relevant software. The manufacturers usually provide hints which approach should be taken and the inherent drawbacks, should they exist.

There are other environment settings—and files influencing the environment—to consider, but these are heavily dependent on the operating system used. GNU-Linux systems normally make use of the files `/etc/ld.so.conf` or `/etc/ld.conf.so` to enable the runtime linker to point to the current libraries needed by some programs. The settings in these files are usually processed by the command `ldconfig`. Another way to publish the location of the libraries in the system are variables like `$LD_LIBRARY_PATH`. Again, this depends on the brand of *NIX in use. Make sure to consult the documentation of the system to ensure that all programs are able to find the necessary libraries without frustrating the users with error messages. If one of these error messages should occur, you can use the list created in Chapter 1 to identify the locations of the needed files.

The settings of choice should be documented accordingly. *NIX systems tend to run for a very long time, given the inherent stability. You are usually not able to remember the rationale behind a decision made 2 years ago, and the reasoning for that decision might have changed as well, taking into consideration new developments, for example. It is always preferable to have access to the documentation, as opposed to having to guess at why a certain setting was necessary at the time the decision was made to implement a certain feature.

5.2.2 Configuring `syslog.conf`

As already mentioned in Chapter 3, the files created by `syslog` are the first line of defense if you need to get an overview of the status of a given system. The default `/etc/syslog.conf` file contains only a very basic setup regarding the logs to be created, because of the varying needs of the different uses and types of systems in production. There are valid reasons for this default behavior. If a system would be configured to log almost everything that might be of interest, then that system would be able to fill up `/var` in no time at all, if that system were busy enough. The other reason is the fact that not all system administrators are willing to fine-tune the behavior of `syslog`. They believe if the default configuration instructs `syslog` to write too much information into the files, then they could miss important information in the created noise.

What information is considered interesting for the use of a given system can be decided only by the administrator of that system. In case you are not sure what information to collect, enable the settings in question and instruct `syslog` to write the information into separate files. These files should be monitored for about a month, and the contained information should enable the system administrator to decide if the information is important enough to continue with the specific setting. To allow the differentiation of the various messages, `syslog` provides so-called facilities to sort the specific messages into categories (see Table 5.1). These categories use levels to allow access to more granular information. The system is straightforward as one would expect in *NIX.

TABLE 5.1 `syslog` Facilities

Facility	Description
`user`	This facility is used by user processes, more than often processes that do not fit into one of the other categories.
`kern`	This facility is used by the kernel, the "brain" of the operating system.
`mail`	This facility is used by all mail-related processes.
`daemon`	Daemons are all processes running in the background, for example, `powerd` (monitors the power available to the system, provided the system's hardware is able to supply the information) or `vold` (takes care of the volumes available to the system, including CD-ROMs).
`auth`	Everything related to authentication, used, among others, by `login`, `su`, and `ssh` (if configured).
`lpr`	Used by all programs related to print services; this includes fax programs, for example.
`news`	Used only by NNTP (Network News Transfer Protocol).
`uucp`	The UUCP (Unix-to-Unix Copy Protocol) is not in use anymore. It was used to copy files between systems on the precursor of the Internet.
`cron`	Usually used by `at` or `cron`, this facility is not used on every *NIX system; please check the `syslog` or `syslog.conf` manpages.
`local0-7`	The facilities `local0` to `local7` are to be used by the administrator and certain programs. Please note that software like *Legato NetWorker* uses the `local0` facility by default; the behavior can be configured, though.
`mark`	Used internally by `syslog`, but can be used by the administrator as well.
`*`	Not a real facility, but a shortcut for all facilities with the exception of `mark`.

As already mentioned, the facilities make use of so-called levels to differentiate between specific classes regarding the messages. The levels imply certain conditions and are used as follows: if you query a "lower" level, all "higher" levels are automatically included. Table 5.2 contains all available levels; it should make the system a little bit more obvious.

TABLE 5.2 `syslog` Facility Levels

Level	Description
`emerg`	The highest level a message can use; a message with this level is usually broadcasted to all users of the system. These are so-called *panic-conditions.*
`alert`	This level is used for messages that require immediate attention, but are not as severe as `emerg`. A good analogy would be `emerg =` "The house is collapsing." and `alert =` "If no one does something about this, then the house will collapse."
`crit`	Used for critical errors, like a failing hard disk.
`err`	Used for all other error conditions not covered by the previous three levels.
`warning`	Used for warnings of a more general nature.
`notice`	Used for conditions that would not be considered as error, but need attention nevertheless.
`info`	Informational messages.
`debug`	Used to allow a program to communicate while it is debugged.
`none`	A special case, this level can be used to suppress all messages of a given facility, for example, `news.none` instructs `syslog` to ignore all messages sent by the `news` facility.

The table should clear things up a bit. If you would activate `auth.info` in `syslog.conf`, for example, then all messages with `auth.notice`, `auth.warning`, `auth.err`, `auth.crit`, `auth.alert`, and `auth.emerg` would be included automatically. Telling `syslog` what you are interested in is just part of the configuration; you need to tell the program where to store the messages as well (see Table 5.3).

TABLE 5.3 Telling `syslog` Where to Store Messages

Target	Example
A filename instructs `syslog` to store the message in the given file. The file has to exist, though. The message would disappear otherwise.	`/var/adm/messages`
The name of another system denoted by the `@` sign. That system needs to be configured in a way that it is able to receive the sent messages. This approach allows you to	`@companyloghost`

→

Target	Example
collect all messages in one place, but is considered insecure. The messages are transferred in clear text usually.	
A comma-separated list of usernames instructs the program to send the messages to the terminals of the indicated users; the used names have to be present in */etc/passwd.*	`root,erikk,silvied`
The asterisk, "*", means to broadcast the message to all active terminals; should be used only for messages of `emerg` level.	`*`

The syslog mechanism is able to react in various ways to incoming messages and to distribute them. Listing 5.11 shows an example for a syslog.conf on Solaris.

LISTING 5.11 Example `syslog.conf` (Solaris)

```
# syslog.conf
#

# important stuff to the console
*.err;kern.notice;auth.notice                   /dev/sysmsg

# same to the general system log
*.err;kern.debug;daemon.notice;mail.crit        /var/adm/messages

# activate mark facility for system
mark.debug                                      /var/adm/messages

# notify some people if something goes wrong
*.alert;kern.err;daemon.err                     operator
*.alert                                         root

# the sky is falling, let the people know
*.emerg                                         *

# messages from user-space
user.err                                        /dev/sysmsg
user.err                                        /var/adm/messages
user.alert                                      operator
user.alert                                      root
user.emerg                                  *

# log what the authentication system is doing
auth.info                                   /var/log/authlog

# local facilities
# local1 is used for the baseline skripts
local1.info                                 /var/log/baselinelog

# local2 - local7 unused at the moment
# if used, document usage here and in system docs
```

The example in Listing 5.11 contains some special cases that warrant an explanation. The instruction `mark.debug /var/adm/messages` means to write a message with the contents `-- MARK --` to `/var/adm/messages`. The interval (20 minutes) is usually compiled into the `syslog` binary. There are two reasons for that configuration. First, there will be a message every 20 minutes, regardless of system load. Machines with a very low load level usually write only once in a while to the `syslog` otherwise; this setting allows you to see if the `syslog` process is still running. Second, the command `grep MARK /var/adm/messages` should show a continuous list of the marking entries, thus allowing you to determine if `syslog` was up and running all the time. If an intruder tried to hide his presence by disabling `syslog` or modifying the log files, then there is a slight possibility that one of the markings is missing. This is not a real monitor, but every detail counts. The entry `user.emerg` seems to overdo it a little, but if a program starts to behave erratically in user space or even crashes, then all users of the system should know about it and decide if they need the presence of the program. The facility `local1` is used for the messages of the baselining-scripts. Remember: Listing 3.19 used `user.crit`, but only for one reason; this facility usually writes messages with `crit` in the system-wide log file, thus making sure that the command works in the examples. Because you know now how to customize the behavior of `syslog`, you should make use of the `local0-7` facilities.

Here are some hints for the configuration of `syslog`. You should always make sure to create the files to be written to with the command `touch` if they do not exist; `syslog` does not create files. Test changes to `syslog.conf` to avoid losing important messages. The program will re-read the configuration either if it is restarted or if you send the `HUP` signal to the process. You should always test a new configuration by sending a message to the changed or added facility via the `logger` program. If this test should fail, you should try to correct the error and test again. If you cannot find the error, you should enable the old, working configuration and test the non-working configuration on a test system with the same OS release. One hint for finding bugs in this configuration: typos become apparent when you type the configuration again, making a copy to find the bugs usually does not help.

You need to make sure to monitor the growth of the log files, especially after changing the configuration. The `/var` directory has to have enough room to store the data and should have some space left. The files should be copied to another location on the system or to a different system, depending on importance of the files. Another option would be to store the files on tape or to write them to a CD/DVD. You should always use the provided mechanisms to rotate the logs; this means either a variant of `logrotate` (available for almost all *NIX systems) or a supplied software like `logadm` on Solaris. Because `logrotate` is available on Solaris, HP-UX, GNU-Linux, and AIX (see Appendix B), the author offers an example regarding its configuration (see Listing 5.12). The log file to rotate is `/var/log/localmessages`. The configurations are usually stored in `/etc/logrotate.d`, the default setting of `logrotate`.

LISTING 5.12 Configuration for `logrotate` to Rotate `/var/log/localmessages`

```
# rotates the logfiles created by local facility
/var/log/localmessages {
    compress
    daily
    dateext
    rotate 31
    create 640 root root
    postrotate
        kill -HUP `cat /etc/syslog.pid`
    endscript
}
```

The configuration reads as follows: `compress`—the old log file will be compressed; `daily`—the configuration is to be executed on a daily basis; `dateext`—the names of the rotated log files will be appended with the date; `rotate 31`—if a file is rotated 31 times, it will be deleted; `create 640 root root`—a new file with the indicated owner and permissions will be created; and the command(s) between `postrotate` and `endscript` will be executed after the rotation. You have to make sure to find out how the respective syslog daemon can be convinced to re-read the configuration. The command shown should work on all systems that store the file `syslog.pid` in `/etc`. Some variants of the program can be restarted with the command `/etc/init.d/syslog reload` or something similar (the manpage usually contains the details).

If you should follow the previous example and use the `local0` to `local2` facilities to write to separate files named `/var/log/local0log` and so forth, then you could use the example shown in Listing 5.13 to rotate these files.

LISTING 5.13 Rotating More Than One File Using `logrotate`

```
# rotates the logfiles created by local facility
/var/log/local0log /var/log/local1log /var/log/local2log {
    compress
    daily
    dateext
    rotate 31
    create 640 root root
    sharedscripts
    postrotate
        kill -HUP `cat /etc/syslog.pid`
    endscript
}
```

There could be more than three log files in the configuration that should be treated equal. The new instruction `sharedscripts` is important in that respect; it instructs the program to execute the `postrotate` command(s) only once and not three times.

If you have no tools to rotate logs in the OS and no means to download a program to rotate the files, then you could always use the script in Listing 5.14 to do the job.

LISTING 5.14 A Script to Rotate Log Files

```
#!/usr/bin/sh
# $Id: logrotate.sh,v 1.1 2004/08/26 09:17:04 erikk Exp erikk $
################################################
#
#     file:          logrotate.sh
#
#     purpose:       rotates the logfiles
#
#     author:        Erik Keller
#
#     parameters:    none
################################################

# Variables
```

```
LOGPATH=/var/log
LOGFILES="local0log local1log local2log"

if test -d $LOGPATH
then
        cd $LOGPATH
        for LOG in $LOGFILES; do
                test -f $LOG.6 && mv $LOG.6 $LOG.7
                test -f $LOG.5 && mv $LOG.5 $LOG.6
                test -f $LOG.4 && mv $LOG.4 $LOG.5
                test -f $LOG.3 && mv $LOG.3 $LOG.4
                test -f $LOG.2 && mv $LOG.2 $LOG.3
                test -f $LOG.1 && mv $LOG.1 $LOG.2
                test -f $LOG.0 && mv $LOG.0 $LOG.1
                test -f $LOG && mv $LOG $LOG.0
        done
        sleep 30
        kill -HUP `cat /etc/syslog.pid`
fi

# EOF
```

You can use this script as a base and customize it as needed. If the log files should be kept longer, then you could either extend the `for` loop or use the script found in Listing 3.23 as a template.

As already mentioned, you should always monitor the log files. The problem is the time needed to check all files for certain contents. There are commercial and open source offerings to make this task a little easier, but the offerings need to be administered and configured as well, which takes time. A possible solution to this dilemma is using a small script to monitor just the log files deemed important enough. A simple script could, based on the frequency of rotations, create a substantial amount of data by itself. For example, if the log file containing failed logins is rotated only once a week, then there is a distinct possibility that the output of these failed logins will be growing as the week goes by, thus becoming more and more unreadable. Listing 5.15 contains an example for such a monitoring script.

LISTING 5.15 A Script to Monitor the File `authlog`

```
#!/usr/bin/perl
# $Id: chkauth.pl,v 1.1 2004/09/03 09:00:36 erikk Exp erikk $
##################################################
#
#     file:          chkauth.pl
```

```
#
#     purpose:       scan the contents of authlog
#                    for failed attempts
#
#     author:        Erik Keller
#
#     parameters:    none
#
##################################################

# Variables
$WORKDIR = "/var/log";
$LOGFILE = "authlog";

# open the file
open(MY_LOG, "$WORKDIR/$LOGFILE")
        or die "$0: couldn't open log file: $!";
while(<MY_LOG>)
{
        chop;

        if(/failed/)
        {
                print "$_\n";
        }
}


close(MY_LOG);

# EOF
```

The script searches the indicted file for the string `failed` and outputs the found lines. If one wants to use this script, then one should keep in mind that the script will find only entries with the string `failed` and not `Failed` or `FAILED`. The expression has to be customized as well. If you would run this script via `crontab` on a regular basis, you would even receive a mail with the output of the script automatically (`cron` mails the output of a command to the owner of the `crontab`). This approach is a quick working solution, not necessarily elegant, but useable. The drawback of this approach, implying that the monitored script will be rotated once a week, is the fact that the output of the script grows with every failed login attempt, and all failed attempts since the last rotation will be mailed periodically. If the script is executed every hour, then all failed attempts since the last rotation will be sent every hour via mail. If there are not that many failed attempts, this should not be a problem, though. This approach has another advantage: the list of failed logins will be not only on the machine, but also, provided one does not use an IMAP server, in the system administrator's mailbox for safekeeping.

5.2.3 Files Common to All Systems on Your Network

The configuration files in use by a *NIX system come in variants. One variant is completely independent of the underlying operating system, for example, `hosts`; the second variant is used only on specific *NIX brands and might even differ between releases of the operating system. The maintenance of the configuration files can be time-consuming if the used systems deploy more files from the second variant.

The `hosts` file should be identical and current on all systems connected to the same network. It should contain all systems and the aliases used like `loghost`, if the log files should be stored on one specific server. Even if DNS (Domain Name System) is used for name resolution inside the network, if that DNS server should fail or needs to be shut down for some time, then all systems could use the `hosts` file for internal name resolution. The latter implies that the entry in `nsswitch.conf` uses the right order, `hosts: files dns`, and not the other way round. This file should be identical on all systems connected to the network, as well.

The `hosts` file contains the link between an IP address and a name for the system using the IP address. The system is not restricted to one name only, though. Listing 5.16 presents an entry.

LISTING 5.16 An Entry in the `hosts` File

```
192.168.0.100        unixhost.example.domain unixhost mailserver
```

This entry allows the system using this file to draw the following conclusions: `unixhost.example.domain`, `unixhost`, and `mailserver` can be found by using the IP address `192.168.0.100`. If a program or a process tries to find the system named `mailserver`, then the name can be resolved to `192.168.0.100`. If the name cannot be found in the `hosts` file, then the configuration file `nsswitch.conf` is used to find other means to resolve the name, in that case sending a query to the IP address of the DNS server found in `resolv.conf` (see Listing 5.17).

LISTING 5.17 The File `resolv.conf`

```
nameserver 192.168.0.200
domain example.domain
search example.domain
```

The line `domain example.domain` instructs the *resolver* to append `example.domain` to every hostname without a domain part. This means a query for `mailserver` will be sent to the DNS server as a query for `mailserver.example.domain`. The line beginning with `search` has no effect on this system; it is used to determine the domain or subdomain to be searched first. Assume the domain `example.domain` belongs to a company with branches in the U.S., Germany, France, Italy, and the U.K. It would make sense to create subdomains for the different countries, thus resulting in subdomains named `boston.example.domain`, `munich.example.domain`, and so forth. The search entry for a system in the U.S. would look similar to the one shown in Listing 5.18.

LISTING 5.18 `resolv.conf` for Subdomains

```
nameserver 192.168.0.200 192.168.1.200 192.168.1.201
domain boston.example.domain
search boston.example.domain example.domain
```

The search for a hostname without a domain part would use the server with the IP address `192.168.0.200` asking for `hostname.boston.example.domain` first. If the system cannot be found, the query would use `hostname.example.domain`. If this query should also fail, the next DNS server in the list (`192.168.1.200`) would be used and so forth. If the list of possible DNS servers is exhausted and no answer is found, the query would return an error. A system like the one mentioned works best if most of the queries stay in the respective network. Queries regarding systems in other subdomains have to be relayed to the respective subdomain the system belongs to, thus requiring the local name server to "know" about the name servers responsible for the other subdomains. The files `hosts`, `nsswitch.conf`, and `resolv.conf` have to be consistent for all systems capable of processing the queries within a subdomain.

One file that has to be consistent for security reasons is `sshd_config`. The file configures the behavior of the `sshd` daemon, which means the configuration of the allowed encryption algorithms to establish a connection and the protocol to be used, among other things. If all clients use a current release of the software, then the `Protocol` entry should contain only `2` (see Section 7.4.1, "Why Use `ssh`?").

The file `passwd` does not seem to fit to the other files mentioned so far, but only at first sight. If the UIDs and GIDs for the users and the groups used on more than one system are consistent on all systems, then moving a user from one system to another is much easier. Besides that, it is possible to use the UID to include an employee number or other information, provided the UID is large enough. Listing 5.19 shows an example.

LISTING 5.19 Using the UID

```
erikk:x:42120:10::/export/home/users/erikk:/bin/sh
42 = Department Intl. MIS
120 = Employee-ID within the department
```

Because many *NIX flavors limit the variable `$MAXUID` (the theoretical `$MAXUID` on Solaris would be 2147483647), you should be careful when implementing such a scheme. UIDs up to 60000 are usually safe. UIDs less than 100 (Solaris) and less than 500 (Linux), respectively, are used for specific accounts and should not be used.

The use of files like `.rhosts` or `/etc/hosts.equiv`, as convenient as they may be, should be avoided, or at least be permitted only after an extensive risk assessment. These files can be used to authenticate a user on different systems, requiring the user to log in only once. If one of the systems is compromised and all other systems are configured to trust each other using the mentioned files, then the whole network of systems is compromised as well. You should think really hard before using these files. If these files are not used (and they shouldn't be), then it could be necessary to take precautions that these files will not be activated (accidentally or otherwise) in the future. The same holds true for the file `.netrc` in `root`'s `$HOME`. Listing 5.20 shows how to make sure that the mentioned files cannot be used or created without being superuser.

LISTING 5.20 Deactivating `.rhosts`, `hosts.equiv`, and `.netrc`

```
root@unixhost> cd /
root@unixhost> touch .rhosts .netrc /etc/hosts.equiv
root@unixhost> chmod 0 .rhosts .netrc /etc/hosts.equiv
root@unixhost> ls -l .rhosts
----------    1 root     other          0 Sep 23 11:54 .rhosts
root@unixhost> ls -l .netrc
----------    1 root     other          0 Sep 23 11:54 .netrc
root@unixhost> ls -l /etc/hosts.equiv
----------    1 root     other          0 Sep 23 11:54
/etc/hosts.equiv
```

Because of the fact that the files do not have any set permissions, only `root` is able to change these files. The file `.netrc` is in `/` on Solaris systems, the `$HOME` of root. Systems with a separate `$HOME` for `root` should use the command accordingly. GNU-Linux uses `/root` for example. If the system allows the login of users, then the indicated files should be created in their home directories as well to make sure the files will not be created by the users. You just have to ensure that no applications rely on the existence of a (usable) copy of one of these files.

5.2.4 Prepare Directories for Shared Use

If directories should be .used by more than one user on a system, then the way the directories are used plays a significant role in the decision of how to enable the shared use. There a two general types of use:

1. All users have read and write permission in the directory, every user is able to read and copy all files, and every user is able to rename or remove the files created by another user.
2. All users have read and write permission in the directory, every user is able to read and copy all files, but no user is able to rename or remove the files created by another user.

The first usage type can be created easily via the permission system. Listing 5.21 shows an example.

LISTING 5.21 Creating a Directory for a Group

```
root@unixhost> ls -l
total 8
drwxr-xr-x    5 chrisk   adminmgt      512 Aug 23 15:33 chrisk
drwxr-xr-x   11 ekeller  other        1024 Sep 14 14:18 ekeller
drwxr-xr-x    8 erikk    staff         512 Sep 20 09:48 erikk
drwxr-xr-x   10 knapf    staff         512 Aug 23 10:19 knapf
drwxr-xr-x    5 marionb  adminmgt      512 Aug 23 15:35 marionb
drwxr-xr-x    5 markusk  adminmgt      512 Aug 23 15:32 markusk
drwxr-xr-x    3 root     other         512 Aug 20 11:55 nusers
drwxr-xr-x   12 silvied  adminmgt      512 Aug 20 18:57 silvied
root@unixhost> mkdir admintrans
root@unixhost> ls -ld admintrans
drwxr-xr-x    2 root     other         512 Sep 20 10:07 admintrans
root@unixhost> chgrp adminmgt admintrans
root@unixhost> ls -ld admintrans
drwxr-xr-x    2 root     adminmgt      512 Sep 20 10:07 admintrans
root@unixhost> chmod g+w admintrans
root@unixhost> ls -ld admintrans
drwxrwxr-x    2 root     adminmgt      512 Sep 20 10:07 admintrans
root@unixhost> chmod o-r,o-x admintrans
root@unixhost> ls -ld admintrans
drwxrwx---    2 root     adminmgt      512 Sep 20 10:07 admintrans
```

```
root@unixhost> ls -al admintrans
total 3
drwxrwx---    2 root     adminmgt      512 Sep 20 10:10 .
drwxr-xr-x   11 root     other         512 Sep 20 10:07 ..
-rw-rw-r--    1 silvied  adminmgt       89 Sep 20 10:14
monthlytasks.txt
```

Listing 5.21 creates a directory to be shared for the members of the `adminmgt` group. The users `silvied`, `marionb`, and `markusk` have read and write permission in that directory. Every user is able to read, change, and delete the files created by another user; the command `chmod g+w admintrans` gave the rights to all members of the group. All rights of the other users were revoked via `chmod o-r,o-x admintrans`; only members of the group `adminmgt` are able to use the directory and list the files in it.

Is it possible to prevent the accidental removal of a file created by another user? There is already one directory on every *NIX system using that kind of protection, the /tmp directory. Listing 5.22 shows the permissions.

LISTING 5.22 The `/tmp` Directory

```
root@unixhost> ls -ld /tmp
drwxrwxrwt    6 root     sys           381 Sep 20 11:13 /tmp
```

The so-called *sticky bit* is used to prevent one user from removing a file created by another user. The set sticky bit is indicated by the letter `t` in the permissions. If a user has write permission in a directory, then that user is able to remove files, even if that user has no write permission on the file. If the sticky bit is set on the directory, then only the owner and `root` are able to remove a file. Setting the sticky bit is shown in Listing 5 23.

LISTING 5.23 Setting the Sticky Bit

```
root@unixhost> chmod +t admintrans
root@unixhost> ls -l
total 10
drwxrwx--T    2 root     adminmgt      512 Sep 20 13:31 admintrans
drwxr-xr-x    5 chrisk   adminmgt      512 Aug 23 15:33 chrisk
...
```

The reason that there is a capital letter `T` this time, is because on some *NIX variants there are no permissions set for others. The command `chmod +t <directoryname>` works.

Using chmod with 1770

The command `chmod` accepts two different types of parameters to indicate permissions for files or directories.

Symbolic: `chmod g+rwx <filename>` indicates that the group has read, write, and execute permissions.

Octal: `chmod 770 <filename>` indicates that the owner and the group have read, write, and execute permissions. All others have no rights at all.

To get the same result using the symbolic notation, you would have to use the following command: `chmod u=rwx,g=rwx,o= <filename>`. The symbolic notation is usable if you want to remove or add the write permission of the group: `g-w` (remove), `g+w` (add). The symbolic notation is easier to read. The octal notation (base eight) is faster and works as follows:

```
user    group   others   Modus
0400    0040    0004     read (r)
0200    0020    0002     write (w)
0100    0010    0001     execute (x)

0500    0050    0005     read/execute (r-x)
0600    0060    0006     read/write (rw-)
0700    0070    0007     read/write/execute (rwx)
```

The addition of the values `4` = read, `2` = write, and `1` = execute equals `7` (read, write, execute). If you use the binary system to write the same values, everything becomes apparent.

```
     rwx
4   0100  read
2   0010  write
1   0001  execute
```

This means that `0110` binary equals `rw-` or 6. If you want to use the octal notation to manage permissions for a file or a directory, you simply have to add the desired permissions to come up with the right number, `755` for `rwxr-xr-x`.

Although there does not seem be room to set the sticky bit, there actually is. A quick glance at the two previous examples shows that there are four

→

digits available. To set the sticky bit for a directory with read, write, and execute permission for the owner and the group, you could use the following command: `chmod 1770 <directoryname>`.

Be sure to consider the fact that listing the contents of a directory using `ls` requires execute permission on the directory.

The use of the sticky bit improves the security of the `/tmp` directory. This mechanism can be used to your advantage as well.

5.2.5 The "Burn-In"

When you choose to perform the "burn-in" on a new system is debatable at best. The author prefers doing it as soon as the system is installed (including necessary applications), configured, and basically ready to run. What is a "burn-in"? Simply put it's a timespan of at least 2 days (one week is even better) during which the system gets "tortured" with all the load that can be created. If the system doesn't survive this test, then the production use will be delayed, but it is better to find the flaws in the hardware while the system is *not* used for real production work. The downside of this approach is that you have to invest a significant amount of time into the installation and configuration of the system to be tested. Therefore, this approach was modified a couple of years ago. If the system seems to behave strange during install or configuration, then the burn-in for the component in question (say the hard disk) will be performed at once to make sure that the system is in a perfect condition. If there are no errors, the installation or configuration continues as planned. If there are errors, then a call to the manufacturer is in order.

NOTE

The log files created during the burn-in should be checked for errors or strange occurrences. You could use the information in Section 5.2.2 "Configuring `syslog.conf`*," to configure* `syslog` *to handle this task.*

If everything went according to plan, it is time to put some load on the system. Listing 5.24 shows a script to create load on the hard disks.

LISTING 5.24 Script to Create Load on the Hard Disks

```
#!/usr/bin/perl
# $Id: loadtest.pl,v 1.6 2004/05/18 17:04:43 erikk Exp erikk $
##################################################################
#
#     file:          loadtest.pl
```

```
#
#     purpose:       read in a config file and create load
#                       on the target system
#
#     author:         Erik Keller
#
#     parameters:     -d directory to write to
#                     -s size of the first file b|k|m|g
#                     -n number of copies
#
#############################################################

use Getopt::Std;

# Variables
getopt('dsn');

$the_dir = $opt_d;
$the_size = $opt_s;
$the_files = $opt_n;

# use mkfile for Solaris, BSD and OS X
$mkfile = "/usr/sbin/mkfile";

# create the directory
@args = ( "mkdir", "$the_dir");
system(@args) == 0
        or die "$0: couldn't create dir: $!";

# create the file
@args_mkfile = ("$mkfile", "$the_size", "$the_dir/Testing.file");
system(@args_mkfile) == 0
        or die "$0: couldn't create file: $!";

# copy loop
for($the_count = 0; $the_count < $the_files; $the_count += 1)
{
@args_cp = ("cp", "$the_dir/Testing.file",
"$the_dir/descht$the_count.file");

system(@args_cp) == 0
        or die "$0: couldn't copy the file: $!";
}
```

```
# clean up afterwards
@args_rm = ("rm", "-r", "$the_dir");
system(@args_rm) == 0
        or die "$0: couldn't remove the dir: $!";

# EOF
```

One word of warning first: this script is able to seriously destabilize a system. Use it at your own risk. It works on every Solaris, BSD, or OS X system. It uses the `mkfile` command that is part of the indicated systems. The original use of this command is to create swap files in the filesystem. The size unit is specified as `b` = blocks, `k` = kilobyte, `m` = megabyte, or `g` = gigabyte. The script creates a directory by using the path and filename supplied with the `–d` option. The size is supplied with the `–s` option, and the file will be copied `–n` times within the directory.

To use the script on HP-UX, you have to replace `mkfile` with `prealloc`. This command does not know about the options `mkfile` understands, though. It accepts a filename and a number (interpreted as bytes). Another solution would be the use of a `tar` archive of the `/usr/local` tree, for example, and to use this `tar` archive as a file to be copied.

Next, take a look at Listing 5.25.

LISTING 5.25 Using `loadtest.pl`

```
erikk@unixhost> ./loadtest.pl -d tests/ltest1 -s 2m -n 50
```

The command in Listing 5.25 initiates the following events: It creates a file in `/tests/test1` with a size of 2 megabytes. The file would be copied 50 times. Then the script removes the created files and directories again. The script will not check if there is sufficient room for the actions to be performed; if it is out of disk space, it terminates with an error message and without cleaning up. To create a usable load situation, you have to make sure that the disk or array to be tested gets filled close to capacity. The script can be used to test disks and network connections. To test a network connection, you should simply use a directory mounted via NFS. You should never use the script to test a network connection inside a production network, though. Listing 5.26 gives an example.

LISTING 5.26 Using `loadtest.pl` over a Network.

```
erikk@unixhost> mount linuxhost:/data1/tests /mymount/test
erikk@unixhost> ./loadtest.pl -d /mymount/test/ltest1 -s 2m -n 50
```

Naturally commercial products like PureLoad™ or others like it allow much finer grained tests, but one of the goals of this book is to help you use the tools at your disposal in the operating system. Besides, if you write a script, then you learn something about the system and know for sure what happens.

5.3 BOOTING A SYSTEM

The sequence a computer starts with is, independent of the used operating system, more or less the same on all platforms. The system receives power, and a ROM or PROM starts to execute the code stored within. What a program does next depends on the hardware.

If it is an x86-based platform, the BIOS tries to find an executable program on whatever storage media is available. There is no further interaction with the user. Many other hardware platforms come with an intelligent, operating system–type program that allows the user to interact with the hardware without using an operating system. The PA-RISC architecture (sadly threatened with extinction) allows a limited interaction with the found hardware before the operating system starts to run. One step further in the evolution is the *Open Firmware* system (*http://playground.sun.com/1275/*) found in systems by Sun Microsystems, Fujitsu Siemens, and Apple. The system is based on ANS Forth and allows, among other things, the definition of variables to ease the access to disks or network cards. This means an Open Firmware–based system is able to connect to another system on the network and load the operating system from that system instead of using the local harddisk. It is even possible to write programs for Open Firmware; there are hardware updates by Sun that get started from the Open Firmware prompt to ensure that there is no activity on the system during the upgrade. If you use a hardware architecture with similar capabilities, then you should make sure to understand the possibilities of such a system. The Open Firmware–based systems allow you to use the command `probe-scsi-all` to start the self-test of all connected SCSI devices and report the findings—a quick way to test the connected hard disks.

If the computer starts to load the operating system, either automatically or by a command issued at the hardware prompt, then every *NIX system goes through stages, the so-called init states.

5.3.1 The init States

The meaning of the different init states varies between the various *NIX derivates. There is a general distinction between init states used for administrative purposes and the so-called multi-user init states. The system used here as an example is a Solaris system, and the init states there are compared to the differences in HP-UX, IRIX, and GNU-Linux. AIX is mentioned only in the runlevels that are actually used. BSD-based systems like OS X do not use runlevels and are therefore not part of this section.

5.3.1.1 init S

S or s is the same on most *NIX systems. This so-called single-user mode is for maintenance only; all *NIX systems regard one of the multi-user modes as "normal." It is actually not really a maintenance mode at all, but a mode that is used if the file `/etc/inittab` is missing or corrupt. If this happens, booting ends at runlevel S.

If the system goes "down" to runlevel S from one of the multi-user runlevels, mounted volumes stay mounted, but all processes belonging to other runlevels are killed. The only exceptions are remotely mounted filesystems. If they exist, the necessary processes to keep them mounted will continue to run.

5.3.1.2 init 1

Runlevel 1 is the "real" single user or maintenance mode. All local filesystems are mounted; logins, except on the console, are not allowed. This mode is used to, for example, install patches in Solaris. The same holds true for GNU-Linux, HP-UX, and IRIX. This runlevel is reserved on AIX.

5.3.1.3 init 2

Runlevel 2, sometimes described as "local multi-user mode," brings the system up to a usable state. Users are allowed to log into the system, but there is no graphical user interface. The same holds true for HP-UX, IRIX, and GNU-Linux. AIX uses this runlevel as its standard runlevel: all services are activated, and the graphical user interface is started.

5.3.1.4 init 3

Runlevel 3 is the extended multi-user mode. Resources are made available over the network (for example, in NFS shares) and the graphical user interface is started if the system is configured that way. This is the standard runlevel of a Solaris system; the same holds true for IRIX. HP-UX uses this runlevel in the same way with the exception of the graphical user interface, as GNU-Linux. AIX allows the use of this runlevel with a user-defined configuration, similar to `init 4` in Solaris or IRIX; AIX allows the user-defined runlevels from `init 3` to `init 9`.

5.3.1.5 init 4

Runlevel 4 is available, but not used in Solaris. It is available to be used with a user-defined configuration to fulfill special requirements regarding the running processes as an alternative to `init 3`, as in IRIX. HP-UX uses this runlevel as standard runlevel including the graphical user interface.

5.3.1.6 init 5

Runlevel 5 is very dangerous for system administrators switching back and forth between Solaris and GNU-Linux. Switching to `init 5` in Solaris takes the system down and removes power if supported by the hardware. GNU-Linux uses this runlevel as standard runlevel, similar to `init 3` in Solaris and IRIX and `init 4` in HP-UX, respectively. System administrators switching between Solaris and GNU-Linux should put a visible indication about the differences on the system's housing.

5.3.1.7 init 6

Runlevel 6 is used to reboot the system; the same holds true for IRIX and GNU-Linux.

NOTE

There is a saying that you can determine the age of a system administrator by the commands he uses to reboot a system. The rule of thumb is this: the older the administrator the more likely you are to see the commands `sync` *Enter* `sync` *Enter* `reboot` *Enter.*

What does the command `sync` *do? It is used to flush the buffers to the respective disks. Often the reply is like this: "But this is what the system does anyway, doesn't it?" Right, if all goes according to plan.*

The separate command should not be necessary anymore. But the author uses the word "should" deliberately, meaning "most of the time." Using the eight characters and the Enter key twice might make the difference if the data is written to the disk or not. The author's opinion is simple (regardless of age): the two syncs should be used, especially if the system feels strange. Losing data for the sake of ten pressed keys is not an option.

5.3.1.8 init 0

Runlevel 0 is used to halt the machine. If the hardware uses Open Firmware, its prompt will be shown. The same holds true for HP-UX and IRIX. This runlevel is reserved in AIX, and GNU-Linux simply halts the system if run on x86.

5.3.2 init State Overview

Table 5.4 gives an overview of the init states on different systems.

TABLE 5.4 `init` State Overview

State	Solaris	AIX	HP-UX	IRIX	Linux
0	Firmware	Reserved	Firmware	Powerdown	halt
1	Admin	Reserved	Admin	Admin	Admin
2	Multi-User local	Multi-User NFS xdm	Multi-User local	Multi-User local	Multi-User local
3	Multi-User NFS xdm	User defined NFS	Multi-User NFS	Multi-User NFS	Multi-User NFS
4	Alternate Multi-User	User defined	Multi-User NFS xdm	Alternate Multi-User	Not used
5	Shutdown and Power down	User defined	Not used	Firmware	Multi-User NFS xdm
6	Reboot	User defined (up to 9)	Not used	Reboot	Reboot

5.4 SUMMARY

This chapter discussed the installation of a system, including the planning phase and some of the necessary configurations. It continued with a discussion about the difference between an installed system and a system that is ready for production, followed by hints concerning what needs to be done and a description of a "burn-in." The chapter closed with a refresher about the various init states and their use on different *NIX brands.

6 The Test System and the "Secure" System

In This Chapter

- A System for Testing
- Documentation

Before you apply patches and upgrades or deploy new software on the production systems, you need to test. The first part of this chapter discusses what a system to run these tests should look like. Another system that should be part of every system administrator's tool chest is the so-called *secure system*. In this sense, a secure system is not a special system, but rather a concept that helps to deal with certain aspects of the daily workload. (How to secure a set of systems is discussed in Chapter 7) Another topic mentioned often in this book is documentation. *NIX already provides tools to tackle this touchy subject right out of the box.

6.1 A SYSTEM FOR TESTING

It is not easy to find the right arguments for a test system, especially to "sell" it to someone not working in the system administration profession. The following section is an introduction to the ideas behind using a system for testing. Hopefully the case is persuasive enough to arm you with the right information to get it into your budget.

6.1.1 Description of a Test System

The test system should be exactly the same system as the production system, in a perfect world. If you manage to put an exact twin of the production system(s) into the budget to be used as a test bed, congratulations, and please proceed to Section 6.1.2 "A 'Secure' System," because you are done here. All others, bear with the author. The system to run tests on should at least be able to mimic the behavior of the production system. This means it should have the same CPU, not necessarily the same amount of RAM. Testing software, upgrades, and patches intended for a Solaris box on a GNU-Linux system could be interesting, to say the least, but the quality of the findings will obviously lack substance. The test system should be able to run all the software deployed on the production system, not necessarily at the same speed, though. It should be possible to re-create scenarios that exist on the production systems or test scenarios intended to be deployed on the production system. If you want to deploy a customization of the application software bought at SAP, for example, you should test it for errors and stability on a system meant for testing only first. This is a prerequisite to deploy any customization on the system. In other words, the procedure is a requirement *before* deployment of the system in your company and still holds true if you need to apply further customizations *while* you are using the system for production. It is that simple: you have to have a test system. If you deploy first on the production system and test later, good luck calling support.

This approach to testing is valid in all aspects regarding your systems, regardless of the software or operating system used. But here comes the upside common to all systems intended for testing only: they should be able to mimic the real system in a deterministic manner. Nothing else is required. The findings and results of the tests should mirror similar tests executed on the real system.

The test system should not be used to run production services. Testing them, even public testing inside the company, is okay, but as soon as the service becomes *mission critical*, it has to be moved to a real server. Make sure there are no problems whatsoever. If the test system goes down in the most horrible way you can imagine, that is what it is there for. Its sole purpose is to find errors and problems without disrupting the flow of work inside the company. You should create a good documentation about the findings and should try to solve the issues that surface during the tests. This approach enables all employees concerned with the project to deal with errors before the system goes into production. The test system should be reinstalled before every new test. Many problems during the deployment of something new stem from the fact that the system used for testing is not an exact duplicate of the production system, because of a newer release of some component or just a feature not to be found on the production systems.

NOTE

The system administrator's workstation is not a test system. Regardless of how impressive the box on your, or your neighbor's, desk may be, resist the temptation to use it for testing. Chances are that there is software on this machine that isn't found on your production systems. Situations like the following happen all the time: an application produced an impressive crash, every time a certain command was issued. Doing the same on the test system produced no error; everything worked as expected. The development department worked overtime for a couple of days. No problems were found; everything worked as expected. After reinstalling the test system, development was able to reproduce the crash, too. There was a simple explanation: one of the developers installed the development suite on the test system to speed up the compilation of the software. The development environment contained a couple of libraries not found on the usual customer system. It was a simple, though common, case of oversight; the application had access to libraries built for a special purpose, ones not needed on any customer's machine. Fixing the error took about two hours. If the test system would have been reinstalled in the first place, nothing would have happened. Quality Assurance (QA) would have spotted the error right away; the discovery would have caused a delay of two hours, instead of days. If you're thinking, "But my system is clean, really!", think again. Do a quick rundown of the software installed on your machine. All of it will be on the production desktops or servers as well, right?

Most *NIX systems are used for a long time due to the inherent longevity of those systems. A certain hardware configuration might no longer be available, but the configuration might still in production use in the company, posing no problems whatsoever. The same goes for variants of the operating system used; they might not run on the newer versions of the hardware. "Never touch a running system" is a saying that originated from the mainframes of old (some of the readers might remember), but still holds true for well-groomed *NIX systems, as well. If you are planning to purchase a new system for production use, try to include one of its smaller brethren in the calculation. The differences between Solaris 8 and Solaris 9, for example, prohibit testing things for one platform on the other. The same sometimes goes even for minor releases of an operating system. There is a reason to change the version number from 2.5 to 2.5.1; the newer version differs from the old one in stability and functionality. Newer does not necessarily translate to more stable, though. Your test system is the platform to find out.

The hardware configuration of the test system heavily depends on the types of tests to be performed on it. If you need to be as flexible as possible, the differences should not be too big. If the production system uses eight network interfaces, the test system should have at least two. The network subsystem would behave differently if there were only one network interface card (NIC). The same holds true for the number of CPUs. There is no need for 256 CPUs like in the production system, but you should have more than one in the test system. *NIX acts differently when using more than one CPU; trying to recreate an error that might be related to the use of more than one CPU on a single CPU system is a futile exercise. If the test system lacks the capability or capacity to run a certain test, you should try to contact the manufacturer of the systems. Chances are there is a lab you could use to run this single test.

Another advantage of the test system is the fact that you are able to run preliminary tests on it before setting up the new production system. Installing a 1024 CPU monster including an array of 100 hard disks simply takes longer than installing the little brother version of the system with four CPUs and four hard disks. Maybe you can even convince your supplier to give you the little brother as a loaner for some time, thus enabling you to see the advantages of using the new type of hardware before buying it.

6.1.2 A "Secure" System

The "secure" system, something every system administrator should have tucked away in a less frequented corner of the data center, should not be confused with the test system discussed in the previous section. The secure system is not supposed to be connected to the network on a permanent basis, nor should it run any services, at all. If you should follow these suggestions, the latter just won't happen anyway.

The secure system is primarily used for the following tasks:

- Download of patches or upgrades
- Part-time inspection of network traffic
- Penetration tests against the test system
- As experience with a system that differs from your daily chores

To illustrate the idea behind the secure system, Table 6.1 shows some possible configurations.

TABLE 6.1 Secure System Possible Configurations

Hardware	Operating System
DEC/Compaq or HP Alpha	Linux, BSD
Apple Macintosh	Linux, BSD
SPARC-Box	Linux, BSD
Motorola 680x0 or 881x0 VME	BSD

It is probably obvious from this table what we're talking about. The secure system uses an older piece of hardware, one either lingering in some dark corner of the data center or that is cheap to get, and involves putting some "alien" operating system ("alien" in the sense of not commonly installed on that platform) on it. OpenBSD (*http://www.openbsd.org*), FreeBSD (*http://www.freebsd.org*), NetBSD (*http://www.netbsd.org*), and many Linux distributions support older hardware. Creating your secure system is about creating a combination of CPU and operating system *not* to be found at the electronics store next door or down the street. This is not to say that these combinations are weird, but the chances are slim that the latest exploit making its rounds in the Internet and used by *script-kiddies* of all ages will be able to deal with one of theses combinations straight from the shelf (figuratively speaking).

You just have to make sure that the operation system of choice is actively maintained at the moment and will be for the foreseeable future. One of the author's favorites, a combination of Amiga hardware and OpenBSD, won't be maintained at the moment, but that might change again, though. As long as the operating system stays current, it should be usable for the mentioned tasks. If you are not sure if you want to deal with a compiler, using one of the BSD variants seems to be the way to go. It might be necessary to compile a version of a program there as well, but the requirements to get the system to do just that are very manageable. It is not about compiling the latest software in early beta status, but rather about having access to tools like `snort`, `ethereal`, or `tcpdump`. If a standards-compliant Web browser is added as well, the system is ready to be used. If this system does not use hardware that is too exotic, upgrading the memory or the hard disks should be financially feasible as well.

You should be able to perform short or even longer network scans with the software that came with the OS. Scanning the network if you suspect an intrusion should always be done using some kind of hardware and software combination not common in the actual network. You should never feel too secure, though.

If the idea of penetration tests against the test system sounds strange to you, please keep in mind that not all security related patches can be applied to every system without breaking installed third-party applications. Hence, you have the need to be able to check for open ports on a system inside the network. If you know which ports are open, you can take precautions against mischief if necessary. Running an `nmap` penetration test against a production system is something you should avoid at all costs. *As long as you are able to mimic the configuration of the production system on your test system, use the test system as target to see what happens.*

Why use the secure system in the first place? Sure, you could run the penetration tests from the workstation against the test system, but the secure system is *not* a permanent part of the network. Thus, its IP address will stand out of the usual traffic, enabling the department responsible for the network security to filter the penetration tests from the log files, therefore not causing an alarm. To filter the IP address from the log files is figurative for not considering it as a potential threat; no IP address should be filtered from log files *under any circumstances*. If an intruder notices that the system administrators run periodic tests against the systems originating from their workstations, he could spoof these IP addresses to cover his tracks. However, if the IP address of the secure system shows up in the log files and you are sure that the system was not connected to the network at that time, you have a clear indication that someone used the IP address to cover the attacks.

Finally, if you are relatively new to *NIX, installing an operating system on that kind of hardware could be difficult, but there are literally hundreds, if not thousands, of walk-throughs and install documents to help with that undertaking. If you tackle this endeavor with an inquiring mind, you might learn a thing or two about the inner workings of *NIX-based systems, but try to make sure you don’t neglect your daily tasks while at it. If you can spare the time, you should have a closer look at the system. Dissecting another operating system that might use different approaches to tackle common tasks could spark ideas of how to solve a certain problem, well, differently. *NIX evolved this way; why shouldn't everyone make use of this technique? But always keep in mind that you are using hardware that might not be replaceable. Make sure you don't lose anything of importance, in case the system’s hard disk fails or the CPU goes up in smoke.

A closing remark about the secure system: if you are a well-versed BSD administrator reading this book to learn something about *NIX, simply turn this section’s message around. Use the information in this book to create the “exotic” solution from your point of view, meaning you could use a *NIX-based system as your “secure” system. Maybe you might learn a thing or two, too.

6.2 DOCUMENTATION

There is no replacement for good documentation, if a system starts to behave erratically at 3:00 A.M. The truth contained in this sentence has hit us many times during the last 18 years while working as system administrators. Problems or strange behavior will always occur while you are busy doing something completely different. Switching into "error fighting mode" is much simpler when you are able to use good documentation as a fallback to bring you up to speed with what happens in the affected part of the system. If you know what was changed when and for what reason, tackling the problem is much easier.

What is *good* documentation? One possible answer: the availability of all relevant information in a structured manner. But how should this structure appear? What information is to be considered relevant? What about rarely needed information, only necessary for certain tasks? How should you be able to search for the needed information? To approach a solution, we will talk about storing information first.

Storing documentation *and* making it readily accessible at the same time is the first hurdle you need to take. Storing the documentation about a system in whatever format for that sake on that system is the most convenient approach. But one should keep in mind that accessing the hardware documentation about the hard drive to fix a corruption on it might get difficult. Almost equally difficult is accessing information in a format that requires having access to a certain kind of software to read it in the first place. All discussions about proprietary data formats aside, if you need to install a word processor on one of the server systems connected to an uninterruptible power supply (UPS) during a blackout, to look up the information desperately needed to get things running again, the documentation can hardly be considered readily accessible. The first conclusion you can draw from these example: documentation should be available in more than one format, readily accessible at all times, and the formats should be as easy to handle as possible. Printing everything on paper comes to mind. Paper is readily accessible, provided the way it is stored allows some kind of structured access, thus relieving the seeker of the information from the task of wading through tons of paper. Changing the stored information could be accomplished by using a pencil or a pen—a fantastic new digital world, beaten hands down by pen and paper. The author may be exaggerating a little, but there are serious implications that motivate painting such a dark picture.

NOTE

Keep your documentation platform-independent. Even if you create the documentation (the documentation you use on a daily basis now) with the help of the most sophisticated program available at the time being and store that information in the program's format, are you sure you will be able to access this information in a couple of year's time? Some of the gray hairs on the author's head stems from trying to convert really old Lotus 1-2-3 files to a format usable with current software, accessing data stored on 8″ floppies at a dusty CP/M system, and successfully converting data written on a tape with a backup software created by a manufacturer who decided to leave the field of computing sciences many years ago. Not that the *author was responsible for this data. He wasn't even with the respective companies at the time the data was created. Then, all of a sudden, the data became interesting again, but no one had bothered to copy the files to the new formats that replaced the old ones from time to time. These scenarios are not that far-fetched, especially not if you are working as a system administrator. In the light of recent developments regarding the licensing of software—you don't buy the software any more; you license the use for a certain time, and that's it—things could get even more complicated. Besides that, the author doesn't like the idea of having to pay someone for the privilege of accessing documents he created—containing his intellectual property—just because the contents became interesting again. Who has the right to decide how much you have to pay to access* **your** *data again? Possibly establishing the price based on the fact how much the information is worth to you? No thank you!*

What are the options to stay as platform independent as possible? Just to be as blunt as permissible: none of the software manufacturers is exempted from this criticism. If you are a Macintosh user and used FrameMaker to write documentation or books during the last couple of years, you need a license to access these documents—and either a Windows machine or a Solaris box to access or modify these documents (Adobe sent a note to the Mac users of FrameMaker stating exactly that—that they had to use another platform to run the current releases of the software). Plain ASCII files will always be accessible; just use any kind of editor to read, modify, or print the files. The quality of presentation of the information is debatable, though. Besides that, searching for *hard disk* in a document containing this string more than a hundred times is cumbersome at best. To sum it up: plain text files are lacking structure. If a plain text file could contain some kind of structure to, for example, differentiate between a headline and the body text or, even more useful, between the name of a command and the name of something else, the format would become an interesting option to store information.

It might be obvious to some readers what could be used to rectify that: XML. Don't shut the book and run just yet. You aren't supposed to start learning XML right now, nor is it a requirement to read this book any further. If you feel inclined to brush up your XML skills, by all means, go ahead; you should probably read on regardless, though. This section is about the necessity to document the scripts, configurations, and hardware details in *some* way—preferably in such a way that searching for information is as easy as possible. This does not necessarily imply that one has to change the way the documentation is created right now. It does not imply that documentation in a non-XML format is bad either. We'll just use the inherent flexibility of *NIX to meddle with things a little, why don't we?

6.2.1 Tools for Documentation

The smallest entity with regard to documentation is a plain text file. As long as this file does not contain too much information, applying a simple structure will be sufficient. This is not about XML here; headlines, paragraphs, indentation, and enough room between the entries should provide the means to structure the information. Creating such documents, using any text editor capable of saving plain ASCII, is more or less what every system administrator does while editing configuration files, anyway. You should use the program *you* feel comfortable with. You just have to make sure to save the files either in plain ASCII or at least a format readable by as many programs as possible. If there is a well-formed directory tree thrown in the mix—a tree *structure*, to be precise—everything starts falling into place.

In other words you could make use of the operating system's capability to create hierarchies by nesting directories to create the structure for storing the documents. If you want to create a structure to represent the array of servers, for example, you can use the following approach. The top-level directory is named after the network or branch of network that houses the servers, the next level contains directories named after the servers and directories named after procedures to be followed, schemes that need documenting, etc. Listing 6.1 shows a possible structure.

LISTING 6.1 Directory Structure to Document Servers

```
servers
    policies
       security
            mail.txt
            firewall.txt
            access.txt
            ...
    common_files
         hosts
         resolv.conf
         nsswitch.conf
         ...
    mailserver
         hardware_description.txt
         IP_config.txt
         applied_patches.txt
         ...
    fileserver_org
    fileserver_admin
    ...
```

You should modify the structure based on the existing setup and requirements to be documented. If your naming scheme is consistent, doing things like, for example, searching for a patch can be done the *NIX way, as shown in Listing 6.2.

LISTING 6.2 Searching the Structure for Patch Information

```
erikk@unixhost> find . -name applied_patches.txt -print\
    -exec grep 113483 {} \;
./mailserver/applied_patches.txt
Patch: 113483-02 Obsoletes:  Requires:  Incompatibles:
    Packages: SUNW ypu
./fileserver_org/applied_patches.txt
Patch: 113483-02 Obsoletes:  Requires:  Incompatibles:
    Packages: SUNW ypu
./fileserver_admin/applied_patches.txt
Patch: 113483-02 Obsoletes:  Requires:  Incompatibles:
    Packages: SUNW ypu
```

The example shows how to search for a patch named `113483`. The naming of the patches is dependent on the naming scheme used by the manufacturer of the operating system used, of course. Solaris, for example, provides the means to create a list of all installed packages and patches with the command `showrev`. To create a list for later use, you could write the output to a file and store it in the corresponding folder of the hierarchy like so: `showrev -a > applied_patches.txt`. Using the `-a` option, adds, among other things pertaining to this system, the `hostid` to the beginning of the list. To create a list just containing the patches, you could use the option `-p`. HP-UX provides the command `swlist` and AIX provides the command `instfix` to achieve similar results. Because GNU-Linux does not make use of patches in the sense of the other mentioned systems, the command `rpm -q -a` could be used to create a list of all installed packages and their respective version numbers.

As you can see, you do have ways to create structured documentation without using any kind of special software beyond the tools that already come with your operating system. Most of the systems supply the administrator with some GUI to manage the installed packages and software, but using them if the machine in question is down or malfunctioning could be difficult. The same holds true for the documents stored in the created hierarchy. The documentation should be stored in three places: in a secure location on the machine, on a machine that is used to gather all created documentation, and as printouts in a secure location. The list of the installed packages and patches will probably result in an impressive pile of paper, but if you cannot access the files on a computer for one reason or the other, having access to a pile of paper containing the needed information is better than nothing at all. By the way, the author might sound paranoid here, but tampering with this pile of paper is much more complicated than tampering with the files stored on *any* computer.

Other commands useful for creating documentation are `ifconfig` to display the information regarding the network interfaces, `uname -m` to display the installed processor architecture in detail, `vmstat` for displaying the *swap* information, and `df` to list all mounted file-systems.

Other documents you should store in the directories documenting the systems are log files and the data sampled by the `sar` probes (see Section 1.2.4, "What Is Going On?", for more information). The load-information contained in the latter is especially useful, because it allows the creation of detailed statistics regarding the load situation on the systems. The following command (see Listing 6.3) writes the sampled data into a file for later examination.

LISTING 6.3 Writing a `sar` File

```
erikk@unixhost> sar -o `uname -n`.`date +%y%m%d`.sar
```

This command creates a `sar` file, using the hostname of the system, the date, and the extension `.sar` to name the file. These files can be read at a later date by using `sar`'s `-f` option. If this file is created a couple of minutes before midnight via a `crontab` entry, you can gather these files to the system used to store the documentation. To make use of these files, you need the `sar` command of the machine the data originated from, but if the production server's load situation does not allow using it to create a report, you could always use the test system for that purpose. Important here is the fact that these files provide access to all data sampled by `sar`, not only the information used to baseline the system (see Section 3.1, "What Is Baselining?"). This means you should be able to create reports and graphs spanning months instead of the last 30 days.

6.2.1.1 Why XML?

First things first: nobody forces you to structure your documentation using XML. But try to use the path of the least resistance. One of the things that drew the author to computers was the fact that a machine could take care of the mundane details, giving him time to take care of the more important things. You already know the author's point of view regarding documentation: if you don't have access to documentation or the documentation you have access to does not reflect the situation you are confronted with, you will have to do more work. And "do more work" does not sound appealing.. After creating the necessary documentation using `vi` and `emacs` for years, someone introduced the author to FrameMaker while he was working at a software development company. The native format of the software is called

SGML (Standard Generalized Markup Language), not a file format in the strict sense of the term, but more a kind of meta-language used to describe and structure the contents of a file. One SGML dialect everyone is already familiar with is HTML. HTML is actually a standardized description of a document, displayable by a Web browser or any other program that knows what to make of the description, used to display the contained data in human-readable form. HTML uses a set of defined entities, interpreted by the Web browser. The entity `<body>` signals the Web browser to consider everything that follows as contents of a page until the entity is closed by `</body>`. The information contained inside `<body>` can be further structured by using entities like `<h1>` or whatever else is defined. The prerequisite is that the program needs to know about the defined entities. You can deduce two facts by the information presented so far:

1. Using an abstract way to structure information enables you to display that information in more than one way. For example, there are Web browsers for the seeing impaired that are able to read the information aloud, without the need to bother filtering out formatting information, like the used fonts for example. Data structured this way is displayable on a number of devices, enabling the devices to display the information within the restrictions the device imposes due to its capabilities.
2. The strict division of data, structure, and format enables the processing of the information with a variety of programs. One example would be the creation of a (albeit simple) table of contents for a Web site, using the command `egrep '<h1>' *.html`. The fact that there is an agreement to enclose headlines on level one of the structure within the meta-tag `<h1>` allows this command to work.

We all are in the middle of a transitional phase, right now. HTML will be replaced by XHTML eventually; the latter is simply the XML structure used to describe a Web site, displayable by programs conforming to the standard. (Don't worry; it is really hard to differentiate between HTML and XHTML at the moment. Deploying your HTML skills still gets you a long way regarding XHTML.) Using this kind of abstraction allows you to structure information, but XML does not stop there. If you feel the need to create new entities to tag information relevant to *a task at hand*, you are free to do so. Whether or not undertaking this effort is worth the time spent needs to be evaluated on a case-by-case basis, though. The procedure isn't overly complicated, but the new entity needs to be rooted in a solid context to be useful.

Creating your own XML structure to be used for the documentation to be created requires the existence of a DTD (Document Type Definition), a set of rules describing which entities are required and which nesting scenarios are permitted.

Well, that is not entirely true; actually using a DTD is an option, but trying to structure information without a DTD makes it impossible to validate the XML files later. Thus, you have no way for you to find out if the document adheres to some kind of structure or not. Furthermore, you need to define transformation rules to present the data if you want to avoid straining your eyes while trying to make sense of all the words inside the <> signs. Doing so has its value, though. You exercise complete control over the structure and presentation of your data. The downside is the time needed for development and testing. On the other hand, why not try the *NIX approach and find out if someone else already solved the problem?

A very common approach to standardize documentation within the IT sector is `DocBook`: a DTD and corresponding *style sheets* to present the information using HTML or PDF as format of choice. The `DocBook` DTD actually allows the creation of almost any type of publication, but its main use is in documentation. The *Linux Documentation Project (http://www.tldp.org)*, for example, uses `DocBook` to distribute its publications in *.ps*, *.pdf*, and *.html*. The same holds true for the documentation accompanying the Linux kernel, the GNOME project, the KDE project, Samba, and so forth. Some publishers have already switched to `DocBook` to create their books. How does this affect you as a *NIX system administrator? Quite simply, because a bigger part of the *NIX community has decided to use `DocBook` to cater to their documentation needs, a variety of programs and tools to help you to create and maintain your documentation have come into existence. Most of it is open source software, released under the GPL license. The other advantage of acquiring `DocBook`-related skills is obvious—because it is considered a quasi-standard, your knowledge can be put to good use in the foreseeable future.

6.2.1.1.1 Installation

`DocBook` is not an application to be installed, at least not a single application. It consists of a DTD, a couple of XSLT (Extensible Stylesheet Language Transformations) documents, and two libraries probably already installed on your system. The libraries are named `libxml2` and `libxslt`. If these libraries should not be part of the *NIX distribution in use, you can simply download the version for the operating system from one of the Web sites in Appendix B. If you are using GNU-Linux as operating system, everything needed should be on the distribution CDs. You could simply install the necessary packages. If your brand of *NIX did not come with the necessary files, proceed as follows. The `DocBook` DTD is to be found on the *DocBook.org* Web site (*http://www.docbook.org/xml/index.html*). Download it and extract it to a directory. If you want to make `DocBook` available to all of the users on your system, create a directory like `/usr/share/xml` or `/opt/share/xml`, if neither of them already exists. Change into the directory, create another directory named `docbookxml_<version>`, replacing `<version>` with the current release, and extract the archive into it. The XSLT style sheets to transform the XML files into HTML and

PDF can be found on the *SourceForge®* Web site at *http://sourceforge.net/project/showfiles.php?group_id=21935*. Download the archive `docbook-xsl` and extract it within the `xml` directory. To create PDF files, the *Formatting Objects Processor* by *apache.org* and a Java Runtime Environment (JRE) 1.3 or better is needed as well. The *FOP* is to be found at *http://xml.apache.org/fop/*; download it and extract it in the `xml` directory. The script `fop.sh` needs the environment variable `$JAVA_HOME` to be set; it usually points to `/usr/java` on Solaris and `/usr/lib/java/jre` on GNU-Linux.

Table 6.2 shows the packages needed to process `DocBook` files.

TABLE 6.2 Packages Needed to Process `DocBook` Files

Package	Description	Source
DocBook	DTDs, containing the defined structure	*http://docbook.org/oasis/index.html*
DocBook XSL	Transformation style sheets (.xslt)	*http://sourceforge.net/project/showfiles.php?group_id=21935*
`libxml2`	Necessary libraries and the program `xmllint`, among others, to validate the documents	The library is either part of your *NIX distribution or can be downloaded from one of the Web sites in Appendix B.
`Libxslt`	Necessary libraries and the program `xsltproc`, among others, to actually transform the files	The library, too, is either part of your *NIX distribution or can be downloaded from one of the Web sites in Appendix B.
`Fop`	The *Formatting Objects* Processor, a Java class to transform .fo files to PDF	*http://xml.apache.org/fop/*

6.2.1.1.2 Testing the Installation

To make sure everything works as expected, running a quick test is advisable. `DocBook` knows two kinds of publications, *Book* and *Article*. *Book* is a book with a table of contents, chapters, index, and appendices. The chapters can contain sections, which can be nested, if needed. *Article* is more or less a single chapter with a table of contents and an index, if needed. This description is restricted to the bare necessities you need to be aware of; there is much more.

You should create a directory to hold the files necessary to run the tests, a file named `testarticle.xml`, and a file named `testbook.xml`. The contents of the files are shown in Listing 6.4 and Listing 6.5.

LISTING 6.4 Contents of `testarticle.xml`

```
<?xml version='1.0'?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML
V4.3.0//EN"
"/opt/sfw/share/docbook/docbook_4.3/docbookx.dtd">

<article id="testingonly">
<title>testing DocBook Article</title>

<sect1>
<title>This is section 1</title>
<para>Just a text to display something.</para>
<sect2>
<title>Subsection in first section</title>
<para>Nested Section
<screen>
# ./maketest.sh testarticle.xml
</screen>
Programcode looks like the example above.</para>
</sect2>
</sect1>

<sect1>
<title>Another Section 1</title>
<para>Text of
<indexterm><primary>dummyterm</primary><secondary>dummy in
"Another Section 1"</secondary></indexterm>second section
1.</para>
</sect1>

<index />

</article>
```

The article contains two main sections (`sect1`), a nested section (`sect2`) within the first main section, and an index.

LISTING 6.5 Contents of `testbook.xml`

```
<?xml version='1.0'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2.0//EN"
"/opt/sfw/share/docbook/docbook_4.3/docbookx.dtd">
```

```
<book>
  <title>Testing DocBook Book</title>
  <chapter id="testchapone">
    <title>Chapter One</title>
    <para>Text in chapter one. Just to
display<indexterm><primary>display</primary><secondary>word
display in "Chapter One"</secondary><seealso>dummy
entry</seealso></indexterm> something.</para>

<sect1>
<title>This is section one</title>
<para>This text should show up in section 1 inside chapter
1(see also <xref linkend="testchapone" />).</para>
</sect1>

</chapter>

<appendix id="appa">
<title>An Appendix</title>
<para>Not the one in your
stomach<indexterm><primary>stomach</primary><secondary>part
of the body</secondary><seealso>dumb
jokes</seealso></indexterm>.</para>
</appendix>

<index />

</book>
```

The book consists of a chapter, a section, an appendix, and an index.

Now you need to create another file inside the directory, one called `maketest.sh`, with the contents shown in Listing 6.6.

LISTING 6.6 Contents of `maketest.sh`

```
#!/bin/sh
#######################################################
#
#     file:          maketest.sh
#
#     purpose:       automates creation of HTML and
#                    PDF from DocBook XML files.
#
#     author:        Erik Keller
```

```
#
#     parameters:    filename of XML file without
#                    extension, e.g. testbook
#
#######################################################

# Variables
DBOOK_PATH=/opt/sfw/share/docbook/docbook-xsl-1.66.1
MY_FOP=/opt/sfw/share/docbook/fop-0.20.5/fop.sh

echo Bulding $1.html $1.pdf

# clean up first
rm $1.fo $1.pdf

# HTML
xsltproc \
        --output $1.html \
        $DBOOK_PATH/html/docbook.xsl \
        $1.xml
# PDF
xsltproc \
        --output $1.fo \
        --stringparam fop.extensions 1 \
        --stringparam draft.mode no \
        --stringparam section.autolabel 1 \
        --stringparam admon.graphics 1  \
        --stringparam paper.type A4  \
        $DBOOK_PATH/fo/docbook.xsl \
        $1.xml

$MY_FOP -fo $1.fo -pdf $1.pdf

# EOF
```

You just need to modify the paths to match the respective system. The script contains all commands necessary to transform an XML file adhering to `DocBook` standard into an HTML and a PDF document. The program `xsltproc` is part of the `libxslt` package. It is used to transform the XML file using the XSLT files. Listing 6.7 demonstrates the use of the script.

LISTING 6.7 Creating the Documents by Running `maketest.sh`

```
erikk@unixhost> ./maketest testarticle
Bulding testarticle.html testarticle.pdf
Making portrait pages on A4 paper (210mmx297mm)
[INFO] Using org.apache.xerces.parsers.SAXParser as SAX2 Parser
[INFO] FOP 0.20.5
[INFO] Using org.apache.xerces.parsers.SAXParser as SAX2 Parser
[INFO] building formatting object tree
[INFO] setting up fonts
...
erikk@unixhost> ./maketest testbook
Bulding testbook.html testbook.pdf
Making portrait pages on A4 paper (210mmx297mm)
[INFO] Using org.apache.xerces.parsers.SAXParser as SAX2 Parser
...
```

The script `maketest.sh` creates the `.fo` file containing the necessary information to transform it into PDF format using `fop.sh`. The XML files shown should transform without any error messages. If you should get error messages, please check for typos; XML is case sensitive regarding the *tags*. If you want to create files in `DocBook` format containing your own information to be transformed, you need a way of validating your XML files against the `DocBook` DTD. You could use the program `xmllint` to perform this validation. If you should know how to program in C, you probably have an idea or two what `xmllint` actually does. If not, `lint` is a program to do syntax checks and more on `.c` files. It notifies the programmer about potential errors. `xmllint` performs similar checks on XML files, as shown in Listing 6.8.

LISTING 6.8 Using `xmllint`

```
erikk@unixhost> xmllint --noout --xinclude --postvalid testbook.xml
testbook.xml:27: error: Opening and ending tag mismatch:
secondary line 26 and indexterm
jokes</seealso></indexterm>.</para>
                           ^
testbook.xml:27: error: Opening and ending tag mismatch:
secondary line 25 and para
jokes</seealso></indexterm>.</para>
                                   ^
...
```

`xmllint` found an error in line 26. It complains about the fact that a tag is not properly closed. If you take a look at the next error message, you will notice that the first error found by xmllint ripples through the remainder of the document, making the whole structure invalid. If you encounter an error like this, all following error messages should be disregarded. Take care of the first error. All other errors might be a result of the first one. Using the program this way disregards most of its capabilities; the program is extremely versatile. Its output usually displays the whole XML tree found in the document. You can suppress this default behavior by using the option --noout; you are interested in error messages only. The options --postvalid and --xinclude instruct the program to postpone the validation until all external XML files are loaded and the complete tree is created in RAM.

A valid question would be this, "Why was the validation not part of the maketest.sh script?" The answer is that there is more than one way to achieve the desired results. As usual, on *NIX the choice is yours. What should the script do if an error pops up while running xmllint? The script would have to deal with the error messages and take some predefined path of action. It would have to stop processing most of the time (but not always) because the error indicates the XML result tree is not valid for one reason or the other. There might be no reason to continue. A structural error inside an XML file needs human intervention. So instead of trying to intercept possible error messages, you could try a slightly different approach to solve the problem. Sometimes, in the past of *NIX (a really long time ago, computer-wise), people grew tired of writing scripts to initiate and monitor the build process and the messages returned by the compilers. They created the program make. The program is about the same age as lint; among other things, its main use is to manage the dependencies that require monitoring while compiling a program or a set of programs. Because xsltproc isn't that different from a compiler, it actually comes pretty close to c1—for the programmers among the readers—so why not use it? It comes with the operating system anyway. Contrary to popular belief, understanding make is not that hard and complex. Its use is actually pretty straightforward and logical, if you do not try to achieve too much in one go. Listing 6.9 shows a makefile for the task at hand.

LISTING 6.9 A Makefile for DocBook

```
# $Id: Makefile,v 1.4 2004/05/10 10:26:16 erikk Exp $
# makefile for docbook documents
# used programs should be in the $PATH
# erik

# vars first

DOCUMENT_NAME = testbook
DOC_DIR = pubs
MY_FOP = /opt/sfw/share/docbook/fop-0.20.5/fop.sh
```

```
DBOOK_PATH = /opt/sfw/share/docbook/docbook-xsl-1.66.1
MY_PAGESIZE = A4

# rules

all: valid html pdf

valid:
    xmllint --noout --xinclude --postvalid  $(DOCUMENT_NAME).xml

html:
    xsltproc  \
            --output  $(DOC_DIR)/$(DOCUMENT_NAME).html  \
            $(DBOOK_PATH)/html/docbook.xsl  \
            $(DOCUMENT_NAME).xml
pdf:
    xsltproc  \
            --output  $(DOCUMENT_NAME).fo \
            --stringparam fop.extensions 1  \
            --stringparam admon.graphics 1  \
            --stringparam section.autolabel 1  \
            --stringparam section.label.includes.component.label 1  \
            --stringparam paper.type $(MY_PAGESIZE)  \
            $(DBOOK_PATH)/fo/docbook.xsl  \
            $(DOCUMENT_NAME).xml

    $(MY_FOP)  -fo  $(DOCUMENT_NAME).fo \
            -pdf  $(DOC_DIR)/$(DOCUMENT_NAME).pdf

    rm $(DOCUMENT_NAME).fo
```

The file does not differ that much from `maketest.sh`, does it? The most notable differences are the targets `valid:`, `html:`, `pdf:`, and `all:`. These targets can be utilized to call the commands following them directly. Listing 6.10 demonstrates the use of a target.

LISTING 6.10 Using `make` with a Target

```
erikk@unixhost> make valid
xmllint --noout --xinclude --postvalid  testbook.xml
erikk@unixhost>
```

The command shown causes the following: because no filename is supplied via the `-f` option, `make` tries to find a file aptly named `makefile` in the current directory.

If it is not found, the program tries `Makefile`. Because the file exists, `make` reads its contents and tries to find the target `valid:`. If it is found, all commands following the target are executed. The target `all:` demonstrates how to "program" in `make`. It calls `valid:`, `html:`, and `pdf:` in that order. Should an error occur during the execution of a target, `make` simply stops. Thus, if `valid:` returns an error, `make` stops executing, and the error messages are already displayed. If you should want to create only the HTML documents, you would start `make` with `make html`. The author usually copies this file and modifies it to fit the needs every time a new book or article project starts.

6.2.1.1.3 Tips Regarding `DocBook` *and XML*

The easiest way to create new XML files is to use an editor that knows about XML, which means it is able to support the writer in writing valid XML. This book was written with `emacs`. There is an extension called `psgml` that enables `emacs` to "understand" XML, more or less. `emacs` is able to deal with XML documents without this extension, but installing `psgml` increases `emacs` usability. If you open an XML file, for example, `emacs` loads the corresponding DTD and validates the document. If the document does not contain the necessary information about which DTD to use, you can use the DTD menu command to read the necessary files. `emacs` is available for almost all operating systems and is distributed with a GPL license.

Since you are working on *NIX, another approach obviously presents itself—creating the documents using `vi`. This editor has no idea what an XML file is, though. It is the writer who is in charge to make the right decisions. Possible errors will show up as soon as you check the files with `xmllint`. If you don't like `emacs` and have no intention to create XML files using `vi`, don't despair. The number of XML-aware editors is growing on a daily basis. You have a great variety to choose from, free and commercial. Finding the right editor is not an easy task, though. You should examine the programs that came with the *NIX distribution first; chances are good something included can allow the aspiring XML author to take the first steps. Use the Internet to find out about new programs that might fit your needs. Besides using the usual suspects regarding search engines, take a look at the Web site *xmlsoftware.com (http://www.xmlsoftware.com/editors.html)*; the list is impressive, but not complete.

A roadblock discovered sooner or later by every aspiring XML writer concerns the characters < and >. If you need to use these characters inside an XML file, they have to be written `&lt;` (easy to remember as less than = `<`) and `&gt;` (easy to remember as greater than = `>`). This mechanism, replacing the string enclosed between `&` and `;`, can also be used to define words or descriptive names that might change at a later date. If you want to write the documentation for an imaginary program called *FroBozz*, for example, and you have been told the name will change eventually, you need to define an entity containing the preliminary name and change it as soon as you get the real name to use. The XSLT processor will change all occurrences of the defined entity while outputting or transforming the document. Listing 6.11 demonstrates the use of entities.

LISTING 6.11 Using Entities in XML

```
<?xml version='1.0'?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML
V4.3.0//EN"
"/opt/sfw/share/docbook/docbook_4.3/docbookx.dtd" [

<!-- entities -->
<!ENTITY frobozz "FroBozz">
]>

<article id="sedarticle">
<title>&frobozz; QuickRef</title>
```

Every occurrence of `&frobozz;` will be replaced by “`FroBozz`” (without the quotes). If the final decision regarding the name of the program is reached, you just need to change the string `&frobozz;` points to and do a `make all`, thus re-creating all documents with the proper name.

6.2.1.1.4 Extracting Information

The advantages of using XML regarding publications are obvious, but what else can be done if the data is structured? If you want to extract certain information from an XML file, the information you are interested in needs to be described in a transformation document (XSLT). Because this subject requires a very thorough discussion—a variety of books on the market cover only transformations—just a short example (see Listing 6.12) of how it could be done is offered.

LISTING 6.12 XSLT File to Extract Certain Information

```
erikk@unixhost> cat getcommands.xslt
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" />

<xsl:strip-space elements="*" />

<xsl:template match="/">
<xsl:text>Tagged with command:
</xsl:text>
```

```
<xsl:apply-templates />
</xsl:template>

<xsl:template match="title">
<xsl:text>Title: </xsl:text><xsl:apply-templates /><xsl:text>
</xsl:text>
</xsl:template>

<xsl:template match="para">
<xsl:apply-templates select="command"/>
</xsl:template>

<xsl:template match="command">
<xsl:text>  </xsl:text><xsl:apply-templates /><xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
erikk@unixhost> xsltproc getcommands.xslt sedarticle.xml
Tagged with command:
Title: sed QuickRef
Title: Edit commands
  man sed
Title: Inserting lines
  sed '/regex/G'
  sed '/regex/{x;p;x}'
```

The XSLT file shown, `getcommands.xslt`, is used to display the text enclosed in the `<title>` tags. If the following `<para>` block contains text tagged with `<command>`, it will be displayed, too. The result is an overview of the contents contained in the article. The instruction `strip-space elements` removes all additional whitespace. This means space, Tab, and CR characters. To create some kind of formatting, `xsl:text` is used to insert new line breaks. Because all nodes would be displayed by default, the display of the other contents inside `<para>` is suppressed using the `se-lect` command after `apply-templates`.

This is just a quick and dirty example to give you an idea what is possible; the real boundary here is your imagination. The result shown could be achieved in a number of ways. This was just one of them. If your intent is to dive deeper into XML/XSLT, beyond creating HTML and PDF files, you should be aware that there is a steep learning curve involved. If you want to use XML/XSLT in production, plan accordingly.

6.2.1.1.5 A Closing Remark Concerning `DocBook` *and XML*

If you are under the impression that using `DocBook` is a good idea to ease the chore of writing documentation, you are right. As already mentioned, the subject fills a lot of books, with no end in sight. What was demonstrated here should enable you to start experimenting on your own. There is an extensive FAQ (Frequently Asked Questions) regarding `DocBook` at *http://www.dpawson.co.uk/docbook/index.html*.

6.2.1.2 A Closing Remark Regarding Documentation

How you create your documentation is up to you. Just make sure you do it and try to keep it current. It is a great time-saver to have access to the what, why, and when something was changed or done. Things implemented some time ago are especially easier to grasp if you are able to use your documentation as a fallback.

6.2.2 Bag of Tricks

This section is about hints and tips to help you to create your documentation even faster. The author hopes you will find something interesting in here, but always keep in mind that there is rarely an absolutely "wrong" or "right" way of doing things in *NIX.

The first hint is something you could do right away: you can create a text file in the home directory and take notes of everything that seems worth jotting down. The author calls this `my_bag_of_tricks.txt`. To repeat a saying of the one of the author's teachers that still holds true for everything regarding *NIX: "You don't have to know everything by heart; just know where to look it up." To give you some ideas, the author's directory contains files named `oracle_oneliners.txt`, `perl_oneliners.txt`, `bash_tricks.txt`, `really_stupid_things_to_avoid.txt`, etc.

6.2.2.1 Getting Things into `vi`

While editing a file in `vi`, you sometimes need to load in the text of another document. Doing so in `vi` is really simple. For example, you have a file open in `vi` called `my_bag_of_tricks.txt`, and there is a text file called `epoch_time.txt` that contains a Perl one-liner to convert the current time and date into the seconds since the epoch (epoch = 1/1/1970, the base date every *NIX system uses for calculations regarding time). Listing 6.13 shows how.

LISTING 6.13 Getting Text into `vi` Using `:r`

```
My bag of Tricks
This command returns the current time in seconds since the epoch:
:r epoch_time.txt
```

The `r` command tells `vi` to read in the contents of the named file to the current position of the cursor. Listing 6.14 shows the result.

LISTING 6.14 Contents of the File after Execution of `:r`

```
My bag of Tricks

This command returns the current time in seconds since the epoch:

perl -e 'print time'
```

This function is not restricted to text files only. To read in the output of a command, you could use the following approach shown in Listing 6.15.

LISTING 6.15 Reading the Output of Commands (`stdout`)

```
My bag of Tricks

This command returns the current time in seconds since the epoch:

perl -e 'print time'

Output of commands can be read in with "r !command":
~
~
~
~
~
:r !ls -l
```

After pressing Enter, the screen looks as shown in Listing 6.16.

LISTING 6.16 Result of `r !`

```
My bag of Tricks

This command returns the current time in seconds since the epoch:

perl -e 'print time'

Output of commands can be read in with "r !command":
total 8
-rw-r--r--    1 ekeller  users         22 2004-06-04 11:09
epoch_time.txt
```

```
-rw-r--r--    1 ekeller  users        108 2004-06-04 11:11
my_bag_of_tricks.txt
~
~
3 more lines                                              10,1
All
```

This sequence works with all commands that output text to `stdout`.

6.2.2.2 The `fold` Command

When you are writing documentation with, for example, `vi`, the lines tend to get really long. The length of a line while working is usually no problem, but as soon as the document is finished, those long lines sometimes get in the way and are hard to grasp. Fortunately, more people used to have to deal with that problem, and the result is the command `fold`. `fold` just does one thing; it folds longer lines to a more manageable 80 characters (default), or any other length using the `-w` parameter. To make sure `fold` does not break a word, the option `-s` is used to break a line only at the nearest whitespace character. Listing 6.17 demonstrates the command.

LISTING 6.17 Reformatting a Text File with `fold`

```
root@unixhost:> cat file_with_long_lines.txt
This file contains really long lines, lines that wrap while d
isplayed in a Terminal window. Paragraphs of a documentation
about how long lines can be really annoying and unreadable if
 the cat command is used.
Fortunately there is help, the command fold. Available on alm
ost every *NIX system.

root@unixhost:> fold -s file_with_long_lines.txt
This file contains really long lines, lines that wrap while displayed
in a Terminal window. Paragraphs of a documentation about how long
lines can be really annoying and unreadable if the cat command is
used.
Fortunately there is help, the command fold. Available on almost every
*NIX system.
root@unixhost:>
```

6.2.2.3 The `bc` Calculator

`bc` is not a calculator like `xcalc`, but a calculating language. The language is powerful enough to fill a small book in itself, so the author will concentrate here only on a handy function: converting between the decimal, octal, and hexadecimal systems.

To convert the number 7 to binary, for example, you use the commands shown in Listing 6.18.

LISTING 6.18 Conversion Decimal/Binary Using `bc`

```
root@unixhost:> bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
ibase=10
obase=2
7
111
quit
root@unixhost:>
```

After the interpreter is started, the command `ibase=10` tells it to treat the input, if it is a number, as decimal; this is the default. The second line `obase=2` means convert the output to binary. The third line is the number `7`, and after pressing Enter, it is converted to binary, which equals `111`. The last command, `quit`, exits the interpreter. If this simple conversion is all that is needed, you have a quicker way to archive the same result, as shown in Listing 6.19.

LISTING 6.19 Using `bc` with a Pipe

```
root@unixhost:> echo "obase=2;7" | bc
111
```

`echo` outputs the string `obase=2;7` and *pipes* it into `bc`. The `;` acts as line delimiter, just like in Perl or C.

There are two versions of `bc`, though, the "original" one and an extended GNU version. The latter contains, among other things, the command `print`, which behaves a bit like the same command in Perl. The default version delivered with Solaris, for example, does not know the `print` command, but you can always install the GNU version, if the additional capabilities are needed.

It is even possible to write programs in `bc`'s language. Listing 6.20 shows how to make use of that capability.

LISTING 6.20 `bc` Program `conv_tab.bc`

```
x=0;
print "\n"
print "dec\tbin\thex\n"
while(x < 21){
        obase=10
        print x, ":\t"
        obase=2
        print x, "\t"
        obase=16
        print x, "\n"
        x=x+1
}
quit
```

If you feed this program to `bc` with `bc conv_tab.bc`, you should get the output shown in Listing 6.21.

LISTING 6.21 Output when Executing `conv_tab.bc`

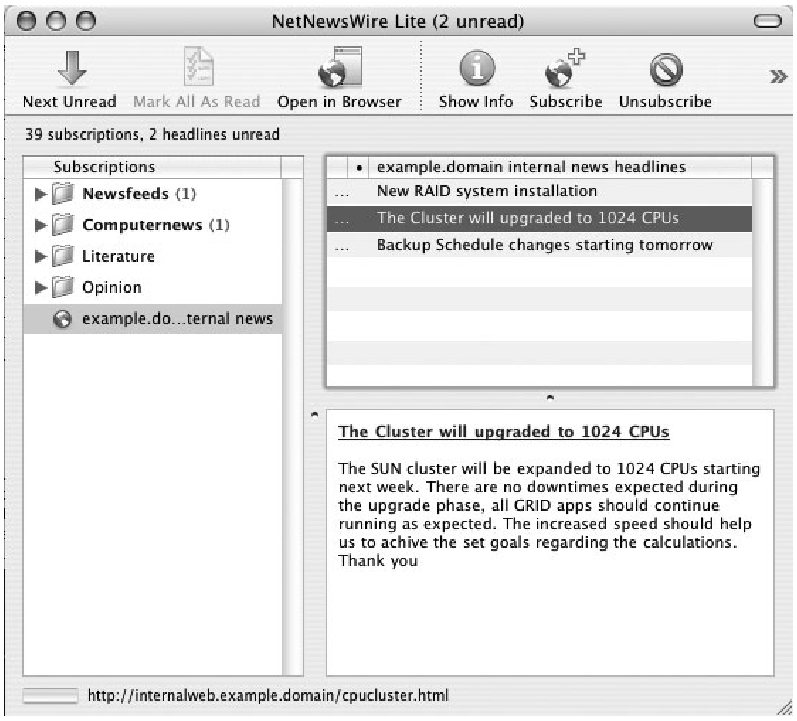
```
dec     bin     hex
 0:     0       0
 1:     1       1
 2:     10      2
 3:     11      3
 4:     100     4
 5:     101     5
 6:     110     6
 7:     111     7
 8:     1000    8
 9:     1001    9
10:     1010    A
11:     1011    B
12:     1100    C
13:     1101    D
14:     1110    E
15:     1111    F
16:     10000   10
17:     10001   11
18:     10010   12
19:     10011   13
20:     10100   14
```

The result is a simple conversion table from decimal to binary and hexadecimal.

6.2.2.4 Using RSS to Document and Communicate

RSS is a topic that is covered in almost all IT-related publications. It is short for *Really Simple Syndication*. It is a protocol to publish news and other information in a fast and easy way to many receivers. The idea behind RSS is as follows: news to be published is wrapped into a defined XML format and is thus readable by a variety of programs, so-called *News Readers*. The simplest action taken by these News Readers is to display the incoming data on the screen. This step requires a minimal amount of processing. It is possible, though, to integrate the incoming information into a publication as well, whether it be into a Web site or on paper. A list of a variety of News Readers for the common operating systems can be found at *http://blogs.law.harvard.edu/tech/directory/5/aggregators*. There are even News Readers to be used on hand-held computers (PDAs) or mobile phones, provided the device is able to connect to a network in one way or the other. On the other hand, the file can be simply copied to the device, but then the news is just as current as the copy on the device, of course.

Figure 6.1 and Figure 6.2 demonstrate how News Readers display incoming information.


NetNewsWire Lite (2 unread)
Next Unread
Mark All As Read
Open in Browser
Show Info
Subscribe
Unsubscribe
39 subscriptions, 2 headlines unread
Subscriptions
Newsfeeds (1)
Computernews (1)
Literature
Opinion
example.do...ternal news
example.domain internal news headlines
New RAID system installation
The Cluster will upgraded to 1024 CPUs
Backup Schedule changes starting tomorrow
http://internalweb.example.domain/cpucluster.html

The Cluster will upgraded to 1024 CPUs

The SUN cluster will be expanded to 1024 CPUs starting next week. There are no downtimes expected during the upgrade phase, all GRID apps should continue running as expected. The increased speed should help us to achive the set goals regarding the calculations. Thank you

FIGURE 6.1 A News Reader showing title and description.

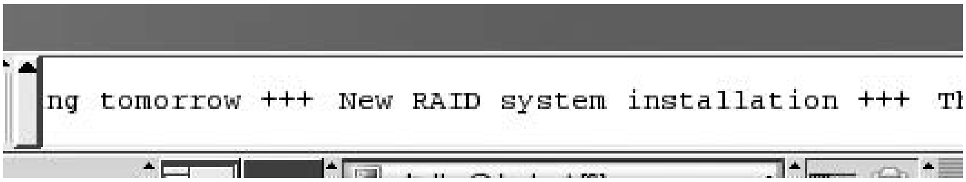

ng tomorrow +++ New RAID system installation +++ Tl

FIGURE 6.2 A News Reader displaying only the title.

The structure used in such a document is quite simple regarding the required XML tags. The examples to follow adhere to *RSS 2.0*. This specification should be readable by almost all programs in circulation. Listing 6.22 shows the basic structure of an RSS channel.

LISTING 6.22 RSS Document Structure without Contents

```
<rss version="2.0">
<channel>
<title></title>
<link></link>
<description></description>
...
</channel>
</rss>
```

The structure is, as already mentioned, really simple, hence the name. Enclosed in the `<title>` tags is the name of the news feed, `<link>` points to a URL containing the Web site delivering the information, and `<description>` contains a short description of the type of news originating from that site. The ellipsis marks the spot where the content is supposed to be. The content is called an item, using an equally simple structure; it is shown in Listing 6.23.

LISTING 6.23 Item Structure RSS 2.0

```
<item>
<title></title>
<description>
</description>
</item>
```

This is the core structure of a message or a news item, using only the required tags, although it diverts from the original specification just a little. The specification allows you to use just `<title>` *or* `<description>`, but some News Readers cannot display items without a title. This is the so-called ticker mode, just displaying the titles of the incoming messages. Therefore, you should use both, thus making sure the intended recipients really get the information.

That’s enough information based on theory only; Listing 6.24 shows how a complete file could look.

LISTING 6.24 Complete RSS File Containing Three News Items

```
<rss version="2.0">
<channel>
<title>example.domain internal news</title>
<link>http://internalweb.example.domain/</link>
<description>example.domain internal newswire</description>
<language>en</language>
<item>
<title>New RAID system installation</title>
<link>http://internalweb.example.domain/newraid.html</link>
<description>
The installation of the new RAID system was successful.
The increased storage will be available shortly.
Thank you
</description>
</item>
<item>
<title>The Cluster will upgraded to 1024 CPUs</title>
<link>http://internalweb.example.domain/cpucluster.html</link>
<description>
The SUN cluster will be expanded to 1024 CPUs starting next week.
There are no downtimes expected during the upgrade phase, all
GRID apps should continue running as expected. The increased speed
should help us to achive the set goals regarding the calculations.
Thank you
</description>
</item>
<item>
<title>Backup Schedule changes starting tomorrow</title>
<link>http://internalweb.example.domain/backsched.html</link>
<description>
The backup schedule will be changed starting tomorrow.
The new times to create the snapshots of Oracle instances
"ACC1" and "ANS42" will be moved to 1:30 AM.
Please move Reportgeneration accordingly.
Thank you
</description>
```

```
</item>
</channel>
</rss>
```

The file shown in Listing 6.24 uses another feature from the specification. The URL in the `<link>` tags points to a Web address containing the elaborate details concerning the news item. Most News Readers open this URL as answer to a double-click on the item. The structure of the file is not that complicated, but maintaining it by hand is error-prone and puts an unnecessary strain on the maintainer of the news. To automate the process somewhat, you could use the script shown in Listing 6.25.

LISTING 6.25 Script for Generating the RSS File

```
#!/usr/bin/sh
# $Id: genrss1.sh,v 1.2 2004/09/13 11:31:46 erikk Exp erikk $
######################################################
#
#     file:          genrss1.sh
#
#     purpose:       generates an rss 2.0 feed from
#                    the files stored in the directory
#                    articles
#
#     author:        Erik Keller
#
#     parameters:    none
#
######################################################

# Variables
WORKDIR=/export/home/users/erikk/work/newstick
ARTICLEDIR=$WORKDIR/articles
RSSFILENAME=$WORKDIR/internalnews.rss

# create the header
echo "<rss version=\"2.0\">" > $RSSFILENAME
echo "<channel>" >> $RSSFILENAME
echo "<title>example.domain internal news</title>" >> $RSSFILENAME
echo "<link>http://internalweb.example.domain/</link>" >> $RSSFILENAME
echo "<description>example.domain internal newswire</description>"
     >> $RSSFILENAME
echo "<language>en</language>" >> $RSSFILENAME
```

```
# get the articles
THENEWS=`ls -t $ARTICLEDIR`

for i in $THENEWS
do
        cat $ARTICLEDIR/$i >> $RSSFILENAME
done

# create the footer
echo "</channel>" >> $RSSFILENAME
echo "</rss>" >> $RSSFILENAME

# EOF
```

This script is no full-blown RSS editor, it just automates the following steps: it creates the header section of the RSS file and fetches a list of files found in the directory `articles`, sorted by the timestamp (`ls -t`). The contents of the found files are appended to the RSS file, newest first; after the last item is appended, the closing tags for `channel` and `rss` are added. A news item in the article directory has the contents shown in Listing 6.26.

LISTING 6.26 News Item in Directory `articles`

```
<item>
<title>The Cluster will be upgraded to 1024 CPUs</title>
<link>http://internalweb.example.domain/cpucluster.html</link>
<description>
The SUN cluster will be expanded to 1024 CPUs starting next week.
There are no downtimes expected during the upgrade phase, all
GRID apps should continue running as expected. The increased speed
should help us to achive the set goals regarding the calculations.
Thank you
</description>
</item>
```

A file like the one shown is much easier to create and maintain, and using the script mentioned makes sure the latest additions will show up first in the feed.

RSS files are distributed via a Web server using the HTTP protocol or as files. Because it is up to the maintainer of the feed how many different files, and thus feeds, are created, using this mechanism allows you to not only keep your users informed about new developments in the network they are connected to in such a way that they do not vanish in the daily storm of e-mails, but also to be a common point of reference for your fellow system administrators. If you would enable the monitoring scripts to write their messages to this format as well, all persons involved would be able to use single points of information to keep current. You just have to make sure to differentiate who will have access to what information; some of the data regarding, for example, the monitoring scripts is definitely not fit for public consumption.

The complete specification is to be found at *http://blogs.law.harvard.edu/tech/rss*.

6.2.2.5 The `scri,pt` Command

If you need to log all commands and responses to these commands into a file for documentation purposes or other uses, the command `script` is there to help. It copies everything that shows up in a terminal window to a file. This approach is especially handy if you want to run tests, want to try to get to know a new program (running in terminal, of course), or simply want to document what you were doing. Listing 6.27 demonstrates the use of the program.

LISTING 6.27 Recording a Terminal Session Using Script

```
root@unixhost> script logadmex1.txt
Script started, file is logadmex1.txt
root@unixhost> crontab -l | grep logadm
10 3 * * * /usr/sbin/logadm
root@unixhost> /usr/sbin/logadm
root@unixhost> ls -l /var/adm/message*
-rw-r--r--    1 root    root   63521 Sep 26 14:58 /var/adm/messages
-rw-r--r--    1 root    root   338591 Sep 21 15:43 /var/adm/messages.0
-rw-r--r--    1 root    root   352923 Sep  8 03:10 /var/adm/messages.1
-rw-r--r--    1 root    root   84875 Aug 28 03:10 /var/adm/messages.2
-rw-r--r--    1 root    root   399605 Aug 18 14:33 /var/adm/messages.3
root@unixhost> Script done, file is logadmex1.txt
root@unixhost> cat logadmex1.txt
Script started on Sun Sep 26 14:58:26 2004
root@unixhost> crontab -l | grep logadm
10 3 * * * /usr/sbin/logadm
root@unixhost> /usr/sbin/logadm
root@unixhost> ls -l /var/adm/message*
-rw-r--r--    1 root    root   63521 Sep 26 14:58 /var/adm/messages
-rw-r--r--    1 root    root   338591 Sep 21 15:43 /var/adm/messages.0
-rw-r--r--    1 root    root   352923 Sep  8 03:10 /var/adm/messages.1
-rw-r--r--    1 root    root   84875 Aug 28 03:10 /var/adm/messages.2
-rw-r--r--    1 root    root   399605 Aug 18 14:33 /var/adm/messages.3
root@unixhost>
script done on Sun Sep 26 14:58:53 2004
```

As shown in Listing 6.27, the recording is started by using the command `script <filename>`. If you are done recording, you use Ctrl-D (end of file) to stop the program and close the file it currently writes to. If you should want to append another session to an already existing file, you could use the `-a` option; the file will be overwritten otherwise.

The resulting protocol of what happened is a good base for a script, by the way.

6.3 SUMMARY

This chapter discussed the necessity of a system to test new software, patches, and upgrades before they are applied to the system(s) in production use. It detailed the requirements regarding the test system and contained some pointer to be used as arguments why a test system should be available to the IT staff. The next topic was the use of the so-called "secure system," a system using a different operating system, used exclusively for maintenance tasks related to the production systems. Again, the "secure system" is no answer to the threats originating from the outside of the company's network, but is rather a simple approach to break a possibly existing monoculture of operating systems in a network. A monoculture enables a cracker attacking from the outside to compromise the systems on the inside of the network, because most systems in a monoculture are susceptible to the same threat. Because the operating system of the "secure system" is different, it *might* not be a potential victim of the same type of attack. The "secure system" is *not* supposed to be part of the network all the time and can thus be used to test the existing test systems or even systems in production.

The chapter continued with a discussion about different approaches to ease the creation of documentation. It closed with a handful of tips concerning how programs that are already part of the *NIX system can be used to streamline the creation of content for the necessary documentation. It discussed the virtues of `DocBook`—a set of DTDs and XSLTs for creating books and articles—which can be used as part of the documentation. Further, it demonstrated the use of the RSS format to convey information to a number of recipients.

This page intentionally left blank

7 Security

In This Chapter

- General Thoughts
- File Security
- System Security
- Network Security
- Security Policies

Security has always been on the agenda of every system administrator, even more so since the Internet changed from being a network of computers primarily inside research institutions to the global grid it is known today. Virtually every system connected to the Internet is able to access every other machine via this connection, regardless of geographical distance. It is a known fact that not every system connected to the Internet has the best interest of the other *netizens* in mind. Sometimes it is necessary, though, to find the right arguments justifying the increased budget to make sure all security precautions are on a current technical level. This chapter contains a discussion about the different approaches to security.

CAUTION

Securing a system or a network requires extensive knowledge and experience. Both can be acquired, but the infrastructure an administrator is responsible for cannot be used increase the knowledge or gain experience. It is not possible to become a *security specialist by reading a book. You should get the support of a security professional if there is any doubt about the state of the systems or the network regarding their security status.*

To increase your knowledge about the subject of security, you should try to obtain as much information as possible, whether by reading books or searching the Internet. The best approach is to concentrate on one specific area and to test and research extensively. Gaining knowledge and experience is (a rewarding) part of the progression to a professional status.

The following is a list of tried and proven security rules:

- No expectations
- No one and nothing is to be trusted
- Nothing is secure
- Security impacts usability
- A healthy paranoia could save the day

This list does not contain a single exaggeration. It contains the ground rules of security.

The hints and techniques in the following chapter cannot guarantee a secure environment. They are meant to be taken into consideration to achieve a certain state of security. A secure environment needs daily maintenance and constant alignment to maintain the status quo.

7.1 GENERAL THOUGHTS

The most impressive security measures are completely useless if the internal customers (the users of the systems) are not aware of the reasoning behind them. Most users consider security as something that “gets in the way,” forcing them to remember new passwords after a certain period of time, and prohibits them from installing the latest add-ons to their Internet browsers, basically taking away the ability to control their systems. The best *security policy* (see Section 7.5, “Security Policy”) will be ineffective if the users cannot understand why it is in place. There are companies whose system administrators have decided to go from securing the machines to damage control only. These system administrators are not willing to cope with the constant flow of complaints relayed to them via upper management by their users any more. The complaints are similar to this: “It is not possible to work without (insert the latest software/add-on/extension to the system).” The impact on the morale of the security team is too much. The crucial oversight here: the security department forgets to keep the upper management informed (see Section

3.3.7, "Management Relations"). Needless to say, if something should happen, the security team is to blame.

The administrators should try to keep their users current about the possible consequences of a security breach. They need to be told about known insecurities of certain operating systems and the speed with which a virus is able to infect a system by simply opening an e-mail. The best approach to raise awareness is periodic training or lectures about known threats. These lectures are not supposed to overwhelm the users with technical specifications, but to train them about basic security measures and how they are able to profit from obeying the security policies. If the schedule of the system administrators does not allow them to conduct the trainings themselves or the administrators lack the necessary skills to do the training, an external consultant should be hired, thus allowing the administrators to learn how to do it first-hand. During a security conference, about a hundred members of upper management, coming from various companies, were shocked to the core as a security consultant demonstrated how to compromise a current Windows system in a matter of minutes and how to use it to send out a huge number of unsolicited e-mails. Mind you, that system was not present at the location the conference took place, but was sitting on a desk at a security firm at the other side of the country. An employee of that security firm discovered the Trojan after opening an e-mail on this machine and took the time to investigate its functionality. The question comes to mind, "How did the virus get into the network in the first place?" The machine in question was a laptop; the employee used this system to check his private mail account as well—using the same software. Again, examining a virus or Trojan should be done only by a *trained* professional.

Another eye-opener for the managers present was the demonstration of a Trojan that survived a reboot and became active again once the operating system was up and running again. The program survived a reboot because of some clever programming; it could be thrown out of the system only by shutting the system down and waiting for at least 30 seconds before powering it on again. Programs like this are not widespread at the moment because their creation requires a very deep knowledge about the underlying hardware, but there is no way of knowing if and when the situation might change for the worse. You need to be careful!

As responsible system administrators, you should monitor Web sites like the "Internet Storm Center" at *http://isc.sans.org* or "SecurityFocus" at *http://www.securityfocus.com* on a daily basis and thus sparing you a lot of trouble. If you are aware of a threat, you can do something about it. Keeping track of all published vulnerabilities, security breaches, and general insecurities regarding the various operating systems can be frustrating, even if the biggest part of them does not concern *NIX. However, there are other operating systems, and they usually *are* part of the same network. If there is a reason to act on a certain information, then you should keep a clear head, inform management and the users (the order depends on the severity of the threat), and take the necessary precautions. If the precautions include blocking a certain port on the network and thus rendering a service used inside the network not operational, then you should do your best to explain the situation, get an approval from upper management, and proceed. The only thing left is to wait for the manufacturer of the affected software or hardware to deliver a patch or upgrade. If your users are not able to work without access to this special port, then you have to try to make a concise risk assessment, present the findings to upper management, and wait for a decision (ideally with the port in question or the affected software blocked). You have to contact the manufacturer to find out about possible solutions at the same time. Should the manufacturer prove unwilling to work with you on that situation—whether it be a simple denial, the promise to deliver a patch somewhere in the not too distant future without any suggestions to hold the situation at bay for the time being, or, even worse, the admittance that the problem exists but "There are so many systems on the Internet, chances are slim that you will be affected by this threat"—then it is time seriously consider switching manufacturers. The future of the company might be at stake, and you have to have a partner you can rely on. A pending security issue is the best test if your partner in this situation—the manufacturer of the software or hardware—is willing to cooperate. If that should not be the case, then you should think about moving on.

One more word about the often cited "The Internet is huge; why should we be the target of an attack?" rationalization. The author's firewall fends off attacks in the 400 per day range at this very moment, on a DSL connection. Think about the possible threat to a published IP address. You always have to ask yourself, "Are you willing to guarantee the security of 50+ computers you cannot control all the time?"

You should also try to work with your users regarding those issues. You should try to explain to them the virtues of a password that has to be changed on a regular basis, make them understand that this system is for *their* protection as well. Their business contacts wouldn't be very pleased to receive unsolicited e-mail originating from their accounts, would they? If the e-mails contain messages of questionable wording, or even worse, contents that might be a punishable offense in certain areas of the country, then all users can understand that security is something to be taken very seriously. The same holds true for the passwords used. Using the name of the spouse, the pet, or the children is not a good password scheme. You need to explain to users that crackers are using so-called *dictionary attacks* to break passwords. This means they try to test the passwords using words found in dictionaries of all kinds and subjects. These dictionaries contain names as well. Some of the dictionary files circulating on the Internet contain more than 400,000 words, including the Latin terms for insects, animals, etc. So even the most clever scheme of medical terms used as passwords can be broken. The creation of a solid password scheme requires more than a dictionary. Also, if someone manages to be in front of a user's physical computer, that someone will likely look under the keyboard if the password is not tacked to the monitor (as many are).

Another subject is so-called *social engineering*. You have to explain to your users that they should not discuss with strangers the operating systems or the revisions in use or give information about the software used. For example, an inexperienced system administrator received a phone call during a current incident. He was asked by the caller, who identified himself as employee of the company the management of the firewalls and routers was outsourced to, about the makers and the current versions of the routers and firewalls. The caller knew the names of the other system administrators, the name of the company the services were outsourced to, and when the inexperienced system administrator was on duty. The investigation of how the caller gained this knowledge is still pending. You have to educate the users that even seemingly irrelevant information like the names of companies providing services and the like is not for public consumption. The more interesting the target is, the more effort will be taken by crackers to gather information about it to crack it.

Most of the mentioned topics can be dealt with by using a good and sound security policy (see Section 7.5, "Security Policy"), but you should never forget that if the users feel understood, then they will be more willing to adhere to that policy. What follows is a rundown of parts of the systems that need to be secured.

7.2 FILE SECURITY

Because (almost) everything in a *NIX system is a file (processes executed in main memory are files in that memory; even the communication between the processes takes place through files), files are the smallest entity that needs to be protected. The inherent problem with files is as follows: a file can be protected by simply turning the write permission off, but that file will be of limited use after doing just that. The next step is a system of granular permissions. *NIX provides a plethora of possibilities in that respect, but even this approach does not allow you to make the system completely secure. As long as there is a way of writing to files, some insecurity remains.

Holding that insecurity at bay is one of the neverending tasks of every system administrator. *NIX provides you with a certain level of security right out of the box; improving on what is already there is your job.

A quick count of the files in `/usr/bin`, `/sbin/`, and `/usr/local/bin` on the system returns 828 executable files. The directories `/usr/local/lib` and `/usr/lib` contain 6326 files in total; these files can be used by an unspecified number of programs and scripts. The system this information was gathered from was a relatively lean Solaris machine installation. You should try to count the numbers on your own system, as shown in Listing 7.1.

LISTING 7.1 Counting Files and Directories in `/usr/lib`

```
# cd /usr/lib
# ls -lR | wc -l
   6007
```

Based on the number shown in Listing 7.1, it is not possible to be aware of all files and directories in that single directory, let alone be aware of the permissions and the size of the files and directories. To help you with this issue, a whole industry creating software to monitor this information was created. There are commercial products like `tripwire`, open source packages like `aide`, and, another approach, scripts you create yourself.

The question remains, "*What* do you have to monitor?" The *NIX answer is, as usual, it depends. Programs and libraries require the monitoring of not only the size and the permissions, but also the date of the last change of the file. The same holds true for configuration files. Log files are a little harder to monitor because they constantly change due to their nature, but nevertheless they contain vital information. If an intruder should manage to alter your log files, then you have almost no way of finding traces of the intrusion. A system that can be considered totally secure is a myth. You should try your very best to make it as hard and complicated as possible for a possible intruder to compromise your systems, though.

7.2.1 File Level Security

To achieve the mentioned goal regarding the security of files, the next two sections talk about a script you can create yourself and an open source solution.

7.2.2 A Simple File Checker

The first approach is to use a variant of the script discussed in Section 1.2.5,"Create a List." The technique to create a list containing all relevant information pertaining to certain directories and files is used in all approaches to monitoring on a file level. Depending on how the list is created, it can be used for a variety of techniques.

7.2.2.1 Getting the md5 Digest of Important Files

The first step, the creation of the md5 sums, can be completed easily. The script in Listing 7.2 creates a list of the files and a list of the same files including a checksum.

LISTING 7.2 Calculating the md5 Sum

```
#!/bin/sh

FILEPATH=/.HO/file_trans
RUNDATE=`date +"%Y_%m_%d"`
THE_FILE=$FILEPATH/sums_$RUNDATE.md5
THE_LIST=$FILEPATH/list_$RUNDATE.lst

for i in etc bin sbin usr/local/bin
do
        find /${i} -follow ! -type d | sort | xargs md5sum >> $THE_FILE
        ls -alLR /${i} >> $THE_LIST
done
```

You have to modify the `$FILEPATH` variable to fit your needs, and the `$THE_FILE` and `$THE_LIST` variables, too, if a different naming scheme is desired. Depending on the OS used, the command `md5sum` might be called `md5` as well; you have to check the manpage to find out. The `md5sum` on Solaris creates the output shown in Listing 7.3.

LISTING 7.3 Excerpt from the Generated List

```
26cc6c38440b37a11679901f3b7cd081  /sbin/hostconfig
d9e5cf9e2b0df382035383a86b752c4a  /sbin/ifconfig
40bcc0b80566e95c1998cf08fe6d9edc  /sbin/ifparse
42415a8a2410b4df88149c229aaef008  /sbin/in.mpathd
df7397902d12bc50adb2d9d350c2f51a  /sbin/init
.....
```

This script also creates a list of the examined directories and their contents. It will be similar to the one shown in Listing 7.4.

LISTING 7.4 List Containing File Information

```
/etc:
total 5376
drwxr-xr-x   54 root     sys         3584 Jun 18 08:50 .
drwxr-xr-x   28 root     root        1024 Jun 18 11:08 ..
-rw-r--r--    1 root     root        2236 Jun 18 08:50 .cpr_config
drwxr-xr-x    3 root     bin          512 Aug 12  2003 .java
-rw-r--r--    1 root     sys          524 Aug 12  2003 .login
-rw-r--r--    1 root     other         18 Aug 12  2003
.sysidconfig.apps
```

```
Drw-r--r--    1 root     root            0 Jun 18 08:50 .syslog_door
-r-xr-xr-x    1 root     sys           493 Aug 12  2003 TIMEZONE
drwxr-xr-x    2 adm      adm           512 Aug 12  2003 acct
-rw-r--r--    1 root     bin          1423 Aug 12  2003 aliases
.....
/etc/acct:
total 6
drwxr-xr-x    2 adm      adm           512 Aug 12  2003 .
drwxr-xr-x   54 root     sys          3584 Jun 18 08:50 ..
-rw-r--r--    1 root     bin           289 Aug 12  2003 holidays

/etc/apache:
total 117
drwxr-xr-x    2 root     bin           512 Apr 19 17:31 .
drwxr-xr-x   54 root     sys          3584 Jun 18 08:50 ..
-rw-r--r--    1 root     bin          2721 Nov  8  2003 README.Solaris
-rw-r--r--    1 root     bin           348 Nov  8  2003 access.conf
.....
```

The contents of these lists can be easily compared to lists created on a previous run of the script via the `diff` command. Listing 7.5 shows an example.

LISTING 7.5 Checking the File List

```
root@unixhost:> diff list_2004_06_17.lst list_2004_06_18.lst
59c59
< -r--r--r--    1 root     sys           132 Apr 13 13:44 hosts
---
> -r--r--r--    1 root     sys           145 Jun 18 13:16 hosts
```

The comparison indicates that the file `hosts` has changed. Listing 7.6 shows the comparison of the md5 sums.

LISTING 7.6 Comparing the md5 Sums

```
root@unixhost:> diff sums_2004_06_17.md5 sums_2004_06_18.md5
48c48
< 020aed6c1b9016be6b8a16a20c20c48d  /etc/hosts
---
> 4fc4ac505b11b02d8bc33d8c3474ad5e  /etc/hosts
```

There is an important reason to use two lists here: a change in the permissions of a file would not be reflected in the md5 sum. The command `chmod o+w /etc/hosts` *does not change the file but the permissions stored in the filesystem.*

The resulting files should not reside on the machine on which they were created; they should be put in a secure location. Writing them, together with the files from other machines if applicable, to a CD-R ensures that no one is able to modify the contents of the lists.

This solution does not measure up to the available commercial or free programs, but is an inexpensive way to monitor important files and directories. You could use the script as a base to create your own custom solutions.

*Should you use your own solution or buy a product? There is no easy answer to this question. It heavily depends on the knowledge and capabilities of the system administrator, not to mention the time constraints. If a system administrator is interested in creating his own solutions, the available time is often a problem. If the time needed (a good rule of thumb is add 50% on top of your calculations) is available, then every *NIX system provides the tools to create the solutions. You should try, though, to find out if another system administrator or programmer is already working on or was already successful in solving a certain issue. If that is the case, then you should inspect the work in progress or existing solution if it solves or at least helps to solve the issue at hand. For system administrators without extensive programming knowledge using this approach comes with the added advantage that the found solution can also be used to gain experience by inspecting the program. "Constant learning" is not just a buzzword for system administrators, but a way of living.*

There is one drawback to this approach regarding security though; you should make a reasonable assessment of your knowledge and capabilities if it comes to the topic of security. If you have any doubt, using an already tested and possibly supported product is the only way to go. Whether this product is created by a consultant or bought from a manufacturer depends on the required functionality. One possible solution could be to work with a consultant to gain the required knowledge; this approach works for the commercial solution and the custom solution equally well, because it furthers the knowledge of the system administrator. This consulting should continue as long as necessary to ensure that the system administrator will be able to deal with the solution in the future. If the budget allows participation in training held by manufacturers or consultants, then you should make sure that training is tailored to be as good as possible to the needs at hand.

7.2.3 Checking Files with `aide`

The program `aide` was created as a replacement for the better known program `tripwire`, as the development of `tripwire` was forked into a commercial version and an open source product. The open source variant is still maintained, but the last update of the packages (at least as of April 2005) took place in March 2001. The goal of `aide` is quite ambitious; the program should become better and more versatile than `tripwire`.

The basic approach behind both programs is the same as the one behind the script in Section 7.2.2, "A Simple File Checker"—creating the means to monitor files and directories regarding changes. The `aide` program is naturally more versatile than the script, but needs, under rare circumstances, to be compiled on the system it should be deployed on. This means the system administrator needs to know how to compile a program and deal with possible error messages. The program is a part of most GNU-Linux distributions and can be downloaded for Solaris at *http://www.sunfreeware.com*.

After the program is installed successfully, you need to edit the file `aide.conf` to customize it for the job at hand. Listing 7.7 shows an example configuration for Solaris.

LISTING 7.7 Example `aide.conf` (Solaris)

```
#AIDE conf

@@define TOPDIR /.HO/aid

@@ifndef TOPDIR
@@define TOPDIR
@@endif

# location of db file
#
database=file:@@{TOPDIR}/aide.db

# location of db file to write to
database_out=file:@@{TOPDIR}/aide.db.new

# Whether to gzip the output to database
# gzip_dbout=no

#verbose=5
verbose=20
```

```
#report_url=stdout
#other possibilities
#report_url=stderr
#NOT IMPLEMENTED report_url=mailto:root@foo.com
#report_url=file:/tmp/some_file.txt
#NOT IMPLEMENTED report_url=syslog:LOG_AUTH
report_url=stdout

   # Here are all the things we can check - these are the default rules
   #
   #p:      permissions
   #i:      inode
   #n:      number of links
   #u:      user
   #g:      group
   #s:      size
   #b:      block count
   #m:      mtime
   #a:      atime
   #c:      ctime
   #S:      check for growing size
   #md5:    md5 checksum
   #sha1:   sha1 checksum
   #rmd160: rmd160 checksum
   #tiger:  tiger checksum
   #R:      p+i+n+u+g+s+m+c+md5
   #L:      p+i+n+u+g
   #E:      Empty group
   #>:      Growing logfile p+u+g+i+n+S

   # You can alse create custom rules - my home made rule definition
goes like this
   #
   MyRule = p+i+n+u+g+s+b+m+c+md5+sha1

   # Next decide what directories/files you want in the database

   /etc p+i+u+g     #check only permissions, inode, user and group for
etc
   /bin MyRule      # apply the custom rule to the files in bin
   /sbin MyRule     # apply the same custom rule to the files in sbin
   /var MyRule
   !/var/log/.*         # ignore the log dir it changes too often
   !/var/spool/.*       # ignore spool dirs as they change too often
```

```
    !/var/adm/utmp$      # ignore the file /var/adm/utmp
    !/var/cron/log/.*    #ignore /var/cron/log
    /opt/sfw MyRule      # Solaris specific
    /usr/openwin/bin MyRule      # Solaris specific
    /usr/dt/bin MyRule   # Solaris specific
    /usr/ccs/bin MyRule  # Solaris specific
    /usr/ucb MyRule      # Solaris specific
#EOF
```

You can use the example in `/usr/local/doc/aide/aide.conf` as a starting point, but the file mentioned should not be used as a configuration file as it is. It is just an example. You could use the example in Listing 7.7 as a guide for what needs to be changed, but should use caution because not all modifications apply to all systems. This means not all modifications make sense on all systems. Some entries in the configuration in Listing 7.7 are shown in bold; these require further explanation. The entry `@@define TOPDIR /.HO/aid` defines the base directory for `aide`. The entries `database=file:@@{TOPDIR}/aide.db` and `database_out=file:@@{TOPDIR}/aide.db.new` search for an existing database and create a new one in the indicated directory, respectively. The entry `MyRule = p+i+n+u+g+s+b+m+c+md5+sha1` is a defined rule, probably a little paranoid. It runs the following tests: `p` checks the permissions; `i` the inode that points to the file in question; `n` the number of links; `u` the owner of the file, `g` the group the respective file belongs; `s` the size of the file; `b` the number of blocks that the file occupies; `m` the date the respective file was last modified; `c` the time the file was last modified; `md5` creates an md5 sum; and `sha1` a sha1 sum. There is no such thing as total control, but this rule should cover almost all bases. The entry `!/var/log/.*` instructs the program to ignore the subdirectory `log` in `/var`.

You simply store the `aide` executable and the modified configuration file `aide.conf` in the location pointed to by `TOPDIR`. Then you need to create the initial database with the `--init` option of the program, as shown in Listing 7.8.

LISTING 7.8 Creating the Initial Database

```
# ls
aide  aide.conf  bu
# ./aide --init -c aide.conf
# ls -l
total 2443
-rwxr-xr-x    1 root     other     1093216 Sep 16  2003 aide
-rw-r--r--    1 root     other        2048 Sep 17  2003 aide.conf
-rw-------    1 root     other     1380792 Aug 31 20:06 aide.db.new
#
```

The -c option points to the configuration file in the current directory. The program created a new database based on the entries in the configuration file called aide.db.new. The information stored in the database is the base for all checks of the indicated directories. You need to change the name of this database to aide.db, as indicated in the configuration file.

If you want to run a check on the directories indicated in the configuration file, you use the option --check as shown in Listing 7.9.

LISTING 7.9 Checking the Directories for Changes

```
# ./aide --check -c aide.conf
AIDE found differences between database and filesystem!!
Start timestamp: 2004-08-31 22:04:24
Summary:
Total number of files=3804,added files=1,removed files=0,changed
files=1

Added files:
added:/opt/sfw/bin/test_whoami
Changed files:
changed:/opt/sfw/bin
Detailed information about changes:

Directory: /opt/sfw/bin
  Mtime    : 2004-08-20 15:35:29              , 2004-08-31 22:03:24
  Ctime    : 2004-08-20 15:35:29              , 2004-08-31 22:03:24
```

The output in Listing 7.9 indicates that a new file named test_whoami was found in /opt/sfw/bin and reports the time and date the modification took place.

If you should apply patches or updates to this system or should have installed new software, then you need to update the database as well by using the option --update. If you use this option to update the database, aide reports all files and directories that have changed since the last run, as shown in Listing 7.10.

LISTING 7.10 Updating the aide Database

```
# ./aide --update -c aide.conf
AIDE found differences between database and filesystem!!
Start timestamp: 2004-08-31 22:12:08
Summary:
Total number of files=3804,added files=1,removed files=0,changed
files=1
```

```
Added files:
added:/opt/sfw/bin/test_whoami
Changed files:
changed:/opt/sfw/bin
Detailed information about changes:

Directory: /opt/sfw/bin
  Mtime    : 2004-08-20 15:35:29              , 2004-08-31 22:03:24
  Ctime    : 2004-08-20 15:35:29              , 2004-08-31 22:03:24

# ./aide --check -c aide.conf
#
```

Another run using the `--check` option gives no results, as there were no changes found.

You need to store the database, the executable, and the configuration file in a secure location. Doing so ensures that no one is able to change the information stored in the database and the configuration file and that no one is able to modify the program itself. One possible option is to store the files on a CD-R. If you store the files on a CD-R, then you should change the database path to point to the CD-ROM drive prior to writing the files to the media. It is a good idea to delete all traces of the program from the system after the files are copied to the secure location. If a break-in should occur, then the intruder would not know that there are monitored files on the system.

You should experiment with the program to get a feeling of what is possible and how to achieve the desired goals before it is used in a production environment. Your final configuration should cover all files and directories that should not change at all. Keep in mind, though, *every* directory not monitored this way is in danger to be used as storage location for unwanted files. If the directory `/var/spool/mail` is not monitored, for example, then it could be used by a possible intruder to store programs that are able to compromise the system. This is not meant to scare you, but it is the job of every system administrator to protect the systems against possible threats, within reason.

7.3 SYSTEM SECURITY

The next unit that needs protection is the physical system itself. This includes the hardware and the software. It is quite amusing to read about security issues regarding *NIX that imply that a possible intruder has physical access to the system to be compromised. If strangers (this includes every person with no really good reason to even be in the vicinity of the computers) are allowed to access the computers physically, then not even the most sophisticated monitoring software will be of any use at all. Physical access to every system that is connected to the network has to be governed by ironclad rules. The common use of laptops poses a problem that cannot be easily solved in that respect.

7.3.1 Where to Store the Install Media

If you allow physical access to the installation media *and* the system, then every system is hackable in a matter of minutes if the intruder knows what needs to be done to compromise the system. Because you need fast access to the installation media in case of a problem, you need to find a logical compromise regarding the location where the media should be stored. One option would be a sturdy closet, not located next to the systems, but some rooms away. The closet needs to be locked and not used for other purposes like storing files, if possible. The same holds true for the location the backups. If the storage location of the backups should be secure (and it definitely should be), then it could be used for the purpose; this is not an optimal solution, but a beginning. Every system administrator should possess a key to the location where the installation media is locked away and carry that key with him all the time. It does not help to fix a problem in the data center (ground-floor or basement) if the system administrator needs to get to his office (14th floor) to get the securely locked away keys to the closet to gain access to the installation media, while trying.

To sum things up: the installation media should never be stored next to the system or, even worse, be stored in an envelope glued to the system. If an intruder should get physical access to the system, it will be compromised in a very short amount of time.

7.3.2 How to Monitor Logins

A mistyped password by a user during login is a quite normal event. If there are 20 mistyped passwords for `root` originating from a terminal somewhere in the building, on the other hand, then you need to investigate what is going on. The same holds true for a failed login for a user if this happens more than three times. That user might have pressed Caps Lock without being aware of it, but this user will definitely call you and complain about a non-working login. If this call should not come in, then you need the documentation to find out where the workstation is located and have a closer look.

If you want to monitor such events, though, then you are confronted with the fact that you need to stare at the log files or a console on a constant basis. The other way you could make sure you do not miss events like the ones described is the use of the `syslog-facility auth.notice`, configured to write all login-related events to a file. Listing 7.11 shows the necessary entry in `syslog.conf`.

LISTING 7.11 Entry in `syslog.conf` to Monitor Logins

```
...
auth.info                    /var/log/authlog
...
```

The file `/var/log/authlog` (and you are free to choose any suitable name) will, depending on the number of logins per day, grow very fast. You need to make sure to rotate the file periodically and store the copies in a secure location if they are needed in a later investigation. After adding this entry into the configuration file, you need to tell `syslog` to re-read its configuration by sending the process the *hang-up* signal. You need to create the file if it does not exist and try to test the functionality by using a non-working login, as shown in Listing 7.12.

LISTING 7.12 Restarting `syslog` and a Non-Working Login

```
root@unixhost> ps -ef | grep [Ss]yslogd
    root   350     1  0 14:56:05 ?        0:00 /usr/sbin/syslogd
root@unixhost> kill -HUP 350
root@unixhost> exit
erikk@unixhost> su - knapf
Password:
su: Sorry
erikk@unixhost> su -
Password:
Commodore 64 38512 Bytes Ready!
# tail -2 /var/log/authlog
Sep  1 11:46:03 unixhost su: [ID 810491 auth.crit] 'su knapf' failed
 for erikk on /dev/pts/2
Sep  1 11:46:16 unixhost su: [ID 366847 auth.notice] 'su root'
succeeded
 for erikk on /dev/pts/2
```

Listing 7.12 demonstrates a failed login by user `erikk` while switching to the user `knapf`. If you would search for the string failed in the log file, then you would be presented with all failed logins that occurred since the last rotation of the log file, provided that log file was not tampered with, another reason to monitor such files. The greeting string on the system was chosen deliberately; it is of no concern to anyone what operating system is used on a machine. The fact that the system administrators know is sufficient.

7.3.3 When and How to Use `chkrootkit`

If you should suspect that something is wrong with a system and you further suspect a break-in, then it is time to analyze the system thoroughly. One program to be used in that respect is called `chkrootkit`.

A *rootkit* is a set of programs that were installed on the system, enabling an intruder to work as `root` on that system without your knowing. A rootkit manipulates settings and programs on the system in a way that prevents detecting the break-in or makes it at least very difficult to find any traces of it. Because you cannot be sure what was modified or replaced, you need to use untainted binaries to detect such a set of programs (see Section 5.1.4, "Copy the Important Commands"). If you need to learn about the inner workings of rootkits and how to detect them, you should consult the information on *http://www.chkrootkit.org*. Using a rootkit on another system could be a punishable offense in most parts of the world.

To make one thing perfectly clear first: not every system showing some kind of strange behavior was taken over by a cracker. You need to check first if the system displays uncommon patterns of network activity. You need another system, known to be clean, to run these checks. If you were not able to find another reason for the strange behavior and still suspect an occupied system, then you need to check the machine.

The first step you need to take is to check the system using `aide` (see Section 7.2.3, "Checking Files with `aide`") or using the lists created in Section 7.2.2, "A Simple File Checker." If there is any reason to believe that something is wrong with the system, then it should be taken off the network immediately. This does not mean that you should shut down the system, but rather disconnect it from the network at once. After that, you need to check the system for the reported modified files, configurations, etc. Which commands, settings, and programs were modified? These modifications allow you to assess the scope of the break-in. If you should not have experience regarding the treatment of a compromised machine, *get professional help.* If the incident damaged the company's data or the system itself, then reporting that damage is governed by strict rules. These rules include a defined procedure concerning how the compromised system is to be secured. Doing computer forensics requires extensive training. If any proof should get lost in the process, insurance and law enforcement cannot do anything for you. Describing the detailed steps required is beyond the scope of this book. If you know the required steps, proceed.

The executable program `chkrootkit` and the required system commands usable on the compromised system should be on a CD-R. You have to mount that CD-R and start the program as shown in Listing 7.13.

LISTING 7.13 Executing `chkrootkit`

```
# ./chkrootkit -p /cdrom/sol_bin
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not infected
Checking `chfn'... not infected
Checking `chsh'... not infected
Checking `cron'... not infected
Checking `date'... not infected
Checking `du'... not infected
Checking `dirname'... not infected
Checking `echo'... not infected
Checking `egrep'... not infected
Checking `env'... not infected
Checking `find'... not infected
Checking `fingerd'... not infected
Checking `gpm'... not found
Checking `grep'... not infected
Checking `hdparm'... not found
Checking `su'... not infected
Checking `ifconfig'... not infected
Checking `inetd'... not infected
Checking `inetdconf'... not infected
Checking `identd'... not found
Checking `init'... not infected
Checking `killall'... not infected
Checking `ldsopreload'... not tested
Checking `login'... not tested
Checking `ls'... not infected
Checking `lsof'... not found
Checking `mail'... not infected
Checking `mingetty'... not found
Checking `netstat'... not infected
Checking `named'... not infected
Checking `passwd'... not tested
Checking `pidof'... not found
Checking `pop2'... not found
Checking `pop3'... not found
Checking `ps'... not infected
Checking `pstree'... not found
Checking `rpcinfo'... not infected
Checking `rlogind'... not infected
Checking `rshd'... not found
Checking `slogin'... not found
Checking `sendmail'... not infected
Checking `sshd'... not infected
Checking `syslogd'... not infected
Checking `tar'... not infected
Checking `tcpd'... not found
Checking `tcpdump'... not infected
Checking `top'... not infected
Checking `telnetd'... not infected
Checking `timed'... not found
Checking `traceroute'... not infected
Checking `vdir'... not infected
Checking `w'... not infected
Checking `write'... not infected
Checking `aliens'... no suspect files
Searching for sniffer's logs, it may take a while... nothing found
Searching for HiDrootkit's default dir... nothing found
Searching for t0rn's default files and dirs... nothing found
Searching for t0rn's v8 defaults... nothing found
Searching for Lion Worm default files and dirs... nothing found
Searching for RSHA's default files and dir... nothing found
Searching for RH-Sharpe's default files... nothing found
Searching for Ambient's rootkit (ark) default files and dirs... nothing
found
Searching for suspicious files and dirs, it may take a while...
Searching for LPD Worm files and dirs... nothing found
Searching for Ramen Worm files and dirs... nothing found
Searching for Maniac files and dirs... nothing found
Searching for RK17 files and dirs... nothing found
Searching for Ducoci rootkit... nothing found
Searching for Adore Worm... nothing found
Searching for ShitC Worm... nothing found
Searching for Omega Worm... nothing found
Searching for Sadmind/IIS Worm... nothing found
Searching for MonKit... nothing found
Searching for Showtee... nothing found
Searching for OpticKit... nothing found
Searching for T.R.K... nothing found
Searching for Mithra... nothing found
Searching for LOC rootkit ... nothing found
Searching for Romanian rootkit ... nothing found
Searching for Suckit rootkit ... nothing found
Searching for Volc rootkit ... nothing found
Searching for Gold2 rootkit ... nothing found
Searching for TC2 Worm default files and dirs... nothing found
Searching for Anonoying rootkit default files and dirs... nothing found
Searching for ZK rootkit default files and dirs... nothing found
Searching for ShKit rootkit default files and dirs... nothing found
Searching for anomalies in shell history files... nothing found
Checking `asp'... not infected
Checking `bindshell'... not infected
Checking `lkm'... not tested
Checking `rexedcs'... not found
Checking `sniffer'... Checking `w55808'... not infected
Checking `wted'... not tested: can't exec ./chkwtmp
Checking `scalper'... not infected
Checking `slapper'... not infected
Checking `z2'... not tested: can't exec ./chklastlog
#
```

The "secure" commands and programs are found on the CD-R in the directory `/sol_bin`. The list of inspected objects on the system is shown in full to give you an example what gets searched and where. The fact that the program found nothing of importance does not necessarily mean that the system is clean, though. You performed the test for a reason, because the system behaved erratically. The logical next step you need to take would be to use the program in expert mode via the `–x` option. Doing so requires a solid understanding of system programming on *NIX, especially the variant to be examined. When run in expert mode, the program outputs a lot of data, including the paths compiled into the system programs, among other things. If you need to filter out this information—this means you are interested in the paths compiled into the programs—then you can use a command like the one shown in Listing 7.14.

LISTING 7.14 Extracting the Paths Compiled into the System Commands

```
# ./chkrootkit -x | egrep '###|^/' | more
###
### Output of: /usr/bin/strings -a /usr/bin/basename
###
/* SVr4.0 1.8
###
### Output of: /usr/bin/ls -l /usr/bin/basename
###
###
### Output of: /usr/bin/strings -a /usr/ucb/biff
###
```

```
/usr/lib/ld.so.1
###
### Output of: /usr/bin/ls -l /usr/ucb/biff
###
###
### Output of: /usr/bin/strings -a chfn
###
###
### Output of: /usr/bin/strings -a chsh
###
###
### Output of: /usr/bin/strings -a /usr/sbin/cron
...
```

This listing filtered the output via `egrep '###|^/'`; therefore, the output is restricted to lines containing either the string `###` or start with `/`. This way the output contains only the examined programs and the paths found in them.

If you just need to examine one program for strings that are compiled into it, then you could use the command aptly called `strings`, as well. Listing 7.15 shows its use.

LISTING 7.15 Using `strings` to Extract the Paths from a Program

```
erikk@unixhost> strings /usr/sbin/cron | egrep ^/
/usr/bin:/bin
/etc/crontab
/usr/sbin/sendmail
/dev/null
/var/cron
/var/run/
/bin/sh
```

Listing 7.15 demonstrates the extraction of all strings contained in the `cron` command. Because the output could be potentially very long and could contain a lot data you are not interested in, you can filter the output to include just lines starting (`^`) with `/`. The `chkrootkit` program uses the same approach.

Some recommend running `chkrootkit` periodically via the `crontab`. This could be a valid approach, as long as you are aware of the fact that a possible intruder could modify the `chkrootkit` program in such a way that it never reports anything or signals everything is fine, respectively. The same holds true for the needed system programs.

7.3.4 Testing a System with `nmap`

A warning first: before using `nmap` you have to make sure that using it is not a punishable offense wherever you are. This is not a joke; it is a serious threat to the security community as a whole. Some are trying to outlaw the use and even the possession of that kind of software. Normally it is no one's business what you do with your systems or the systems you are responsible for, respectively. However, be clear; running `nmap` against a system you do *not* own can result in jail time.

So what is `nmap` used for? The program can be used to map an existing network by scanning it for active IP addresses or to test active IP addresses for open ports, among other things. The two uses mentioned could be combined into one scan, as well. Now, it should be obvious why this program can pose a real threat to the security of a company. On the other hand, `nmap` has received numerous awards and recommendations in the security community for being one of the best programs to monitor the network *you* are responsible for. Many large companies use the program to monitor their networks on a constant basis—in one case even involving a network of over one million IP addresses. The company in that example uses a dedicated machine with a couple of network cards, though.

If you need to check whether a certain system or a range of systems is available, you could use the program as shown in Listing 7.16.

LISTING 7.16 Listing the Available Machines in an IP Segment Using `nmap`

```
root@unixhost> nmap -sP 192.168.0.1-254

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Host  (192.168.0.1) appears to be up.
Host linuxhost (192.168.0.2) appears to be up.
Host osxhost (192.168.0.3) appears to be up.
Host unixhost (192.168.0.5) appears to be up.
Host cvshost (192.168.0.200) appears to be up.
Nmap run completed -- 254 IP addresses (5 hosts up) scanned in 6 sec-
onds
```

The option `-sP` instructs `nmap` to contact every address in the range of `192.168.0.1-254` in different ways. If there should be an answer, then the IP address is appended to the list. If you need to run such a scan, make sure to talk to the people responsible for security first. This kind of scan will show up on most firewalls and IDS (Intrusion Detection System), thus triggering an alarm.

If you need to scan a single system for open ports, you could use the program as shown in Listing 7.17.

LISTING 7.17 Port Scan Using `nmap` on a Single Host

```
root@unixhost> nmap -PO -v -v 192.168.0.3

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming SYN Stealth
scan. Use -sP if you really don't want to portscan (and just
 want to see what hosts are up).
Host osxhost (192.168.0.3) appears to be up ... good.
Initiating SYN Stealth Scan against osxhost (192.168.0.3)
The SYN Stealth Scan took 852 seconds to scan 1601 ports.
Interesting ports on osxhost (192.168.0.3):
(The 1600 ports scanned but not shown below are in state: filtered)
Port       State       Service
123/tcp    closed      ntp

Nmap run completed -- 1 IP address (1 host up) scanned in 852 seconds
```

The option `–P0` instructs `nmap` not to test the availability of the indicated system by sending an ICMP packet, and `–v –v` tells the program to be extremely talkative, which means the program's output contains more detailed information. Because no specific scan was indicated on the command line, the program performed a so-called *SYN* scan. This means that the program sends certain packages to the system, trying to convince it to answer in a certain way. If this certain answer is received, then the tested port is open. The program reported one open port of 1600 tested ports in Listing 7.17. In that case the reason is that the machine uses a firewall that limits packets to port 123 (NTP) to certain IP addresses and blocks everything else.

Another use of the program (most ISPs use this feature) is to determine the operating systems connected to the network with the help of the `–O` option. Listing 7.18 shows an example.

LISTING 7.18 Using `nmap` to Reveal the Operating System Used (1)

```
root@unixhost> nmap -PO -v -v -O 192.168.0.5

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming SYN Stealth
scan. Use -sP if you really don't want to portscan (and just
want to see what hosts are up).
Host unixhost (192.168.0.5) appears to be up ... good.
Initiating SYN Stealth Scan against unixhost (192.168.0.5)
Adding open port 22/tcp
The SYN Stealth Scan took 919 seconds to scan 1601 ports.
```

```
Warning:  OS detection will be MUCH less reliable because we
did not find at least 1 open and 1 closed TCP port
For OSScan assuming that port 22 is open and port 43840 is
closed and neither are firewalled
Interesting ports on unixhost (192.168.0.5):
(The 1600 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
Remote operating system guess: Solaris 9 with TCP_STRONG_ISS set to 2
OS Fingerprint:
TSeq(Class=TR%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=C0B7%ACK=S++%Flags=AS%Ops=NNTMNW)
T2(Resp=N)
T3(Resp=N)
T4(Resp=N)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)

Uptime 0.107 days (since Tue Sep 14 15:43:22 2004)
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: 60665B23 2CC002B3 C9C05D50 61377D05 7947ED9B
6B5510E9
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 924 seconds
```

The program found an open port and tried to determine the operating system by using a database containing the reactions of various operating systems in answer to certain network packages. The program determined the right operating system (Solaris). Further the information indicates that the OS is using a hardened TCP/IP stack. Listing 7.19 shows the scan of a GNU-Linux system.

LISTING 7.19 Using `nmap` to Reveal the Operating System Used (2)

```
root@unixhost> nmap -P0 -v -v -O 192.168.0.2

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
No tcp,udp, or ICMP scantype specified, assuming SYN Stealth
scan. Use -sP if you really don't want to portscan (and just
want to see what hosts are up).
Host linuxhost (192.168.0.2) appears to be up ... good.
```

```
Initiating SYN Stealth Scan against linuxhost (192.168.0.2)
Adding open port 22/tcp
The SYN Stealth Scan took 379 seconds to scan 1601 ports.
For OSScan assuming that port 22 is open and port 53 is closed
and neither are firewalled
For OSScan assuming that port 22 is open and port 53 is closed
and neither are firewalled
For OSScan assuming that port 22 is open and port 53 is closed
and neither are firewalled
Interesting ports on linuxhost (192.168.0.2):
(The 1597 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
53/tcp     closed      domain
113/tcp    closed      auth
548/tcp    closed      afpovertcp
No exact OS matches for host (If you know what OS is running on
it, see http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.00%P=powerpc-apple-
darwin7.2.0%D=9/14%Time=41471C6D%O=22%C=53)
TSeq(Class=RI%gcd=2%SI=1BEE5E%IPID=I%TS=100HZ)
TSeq(Class=RI%gcd=1%SI=37DCA7%IPID=I%TS=100HZ)
TSeq(Class=RI%gcd=1%SI=37DCDF%IPID=I%TS=100HZ)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=N)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)


Uptime 0.432 days (since Tue Sep 14 08:07:25 2004)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=3661023 (Good luck!)
TCP ISN Seq. Numbers: 7764FBE7 76D94B4B 7687E0EA 7761A693
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 399 seconds
```

The program was not able to determine the operating system this time. It created a so-called TCP/IP Fingerprint of the system instead and asked the user to submit the TCP/IP Fingerprint to the creator of the program if the operating system is known. Further, the output indicates that the TCP/IP stack used is not as secure as the one scanned in Listing 7.18. This means the program estimates a chance to guess the sequence numbers of the TCP/IP stack. Whether you should send the TCP/IP Fingerprint to the creators of the program depends on many things. If the security policy of the company prohibits it, then there are no further questions about it. If you should find an illegal system on your own network using the same TCP/IP Fingerprint, then you would have the advantage of knowing what you are dealing with, provided someone submitted the Fingerprint in the first place. On the other hand, if a cracker manages to gain access to your network, then he would know how to identify another operating system. The world is just not black and white regarding security; it depends what the user of a program is up to.

The functionality shown should enable system administrators to scan their networks for suspect open ports inside their networks. If you use a firewall on a system, then you should definitely use `nmap` to test the configuration and make sure it really works the way it should. Using the program in this way enables you to test different scenarios on the test system before applying the configuration to the production servers. The program is capable of a lot more things; you should invest the time to read the manpage. And one more time: *this program should be used to scan your own machines only*. Scanning the networks of other people this way could get you into trouble in no time.

7.4 NETWORK SECURITY

Depending on the services that are provided in the network, securing a whole network can be an extensive undertaking. The topic has filled various books, and even if you would find the time to read them all, you would still have questions left. Therefore, this section discusses programs and configurations to secure single machines, thus making the network as a whole safer.

7.4.1 Why use `ssh`?

A quick reminder helps to understand the implications. The Internet was not built for commercial use. Originally, most system administrators knew the other administrators of the connected machines, either personally, if the systems were located in the same geographical area, or by phone. Almost all systems connected to the Internet were trustworthy. If someone did something stupid or out of bad intent, then the community either brought him to reason, or the system was removed from the net. Does this sound too good to be true? Now let's move forward to here and now. All people interested in the layout of the current Internet should have a look at the *Internet Mapping Project* (*http://research.lumeta.com/ches/map/*). This site contains some very impressive diagrams or maps of the Internet.

TIP

Who is Bill Cheswick? If you surf to the Internet Mapping Project, take a look at http://research.lumeta.com/ches/. *Bill Cheswick is one of the people working on the Internet Mapping Project. He is, in the author's opinion, one of THE security experts in this world. If you should have a chance to visit one of his lectures, do it. One of the author's favorite quotes from him is this: "The best firewall for Windows is the shrink-wrap around the box." He has a very humorous way of describing all things related to security and computer security. His book* Firewalls and Internet Security: Repelling the Wily Hacker *[Cheswick03] should be on the bookshelf of everyone concerned with security.*

A quick glance at the maps will show everyone how much the Internet has changed. Based on this heritage it becomes obvious that not all protocols were designed with security in mind. The more common examples are the Telnet and the FTP protocol. No one would have sniffed the traffic between two hosts for financial gains in the old days, but the times have changed. Both protocols mentioned transfer everything, including passwords and account names, as clear text over the network, enabling all people with malicious intent to read them at will, not a very promising idea. Because you have no way of knowing who is listening in, even inside your own networks, you need to make sure that if someone eavesdrops, that they will not see anything of importance or, better yet, not see anything at all. The protocols/programs FTP, Telnet, `rsh`, and `rlogin` should not be used any more.

An almost perfect replacement for the programs and protocols mentioned is the `ssh` (secure shell) package. It consists of three programs, `ssh`, `scp`, and `sshd`. The program `ssh` replaces `telnet`, `rlogin`, and `rsh`. It operates more or less like the mentioned programs, but encrypts the connection. The program `scp` (secure copy) replaces the `rcp` (remote copy) command. The syntax is very similar, but the program encrypts the sent and received packages. The program `scp` is not a complete replacement for the `ftp` command, but greatly surpasses the security of that command. The last program `sshd` is a daemon that enables other programs to connect to it.

The `ssh` package is part of almost every *NIX distribution nowadays. If you should have a system connected to a network without that package installed, then you should check the manufacturer's Web site for availability and how to install the software. You need to follow the installation instructions to the letter to ensure that the program works as intended. If you really have no way to install the software on a production system, then you need to be well aware of the inherent risk.

You need to check the configuration of `sshd` on a new system for certain entries to make sure the system is a secure as possible. Listing 7.20 shows an example.

LISTING 7.20 Configuration File for `sshd` (Excerpt)

```
...
#Port 22
#Protocol 2,1
Protocol 2
#ListenAddress 0.0.0.0
...
# Logging
#obsoletes QuietMode and FascistLogging
#SyslogFacility AUTH
#LogLevel INFO
...
# Authentication:

#LoginGraceTime 120
#PermitRootLogin yes
PermitRootLogin no
#StrictModes yes
...
X11Forwarding yes
#X11DisplayOffset 10
...
```

There is much more to configure than the lines shown in the listing. The configuration of `sshd` reads as follows: lines starting with `#` are comments. If such a line contains a setting, then the setting shown is the default, usually compiled into the program. Some entries in Listing 7.20 are more important than others; these entries are shown in bold. The first setting to check is `Protocol`. This setting governs what protocols will be accepted to establish a connection. The commented line contains the default `2,1`. This means that the program tries to establish a connection using the protocol type `2` first; should this fail, it allows you to use protocol type `1`. This setting was changed to `Protocol 2`. Protocol type `1` is considered insecure and should not be used anymore. The configuration file on Solaris systems already contains a warning that the type `1` protocol will be deprecated in the not too distant future. The next line of interest is `SyslogFacility AUTH` (see Section 5.2.2, "Configuring `syslog.conf`"). This line establishes the fact that the program is able to write log files using the `syslog` facility of the system and that the log entries usually start at `info`. If there are messages with a higher level, then these messages will be written to other log files as well, according to the configuration of `syslog`. The value of `PermitRootLogin` should always be `no`, provided there are no really good reasons to allow `root` logins. It is not possible to log in using the `root` account with this setting in place. This ensures that a "lost" `root` password cannot be used to connect to the system from the outside. If you want to work as `root` on the system, you have to log in as a normal user and change to `root` when authenticated on the system. As convenient a straight `root` access may be, you should turn this setting off if possible. The last setting, `X11Forwarding yes`, allows you to use `ssh` to connect an X11 server to the machine and use GUI applications stored on the remote machine. This setting is sometimes considered insecure.

The file `ssh_config` contains the configuration for the client program `ssh`. It does not contain as many settings as the server. `ssh` is used to connect to a server running `sshd`. The only thing to check here is the protocol setting; you should not allow the program to use protocol type `1` as well. The program uses the file in the following manner: if an option was not set on the command line, then the program tries to find the setting in `$HOME/.ssh/`. If the setting could not be found in there, the setting in the general `ssh_config` file is used.

Listing 7.21 demonstrates the use of `ssh` to connect to a remote system.

LISTING 7.21 Login Using `ssh`

```
erikk@osxhost:> ssh -X 192.168.0.2
erikk@192.168.0.2's password:
Last login: Wed Sep 15 15:33:12 2004 from osxhost

erikk@linuxhost:>
```

Because the option `-l` was not used, the name of the current user is used as account name, and the `-X` option enables X11 forwarding. It is possible to start X11 programs on the remote host and display them locally. Regarding the login shown in Listing 7.21, the X Window would appear on the machine `osxhost` while the program runs on `192.168.0.2`.

The second client program is `scp`. It creates an encrypted connection to transfer files between the systems. This program replaces the `rcp` command. Listing 7.22 demonstrates its use.

LISTING 7.22 Transferring a File Using `scp`

```
erikk@unixhost> scp base_netconns.log.040831 \
erikk@192.168.0.2:base_reports/unixhost/.
erikk@192.168.0.2's password:
base_netconns.log.04 100% |*****************************|  2130
00:00
```

Neither the account details nor the transferred data could be intercepted during this transfer. The mode used that addresses *erikk@192.168.0.2* looks like an e-mail address and means the user `erikk` on the system at `192.168.0.2`. The program uses the current username if the name part is omitted from the address. If you need to keep the permissions and the ownership of the files, then you would use the -p option. To copy a directory tree, you would use the option -r (recursive), as shown in Listing 7.23.

LISTING 7.23 Transferring a Directory Tree

```
erikk@unixhost> scp -rp var erikk@192.168.0.2:base_reports/unixhost/.
erikk@192.168.0.2's password:
base_retrans.log.0   100% |********|    10       00:00
sar_idl.log.040831   100% |********|   315       00:00
dw__export_home_user 100% |********|     2       00:00
dw__export_home_user 100% |********|     3       00:00
dw__export_home_user 100% |********|     8       00:00
dw__export_home_user 100% |********|    21       00:00
...
```

The command shown in Listing 7.23 copies the `var` directory and all subdirectories in it to the system at the IP address `192.168.0.2`. If you do not need the display of the transfer in progress, you could use -q (quiet) to suppress it.

As you can see, transfer of data between systems can be done in a secure way by making use of the `ssh` package. This capability is especially important if you need to transfer configuration files or files containing sensitive data from one system to another. You should avoid using the old, insecure commands like `rcp`. You never know who is listening in. If you need to transfer files over the Internet, then using insecure commands should be out of the question anyway.

7.4.2 Unused Ports and Accounts

Every *NIX system uses ports to channel the data streams between systems and even for communication that is internal to one single system. Ports are used by programs and services to wait for messages. Software that sends out e-mail would use the port number 25 to talk to the mail server process and deliver the mail. If you use a system as mail server, you should not block port 25. But provided there is no firewall to block unused ports, it is a good idea to block all unused ports on a given system. This does not mean you should block everything unknown to you, but at least a list of services you definitely won't use. Listing 7.24 shows examples of unused or not to be used ports.

LISTING 7.24 Unused Ports

```
...
# FINGERD - finger daemon
```

```
#finger  stream  tcp6    nowait  nobody  /usr/sbin/in.fingerd
    in.fingerd
...
# RWALLD - rwall daemon (allows others to post messages to users)
#walld/1 tli     rpc/datagram_v  wait    root
/usr/lib/netsvc/rwall/rpc.rwalld        rpc.rwalld
...
# FTPD - FTP server daemon
#ftp     stream  tcp6    nowait  root    /usr/sbin/in.ftpd
       in.ftpd -a
...
```

These are three examples for deactivated services in `/etc/inetd.conf`. You just need to put the `#` sign at the beginning of the line and that line will be considered a comment from now on. If you restart either `inetd` or `xinetd`, then the services will not be available. If `xinetd` is used, then `/etc/inetd.conf` is one of two ways to configure the system. The program usually uses the file `/etc/xinetd.conf` for global settings and the configuration files in `/etc/xinetd.d` to configure the different services. These services can be turned off or on in the respective configuration file by using either `disable = yes` or `disable = no`. If there should be an active connection, then restarting the daemon is not enough to terminate this connection; you need to stop all waiting daemons by hand. Making sure that all services started by `xinetd` are really off is one of the rare situations that warrants a reboot if possible. The `finger` service hails from the time when most of the population on the old *ARPAnet* was trustworthy; it is rarely used these days and should be shut down. The name `rwall` comes from *write to all* and is a quick way of notifying all users on a given system. Again, if you do not know it or make use of it, it should be turned off. Shutting down the last service, `ftp`, could pose a problem. Even if the machine is not used as a server, chances are at least some users use the protocol to transfer files to and from their accounts. If that should be the case, it is your job to convince the users to use `scp` instead because `ftp` transfers passwords and everything else as clear text. System administrators should always make sure that the systems do not react to requests for services that should not be available in the first place. A quick `nmap` scan might reveal interesting ports that could be turned off.

One type of incident makes news more often than it should—breaking into a system through the use of accounts of former users. What holds true for unused ports holds equally true for unused user accounts: if you do not use it, turn it off. It is amazing how simple it is to gather information about possible user accounts, especially accounts that are not in use anymore. It takes just a bit of social engineering to get hold of this information. It is not even bad intention most of the time. Look at this subject from another perspective: what happens to paper that is not used anymore? Will it be shredded to make it harder to gain information from it?

Will it be dumped to the recycling bin without checking first? If you are not a hundred percent sure that every piece of paper goes into the shredder, then the chances are good that someone else tried to gather the information this way. It would not be the first time that sensitive data and passwords were leaked that way.

7.4.3 Monitoring Network Usage

The third component to monitor is the network the systems are connected to. As already mentioned, monitoring a complete network is beyond the scope of this book; this section discusses the monitoring of single components, the systems connected to the network. To learn more about monitoring networks, you should use the Internet to investigate which topics are of interest, but there is no catch-all approach to monitoring networks due to the complexity of the task.

TIP

All software packages discussed in this section can be used on a laptop without any problems, provided the laptop is equipped with a real operating system. This means preemptive multitasking is a must. The exercise to monitor network connections would be futile otherwise. This is not about monitoring a network on a permanent basis, but about providing the means to execute a monitoring session if needed. A laptop with GNU-Linux, Solaris x86, or some BSD variant or an Apple PowerBook running OS X can be transported to every spot in the network and used to start a quick monitoring session to identify a problem. The operating systems mentioned usually come with the software already installed or on the distribution media ready to install, so there is no need to download the software separately and install it. The "real" monitoring of a network or even a network segment requires a finely tuned system dedicated to the task.

7.4.3.1 Monitoring Network Traffic with `snoop` (Solaris)

The command `snoop` is part of the *NIX derivate Solaris. It is located at `/usr/sbin/snoop` and executable only for the superuser. The reason for that is based on the way the program operates; it switches the NIC to promiscuous mode. In this mode, the NIC "sees" every network package that passes, instead of only those meant for its own IP-address, thus enabling the user to snoop on other people's data.

There is another program with similar functionality ported to almost any *NIX flavor, including Solaris, called `tcpdump`, see Section 7.4.3.2, "Monitoring Network Traffic with `tcpdump`." Which program you use depends on your requirements and the dominant operating system in use.

Listing 7.25 illustrates the power of `snoop`; it shows how to `snoop` a `dig` request.

LISTING 7.25 Using `snoop` to Sniff a Connection

```
root@unixhost:> snoop -o digtest.cap host 192.168.0.5
Using device /dev/hme (promiscuous mode)
2 ^C
root@unixhost:> ls -l
total 1
-rw-r--r--    1 root     other         248 Jun 24 14:47 digtest.cap
```

In Listing 7.25 `snoop` was used to capture packages that had the IP address `192.168.0.5` as either source or destination. The captured packages were saved to the file `digtest.cap`. The line with the number `2` shows the number of captured packages. The resulting file—especially its permissions—should give you a visual indication why captured files should be stored out of reach from other users. They are world-readable. You could use the command `umask 0077`, but storing the files in a secure location seems to be the best approach. The command issued on host `192.168.0.5` was `dig examplehost3`, as shown in Listing 7.26.

LISTING 7.26 `dig` Command Issued on `192.168.0.5`

```
root@unixhost:> dig examplehost3

; <<>> DiG 8.3 <<>> examplehost3
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; QUERY SECTION:
;;      examplehost3, type = A, class = IN

;; ANSWER SECTION:
examplehost3.          30M IN A        192.168.0.3

;; Total query time: 224 msec
;; FROM: unixhost to SERVER: default -- 192.168.0.150
;; WHEN: Thu Jun 24 14:47:09 2004
;; MSG SIZE  sent: 39  rcvd: 55
```

To inspect the captured packages, you use the command `snoop -i digtest.cap`, as shown in Listing 7.27.

LISTING 7.27 Displaying the Captured Packages (Default View)

```
root@unixhost:> snoop -i digtest.cap
  1   0.00000 unixhost -> mylocaldns DNS C examplehost3. Internet Addr
?
  2   0.22161 mylocaldns -> unixhost DNS R examplehost3. Internet Addr
 192.168.0.3
```

The information in Listing 7.27 tells you that a system named `examplehost1` queried a DNS about the IP address of a machine named `examplehost3` and the resulting answer. If this is all you need to know, the default view is sufficient. To drill a little deeper, you add the option `-V`, as shown in Listing 7.28.

LISTING 7.28 Displaying the Captured Packages a Little More Verbose

```
root@unixhost:> snoop -V -i digtest.cap
________________________________
  1   0.00000 unixhost -> mylocaldns    ETHER Type=0800 (IP), size = 81
      bytes
  1   0.00000 unixhost -> mylocaldns IP  D= 192.168.0.150 S=192.168.0.5
      LEN=67, ID=54825, TOS=0x0, TTL=255
  1   0.00000 unixhost -> mylocaldns UDP D=53 S=33107 LEN=47
  1   0.00000 unixhost -> mylocaldns DNS C examplehost3. Internet Addr ?
________________________________
  2   0.22161 mylocaldns -> unixhost    ETHER Type=0800 (IP), size =
      97 bytes
  2   0.22161 mylocaldns -> unixhost    IP  D=192.168.0.5
      S=192.168.0.150 LEN=83, ID=0, TOS=0x0, TTL=57
  2   0.22161 mylocaldns -> unixhost    UDP D=33107 S=53 LEN=63
  2   0.22161 mylocaldns -> unixhost    DNS R examplehost3.
      Internet Addr 192.168.0.3
```

Now the sequence of the conversation between the hosts becomes visible, but the details are still hidden. To display all captured information, you use the option `-v`. Listing 7.29 shows the result.

LISTING 7.29 Displaying All Captured Information

```
# snoop -v -i digtest.cap
ETHER:  ----- Ether Header -----
ETHER:
```

```
ETHER:  Packet 1 arrived at 14:47:9.74
ETHER:  Packet size = 81 bytes
ETHER:  Destination = 0:30:ab:4:b2:7f,
ETHER:  Source      = 8:0:20:83:7:93, Sun
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:         .... ..0. = not ECN capable transport
IP:         .... ...0 = no ECN congestion experienced
IP:   Total length = 67 bytes
IP:   Identification = 54825
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 255 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = 2038
IP:   Source address = 192.168.0.5, unixhost
IP:   Destination address = 192.168.0.150, mylocaldns
IP:   No options
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 33107
UDP:  Destination port = 53 (DNS)
UDP:  Length = 47
UDP:  Checksum = 6D6E
UDP:
DNS:  ----- DNS Header -----
DNS:
DNS:  Query ID = 2
DNS:  Opcode: Query
DNS:  RD (Recursion Desired)
DNS:  1 question(s)
DNS:      Domain Name: examplehost3.
```

```
DNS:      Class: 1 (Internet)
DNS:      Type:  1 (Address)
DNS:

ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 2 arrived at 14:47:9.97
ETHER:  Packet size = 97 bytes
ETHER:  Destination = 8:0:20:83:7:93, Sun
ETHER:  Source      = 0:30:ab:4:b2:7f,
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4
IP:   Header length = 20 bytes
IP:   Type of service = 0x00
IP:         xxx. .... = 0 (precedence)
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:         .... ..0. = not ECN capable transport
IP:         .... ...0 = no ECN congestion experienced
IP:   Total length = 83 bytes
IP:   Identification = 0
IP:   Flags = 0x4
IP:         .1.. .... = do not fragment
IP:         ..0. .... = last fragment
IP:   Fragment offset = 0 bytes
IP:   Time to live = 57 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = bc52
IP:   Source address = 192.168.0.150, mylocaldns
IP:   Destination address = 192.168.0.5, unixhost
IP:   No options
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 53
UDP:  Destination port = 33107
UDP:  Length = 63
UDP:  Checksum = 3647
UDP:
DNS:  ----- DNS Header -----
```

```
DNS:
DNS:  Response ID = 2
DNS:  RA (Recursion Available)
DNS:  Response Code: 0 (OK)
DNS:  Reply to 1 question(s)
DNS:      Domain Name: examplehost3.
DNS:      Class: 1 (Internet)
DNS:      Type:  1 (Address)
DNS:
DNS:  1 answer(s)
DNS:      Domain Name: examplehost3.
DNS:      Class: 1 (Internet)
DNS:      Type:  1 (Address)
DNS:      TTL (Time To Live): 1800
DNS:      Address: 192.168.0.3
DNS:
DNS:  0 name server resource(s)
DNS:  0 additional record(s)
```

Listing 7.29 displays all the captured data. Given the amount of information contained in the captured file, using `-v` during live captures is not advisable. If you suspect there is some traffic inside your network that does not belong there, `snoop` can be used to monitor traffic without saving it to a file, but a more thorough examination of the suspect packages is impossible because `snoop` discards them as soon as they are displayed.

Covering the various options `snoop` provides would fill a separate book; Table 7.1 contains some examples for `snoop`.

As already mentioned, this table contains only a fraction of what is possible with `snoop`. All examples can be used to filter through an existing capture file, or you can use the option `–i` and the filename.

Every Solaris administrator should have a closer look at the `snoop` manpage.

7.4.3.2 Monitoring Network Traffic with `tcpdump`

Almost everything said about `snoop` in the previous section holds equally true for `tcpdump`. The program actually provides even more features. It was developed at the *Lawrence Berkeley National Laboratory* at the University of California in Berkeley and is available for almost all *NIX variants. It is already included in most of the GNU-Linux distributions and most BSD-based distributions, including OS X.

To monitor packages to and from a certain IP address, you could use the command shown in Listing 7.30.

TABLE 7.1 snoop Examples

Command	Result
snoop	Captures all packages "visible" to the network card.
snoop host 192.168.0.2	Captures all packages with source or destination IP address 192.168.0.2.
snoop host 192.168.0.2 and 192.168.0.3	As before, but only packages between 192.168.0.2 and 192.168.0.3.
snoop host 192.168.0.2 or 192.168.0.3	Captures only packages with source or destination IP address 192.168.0.2 or 192.168.0.3.
snoop host 192.168.0.2 and 192.168.0.3 and (tcp or udp) and port 80	Captures all TCP and UDP packages sent between the IP addresses 192.168.0.2 and 192.168.0.3 using port 80.

LISTING 7.30 Monitoring an IP Address with `tcpdump`

```
root@unixhost> tcpdump host 192.168.0.5
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on en0, link-type EN10MB (Ethernet), capture size 96 bytes
13:23:47.704092 IP linuxhost.netarx > unixhost.ssh: P
 2542587687:2542587735(48) ack 404157412 win 63088
 <nop,nop,timestamp 53538553 7873785>
13:23:47.708404 arp who-has linuxhost (Broadcast) tell unixhost
13:23:47.708552 arp reply linuxhost is-at 00:a0:cc:5b:cf:5c
13:23:47.709079 IP unixhost.ssh > linuxhost.netarx: P 1:49(48)
 ack 48 win 49232 <nop,nop,timestamp 8254157 53538553>
13:23:47.709293 IP linuxhost.netarx > unixhost.ssh: . ack 49 win
 63088 <nop,nop,timestamp 53538554 8254157>
13:23:47.714654 IP unixhost.ssh > linuxhost.netarx: P 49:97(48) ack
 48 win 49232 <nop,nop,timestamp 8254158 53538554>
13:23:47.714813 IP linuxhost.netarx > unixhost.ssh: . ack 97 win
 63088 <nop,nop,timestamp 53538554 8254158>
^C
7 packets captured
139 packets received by filter
0 packets dropped by kernel
```

If you need to write the captured packages to a file for later inspection, you could use the option –w. To read the captured file later, use the option –r, as shown in Listing 7.31.

LISTING 7.31 Capturing to a File and Reading the Stored Information with `tcpdump`

```
root@unixhost> tcpdump -c 10 -w osxhost.cap host 192.168.0.3
tcpdump: listening on eth0
10 packets received by filter
0 packets dropped by kernel

root@unixhost> tcpdump -r osxhost.cap
11:38:08.994802 osxhost.52145 > 193.99.144.71.http: S
 2221628259:2221628259(0) win 65535 <mss 1460,nop,wscale 0,
nop,nop,timestamp 1790676964 0> (DF)
11:38:09.064904 193.99.144.71.http > osxhost.52145: S
 1702175240:1702175240(0) ack 2221628260 win 8192 <mss 1452>
11:38:09.065196 osxhost.52145 > 193.99.144.71.http: . ack 1
 win 65535 (DF)
...
```

The packages are stored in the file named `osxhost.cap`. The number of packages to capture is ten (`-c 10`). If you simply read the file using the –r option, only the requested data will be shown. This means that the data is formatted the default way. If you want to see more detail, then you have to use the option –v, as shown in Listing 7.32.

LISTING 7.32 Displaying the File in More Detail

```
root@unixhost> tcpdump -v -r osxhost.cap
11:38:08.994802 osxhost.52145 > 193.99.144.71.http: S [tcp sum ok]
 2221628259:2221628259(0) win 65535 <mss 1460,nop,wscale 0,nop,nop,
timestamp 1790676964 0> (DF) (ttl 64, id 63195, len 60)
11:38:09.064904 193.99.144.71.http > osxhost.52145: S [tcp sum ok]
 1702175240:1702175240(0) ack 2221628260 win 8192 <mss 1452> (ttl 59,
 id 59312, len 44)
11:38:09.065196 osxhost.52145 > 193.99.144.71.http: . [tcp sum ok]
 ack 1 win 65535 (DF) (ttl 64, id 63196, len 40)
...
```

The resulting output contains more information about the type of package and the used parameters. If you need to see an even more detailed output of the captured data, you could use the option –vv, and to get even more than that, you could use –vvv. This option displays all information stored in the file. If you want to see the contents of the packages, you could use –x to display them in hexadecimal and –X to display them in ASCII. Listing 7.33 shows an example.

LISTING 7.33 Displaying the Contents of a Package

```
erikk@unixhost> tcpdump -c 1 -x -X -r osxhost.cap

11:38:09.065732 osxhost.52145 > 193.99.144.71.http: P 1:548(547) ack 1
 win 65535 (DF)
0x0000   4500 024b f6dd 4000 4006 2f79 c0a8 0003
E..K..@.@./y....
0x0010   c163 9047 cbb1 0050 846b 5b64 6575 2209
.c.G...P.k[deu".
0x0020   5018 ffff 394c 0000 4745 5420 2f73 6563
P...9L..GET./sec
0x0030   7572 6974 792f 2048 5454 502f 312e 310d
urity/.HTTP/1.1.
0x0040   0a48 6f73 743a 2077 7777 2e68 6569 7365
.Host:.www.heise
0x0050   2e64
.d
...
```

Listing 7.33 demonstrates the display of a package. How many bytes of a package will be captured depends on the operating system, usually 68 bytes–96 bytes on Solaris. You can set the number of bytes captured for each package via the –s option. If you would use –s 400, then the first 400 bytes of every package will be captured; if you need the whole package, regardless of the number of bytes, you could use the option –s 0. The latter is rarely necessary, though. If you use that option, you need to be careful regarding the size of the captured file. It could grow very fast, depending on network traffic.

The program tcpdump is, like snoop, the perfect tool to monitor your own network. Neither program should be used to spy on users, but the ethics of system administration prohibits that kind of behavior anyway. One drawback regarding tcpdump is the need to fully understand the internals of the protocols used to make most of the captured data. The program is an excellent choice for tasks like monitoring the network for connections to an IP address deemed "dangerous," for example. Table 7.2 contains some examples of how to filter the packages by using instructions tcpdump understands.

This table barely scratches the surface regarding the capabilities of tcpdump. You should take the time and read the program's manpage. To use tcpdump with a

GUI, you could use the program `ethereal` (see Appendix B for ways to obtain the program).

TABLE 7.2 `tcpdump` Examples

Command	Result
`tcpdump host 192.168.0.3`	Captures packages with either source or destination 192.168.0.3.
`tcpdump src host 192.168.0.3`	Captures only packages with source 192.168.0.3; `dst` would capture all packages with the IP address as destination.
`tcpdump host 192.168.0.3 and 192.168.0.200`	Captures packages sent between the two IP addresses.
`tcpdump host 192.168.0.3 and not port 22`	Captures packages to and from 192.168.0.3, not using port 22 (ssh).
`tcpdump host 192.168.0.3 and not 192.168.0.42`	Captures all network activities of 192.168.0.3 as long as they do not include 192.168.0.42.

7.4.4 Intrusion Detection

Intrusion Detection Systems (IDS) are used to identify suspicious activities inside a network or to detect a break-in. These systems use a rule-matching algorithm to detect packages or patterns that lead to the conclusion that there could be a break-in, virus outbreak, or other unwanted activity going on. The word *could* in the last sentence is of importance; there are "legal" activities that are so close to the rules of a break-in or virus outbreak that they trigger an alarm in an IDS system (the packets sent between Outlook and Exchange, for example).

This means that any IDS system is only as reliable as the rules it uses to detect activities inside the network. Keeping these rules current is a task that is often underestimated. It also heavily depends on the expectations you have regarding the use of an IDS system. If the system should not only monitor the network traffic of the *NIX systems, but should also monitor the network traffic of the existing Windows systems, then you have to make sure to run a very tight update cycle, more than twice a day sometimes. The people targeting Windows systems are very resourceful and distribute new malware often.

There are two basic kinds of IDS systems: hardware-based and software-based. The hardware-based systems obviously run software, too. The distinction concerns the fact that hardware-based systems are delivered with customized hardware to improve the performance of the system. One of the common manufacturers of hardware-based IDS systems is Cisco Systems. The software-based system does not imply that it should be installed on the file server, or any other system running additional services for that matter, but that the system is delivered as software only, and the customer needs to supply a machine suitable for the job. It does not matter what type of IDS is deployed; both types suffer from a common drawback—they might lose packages during phases of heightened network activity. If one or more of these lost packages contains the information that something is going on and would have triggered the alarm, then the best IDS system is worth next to nothing in that case. The only thing you can get out of this situation is a feeling of security. You should always have more than one system on this specific job, just to make sure that everything is monitored the way it should be. The log files created by the baselining scripts represent another aspect in this case (see Section 3.1, "What Is Baselining?"). These scripts should be monitored as well. If there are unusual spikes in the counters, then one should check the other systems as well.

A description of how to install and implement a software-based IDS system and how to maintain the system is beyond the scope of this book. If you want to learn more about using IDS systems for permanent monitoring, you should have a look at the snort.org Web site at *http://www.snort.org*. This section discusses how to use an IDS system to run quick checks on certain parts of a network to make sure everything works as expected. If you want to set up a system to monitor a network or a segment of a network, then you have to have a dedicated system, fine-tuned for the task, using special hardware to achieve the best results.

7.4.4.1 IDS with `snort`

The program `snort` is available for almost all *NIX-based systems and comes with a GPL license. More information can be found on the Web site *http://www.snort.org*. A current version for your brand of *NIX should be available at the Web sites in Appendix B; the other way to get `snort` is to download the source and compile the program yourself.

`snort` is not that different to `tcpdump` (see Section 7.4.3.2 "Monitoring Network Traffic with `tcpdump`"), actually. You could use `snort` as a replacement for `tcpdump` or `snoop` (see Section 7.4.3.1, "Monitoring Network Traffic with `snoop`"), but the interesting part of `snort` starts to shine as soon as the program starts in IDS mode. The program uses a set of rules to detect anomalies in the network when run in IDS mode and is able to notify you about its findings. Independent tests proved that `snort` is up to par with even the most expensive hardware-based IDS systems, provided it runs on a machine powerful enough to supply the program with the needed resources for the task. But `snort`'s requirements regarding the hardware are not that high to begin with.

If you want to use `snort` as a *sniffer*, then all you have to do is to start the program with the options `–dv`, as shown in Listing 7.34.

LISTING 7.34 Using `snort` as Sniffer

```
root@unixhost> snort -dv0 host 192.168.0.3
Running in packet dump mode
Log directory = /var/log/snort

Initializing Network Interface eth0

        --== Initializing Snort ==--
Initializing Output Plugins!
Decoding Ethernet on interface eth0

        --== Initialization Complete ==--

-*> Snort! <*-
Version 2.2.0 (Build 30)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
09/16-16:14:16.791992 xxx.xxx.xxx.xxx:143 -> xxx.xxx.xxx.xxx:54164
TCP TTL:32 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x26E0FFDC  Ack: 0x0  Win: 0x0  TcpLen: 20
63 6B 69                                         cki

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/16-16:14:27.804819 xxx.xxx.xxx.xxx:54198 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x0 ID:38668 IpLen:20 DgmLen:60 DF
******S* Seq: 0x1E86E0B7  Ack: 0x0  Win: 0xFFFF  TcpLen: 40
TCP Options (6) => MSS: 1460 NOP WS: 0 NOP NOP TS: 1790705793 0

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/16-16:14:27.875256 xxx.xxx.xxx.xxx:80 -> xxx.xxx.xxx.xxx:54198
TCP TTL:59 TOS:0x0 ID:33807 IpLen:20 DgmLen:44
***A**S* Seq: 0x2015B4B7  Ack: 0x1E86E0B8  Win: 0x2000  TcpLen: 24
TCP Options (1) => MSS: 1452

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/16-16:14:27.875503 xxx.xxx.xxx.xxx:54198 -> xxx.xxx.xxx.xxx:80
TCP TTL:64 TOS:0x0 ID:38669 IpLen:20 DgmLen:40 DF
***A**** Seq: 0x1E86E0B8  Ack: 0x2015B4B8  Win: 0xFFFF  TcpLen: 20

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

09/16-16:14:27.876053 xxx.xxx.xxx.xxx:54198 -> xxx.xxx.xxx.xxx:80
```

```
TCP TTL:64 TOS:0x0 ID:38670 IpLen:20 DgmLen:595 DF
***AP*** Seq: 0x1E86E0B8  Ack: 0x2015B4B8  Win: 0xFFFF  TcpLen: 20
47 45 54 20 2F 73 65 63 75 72 69 74 79 2F 6E 65  GET /security/ne
77 73 2F 6D 65 6C 64 75 6E 67 2F 35 31 31 35 37  ws/meldung/51157
20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F 73 74 3A   HTTP/1.1..Host:
20 77 77 77 2E 68 65 69 73 65 2E 64 65 0D 0A 43   www.heise.de..C
6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D  onnection: keep-
61 6C 69 76 65 0D 0A 52 65 66 65 72 65 72 3A 20  alive..Referer:
...
```

If `snort` is used as a sniffer, then the program behaves in the same way as `tcpdump`. The option `–O` is used to obfuscate the found IP addresses. The option `–L` allows you to use `snort` to write a captured file in the `tcpdump` format. Let's proceed and use snort as IDS system.

7.4.4.2 Configuring `snort`

You need a configuration file to use `snort` in IDS mode. The file is usually stored in `/etc/snort/snort.conf`. If you should want to use a different configuration file, you could supply the file with the `–c` option. One setting in the configuration file points to the directory containing the IDS rules. The file supplied with the `snort` installation can be used as it is, with two small changes. First, you need to change the variable `HOME_NET` from `any` to the IP range your network is using. If the network uses the range from 192.168.0.1 to 192.168.0.254, for example, then you need to change the entry to 192.168.0.0/24. Second, you need to make sure that the variable `RULE_PATH` points to the right directory, the one the rules are stored in. These rules should be updated periodically from *http://www.snort.org*. Because the rules at snort.org are updated every hour, you should take the time to download them before running an important check. Now you can start the program with the command `snort –c <path to configuration file>`. The program will answer with some diagnostic messages. As long as there are no errors, the program runs IDS mode, scanning the network for packages. The key combination Ctrl-C terminates the program again. When the program terminates, it displays a message containing the number of packages seen and indicating if an alarm was triggered, as shown in Listing 7.35.

LISTING 7.35 Messages During the Termination of `snort`

```
====================================================

Snort received 293 packets
    Analyzed: 293(100.000%)
    Dropped: 0(0.000%)
```

```
====================================================
Breakdown by protocol:
    TCP: 210        (71.672%)
    UDP: 58         (19.795%)
   ICMP: 3          (1.024%)
    ARP: 2          (0.683%)
  EAPOL: 0          (0.000%)
   IPv6: 16         (5.461%)
    IPX: 0          (0.000%)
  OTHER: 4          (1.365%)
DISCARD: 0          (0.000%)
=====================================================
Action Stats:
ALERTS: 63
LOGGED: 63
PASSED: 0
=====================================================
TCP Stream Reassembly Stats:
    TCP Packets Used: 76         (25.939%)
    Stream Trackers: 3
...
```

One of the important lines here is the one named `Dropped:`. As long as the value shown is zero, `snort` did not drop any packages. This means the program got to analyze every single packet. The other important line contains the word `ALERTS:`. As long as the value is zero, too, `snort` did not find anything of importance—based on the current rules, of course. If there were any alerts, then `snort` would have stored the information about them and the packets in the file `/var/log/snort/alert`. This files needs to be examined in case of an alert.

7.4.6.3 How to Read the Results

An entry in the file alerts looks similar to that shown in Listing 7.36.

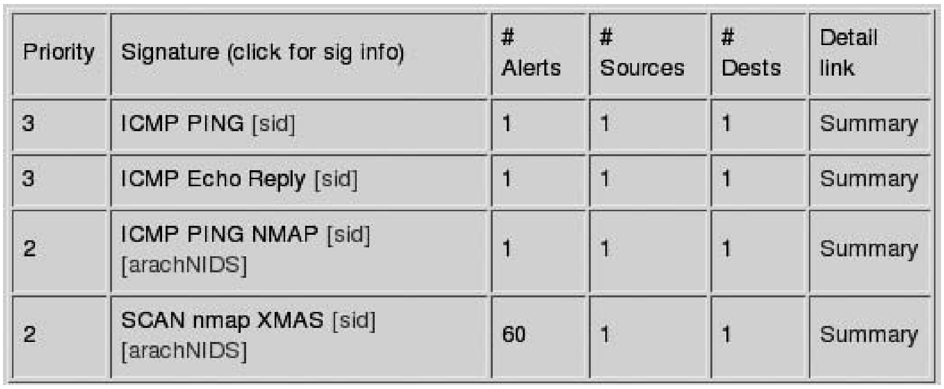
LISTING 7.36 Entry in `alert`

```
...
[**] [1:1228:6] SCAN nmap XMAS [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/16-14:41:46.193112 192.168.0.3:57287 -> 192.168.0.5:265
TCP TTL:44 TOS:0x0 ID:40981 IpLen:20 DgmLen:40
**U*P**F Seq: 0x0  Ack: 0x0  Win: 0x400  TcpLen: 20  UrgPtr: 0x0
[Xref => http://www.whitehats.com/info/IDS30]
...
```

It is obvious the alert was triggered by an `nmap` scan. The second line contains an assessment about the severity of the alert based on the current rules. The third line contains date and time of the alert and the IP addresses of the hosts, followed by technical data about the captured packet. The last line contains a link to a Web site containing a detailed description of the possible incident. If the link points to a *whitehats* site, even the rule that triggered the alarm is shown, but not all rules contain such a link.

The use of `snort` has a distinct advantage: if there is a warning about a new threat at the Internet Storm Center (*http://isc.sans.org*), then the warning sometimes contains a rule to be used with `snort` right away. If the rule is a quick fix to hold a new threat at bay, this means it was not tested in the usual way. The rule is usually accompanied by a warning about this fact, and the tested rule can be expected within the next couple of hours at *http://www.snort.org*.

If you want to avoid the time-consuming search through the `alert` file, then using a program to do it for you would be helpful. There are plenty of programs you can find on the snort.org Web site. One of these programs is a Perl script called `snortsnarf.pl`. The program creates an HTML page, showing the alerts sorted by priority (see Figure 7.1).



Priority	Signature (click for sig info)	# Alerts	# Sources	# Dests	Detail link
3	ICMP PING [sid]	1	1	1	Summary
3	ICMP Echo Reply [sid]	1	1	1	Summary
2	ICMP PING NMAP [sid] [arachNIDS]	1	1	1	Summary
2	SCAN nmap XMAS [sid] [arachNIDS]	60	1	1	Summary

FIGURE 7.1 HTML page created by `snortsnarf.pl`.

The installation of the script is easy; it has to be downloaded from *http://www.snort.org/dl/contrib/data_analysis/snortsnarf/*. The Web site of *Silicon Defense* is no longer available because the company ceased to exist after the last round of Defense Advanced Research Project Agency (DARPA) budget cuts. You have to unpack the file `SnortSnarf-021111.1.tar.gz`. The resulting directory contains the script, which in turn should be started from this directory. See Listing 7.37 for details.

LISTING 7.37 Unpacking and Using `snortsnarf.pl`

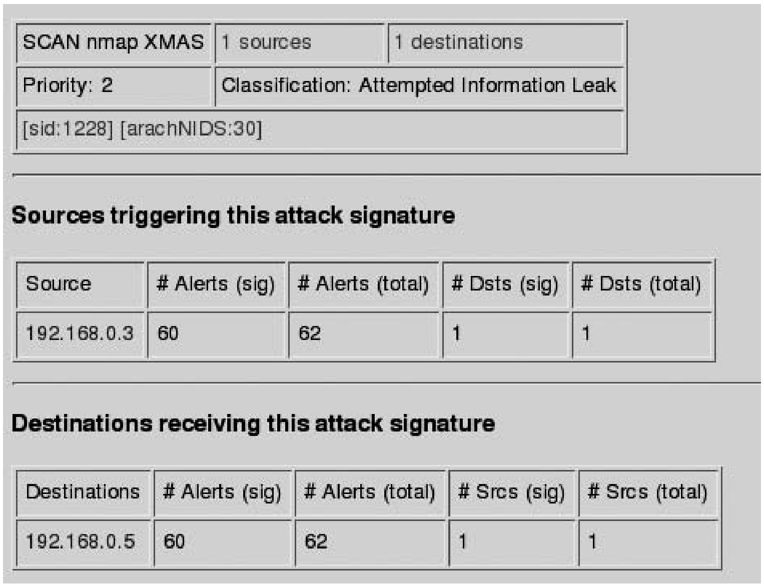
```
root@unixhost> tar xzf SnortSnarf-021111.1.tar.gz
root@unixhost> ls
.  ..  SnortSnarf-021111.1  SnortSnarf-021111.1.tar.gz
root@unixhost> cd SnortSnarf-021111.1/
root@unixhost> ls
.        README            Usage                    nmap2html
..       README.SISR       cgi                      sisr
COPYING  README.nmap2html  include                  snortsnarf.pl
Changes  Time-modules      new-annotation-base.xml  utilities
root@unixhost> ./snortsnarf.pl /var/log/snort/alert
root@unixhost> ls
.        README            Usage                    nmap2html
utilities
..       README.SISR       cgi                      sisr
COPYING  README.nmap2html  include                  snfout.alert
Changes  Time-modules      new-annotation-base.xml  snortsnarf.pl
root@unixhost> ls snfout.alert/
.  ..  192  212  SDlogo.gif  index.html  sig  topdests.html
topsrcs.html
```

The directory `snfout.alert` contains the Web site generated by the script. This set of HTML pages can be opened with every Web browser. If you click on Summary, you are presented with all details about an alarm (see Figure 7.2).

It is even possible to click the IP addresses to get a glance on the details about a system and links to perform a name resolution (see Figure 7.3).

Many more scripts can help you with the analysis of the log files. You just have to choose your weapons at *http://www.snort.org/dl/contrib/data_analysis/*.

Most IDS systems work in a way similar to the one shown here. There are even systems that should be able to "learn" according to the information presented. This means that those systems are able to adapt to the network traffic in your network. Such systems should always be controlled by a pure rule-based system like `snort`. You should then compare the results of both systems to get a feeling about the pros and cons of both approaches.



SCAN nmap XMAS	1 sources	1 destinations
Priority: 2	Classification: Attempted Information Leak	
[sid:1228] [arachNIDS:30]		

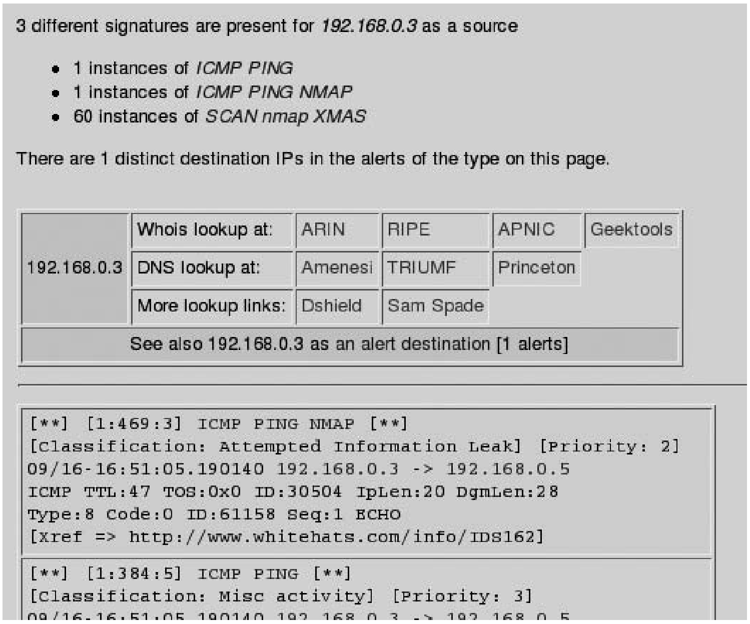
Sources triggering this attack signature

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
192.168.0.3	60	62	1	1

Destinations receiving this attack signature

Destinations	# Alerts (sig)	# Alerts (total)	# Srcs (sig)	# Srcs (total)
192.168.0.5	60	62	1	1

FIGURE 7.2 Alarm details.



3 different signatures are present for *192.168.0.3* as a source

- 1 instances of *ICMP PING*
- 1 instances of *ICMP PING NMAP*
- 60 instances of *SCAN nmap XMAS*

There are 1 distinct destination IPs in the alerts of the type on this page.

192.168.0.3	Whois lookup at:	ARIN	RIPE	APNIC	Geektools
	DNS lookup at:	Amenesi	TRIUMF	Princeton	
	More lookup links:	Dshield	Sam Spade		
See also 192.168.0.3 as an alert destination [1 alerts]					

```
[**] [1:469:3] ICMP PING NMAP [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/16-16:51:05.190140 192.168.0.3 -> 192.168.0.5
ICMP TTL:47 TOS:0x0 ID:30504 IpLen:20 DgmLen:28
Type:8 Code:0 ID:61158 Seq:1 ECHO
[Xref => http://www.whitehats.com/info/IDS162]
```

```
[**] [1:384:5] ICMP PING [**]
[Classification: Misc activity] [Priority: 3]
09/16-16:51:05.190140 192.168.0.3 -> 192.168.0.5
```

FIGURE 7.3 Details of an IP address.

7.4.5 Firewalls

In its most common understanding, a *firewall* is a piece of software running on a dedicated hardware, protecting your network against threats from the outside world by controlling and restricting access to it. This sentence opens up a number of questions. For example, what if the firewall guarding the network fails and *sed quis custodiet ipsos custodes* (who guards the guards)?

Just one firewall for the network, leaving everything else unprotected, can't be enough. Beyond creating layers of firewalls if needed, deploying protection on single machines seems to be the answer. It won't be the "cure-all" secret ingredient, but hardens your systems against threats from both the inside and the outside. Breaking through the firewall is not the only way into the network; most companies supply their employees, or at least some of them, with laptops. Backing them up, let alone securing them, can be rather complicated. A virus written for that operating system cannot harm a *NIX system, but a Trojan or the latest incarnation of spyware poses a security risk to all unencrypted traffic inside the network.

Testing every laptop before it enters the secured network can be done, but this is time consuming and error prone. Making sure all patches are deployed requires constant attention. The growing number of *NIX-based laptops eases the problem a little, but they need patching, too. The problem becomes even worse if one of your servers, trusted by every other system in the network, is compromised. The last line of defense should always be a firewall protecting the system itself, configured to allow only traffic supposed to reach the machine in a controlled fashion.

7.4.7.1 Using a Firewall on Every System

There is a reason to use a firewall on every system in the network. For example, scanning a freshly installed GNU-Linux system with `nmap` shows a result similar to the one shown in Listing 7.38.

LISTING 7.38 `nmap` Scan of a Newly Installed System

```
root@unixhost:> nmap 192.168.0.2

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on linuxhost (192.168.0.2):
(The 1596 ports scanned but not shown below are in state: closed)
Port       State      Service
22/tcp     open       ssh
111/tcp    open       sunrpc
```

```
631/tcp    open        ipp
6000/tcp   open        X11
10000/tcp  open        snet-sensor-mgmt

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
```

The system in Listing 7.38 was hardened by hand, but some ports are still open. Now by simply starting the supplied firewall, the picture looks different without too much work on the part of the system administrator (see Listing 7.39).

LISTING 7.39 `nmap` Scan of the Same System with a Running Firewall

```
root@unixhost:> nmap 192.168.0.2

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on linuxhost (192.168.0.2):
(The 1596 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh
113/tcp    closed      auth

Nmap run completed -- 1 IP address (1 host up) scanned in 160 seconds
```

The output in Listing 7.39 should give you some peace of mind while the system is unattended, and the amount of work to achieve this state was a couple of mouse clicks.

Most *NIX flavors come with a firewall these days; why not use it? The ones in SuSE Linux and Apple's OS X can be configured with a, more or less, friendly GUI. The supplied firewall of Solaris can be configured by GUI or command line. The command-line approach has some drawbacks, but allows a more precise approach to securing the system. For example, you have no way to limit `ssh` connections to certain network ranges using the SuSE GUI.

The difference between an installed system and a system ready for production was discussed in Section 5.2, "How to Ready a System for Production." The differences between such systems are much bigger than most people realize. See Listing 7.40 for the output of an `nmap` run against a freshly installed Solaris system.

LISTING 7.40 `nmap` Scan of a Solaris System

```
root@unixhost:> nmap 192.168.0.5

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on unixhost (192.168.0.5):
```

```
(The 1570 ports scanned but not shown below are in state: closed)
Port       State      Service
7/tcp      open       echo
9/tcp      open       discard
13/tcp     open       daytime
19/tcp     open       chargen
21/tcp     open       ftp
22/tcp     open       ssh
23/tcp     open       telnet
25/tcp     open       smtp
37/tcp     open       time
79/tcp     open       finger
111/tcp    open       sunrpc
512/tcp    open       exec
513/tcp    open       login
514/tcp    open       shell
515/tcp    open       printer
540/tcp    open       uucp
587/tcp    open       submission
665/tcp    open       unknown
898/tcp    open       unknown
4045/tcp   open       lockd
6000/tcp   open       X11
6112/tcp   open       dtspc
7100/tcp   open       font-service
32771/tcp  open       sometimes-rpc5
32772/tcp  open       sometimes-rpc7
32773/tcp  open       sometimes-rpc9
32774/tcp  open       sometimes-rpc11
32775/tcp  open       sometimes-rpc13
32776/tcp  open       sometimes-rpc15
32779/tcp  open       sometimes-rpc21
32780/tcp  open       sometimes-rpc23

Nmap run completed -- 1 IP address (1 host up) scanned in 50 seconds
```

The system in Listing 7.40 needs some serious work or a configured firewall. Listing 7.41 shows the same system with the firewall turned on.

LISTING 7.41 `nmap` Scan of the Same System with a Running Firewall

```
root@unixhost:> nmap 192.168.0.5
```

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on unixhost (192.168.0.5):
(The 1600 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 162 seconds
```

The difference between Listing 7.40 and Listing 7.41 is about 20 minutes of configuration and the command `/etc/init.d/sunscreen start`. *Sunscreen* is the name of the firewall supplied with Solaris.

If you don't feel comfortable using the command line, use the GUI and examine the resulting configuration when time permits. If the GUI allows you to lock down the system in a way that seems sufficient, then the work is done. If not, rest assured that there is no black magic involved in configuring firewalls. Try to gain a basic understanding and verify the results on your test system.

7.4.7.2 Firewalling the Inside of the Network

The configuration of most firewalls allows every connection to be established if the connection originates from the inside of the network. Most firewalls allow you to configure rules for the connections from the inside to the outside as well. Therefore, you are able not only to put rules on the connections from the outside, but also to apply rules to connections from your computers. You should make use of this capability. If you create a list of allowed ports that can be used from the inside your network and you test those rules on the test system to ensure that they are working, why not lock the door for all other ports? If there is an outbreak of a virus or Trojan program, then the virus or Trojan either needs to know which ports are open to establish a connection to the outside world or fails miserably and triggers an alarm that indicates the infected machine. Securing your system in this way is worth investigating.

7.5 SECURITY POLICY

A well-formulated and well-planned security policy, enforced companywide, is the best prerequisite to create a secure infrastructure. As simple as it may sound, it is the best approach to be taken toward a more secure IT environment. Many users put too much emphasis on "personal" in "Personal Computer." This does not mean that the owner of a computer should not be able to do what he wants with the system. However, things get a little more complicated if that system is part of a company network, and the user is not the "owner" of the system. No one would come up with the idea to change the software of a calculator or update the coffee machine with a new shiny GUI. The former might make the results of the calculator non-deterministic, and touching the coffee pot could trigger a revolution inside the IT department.

Seriously though, the main difference between computers and calculators or typewriters is the fact that computers can do much more than those accessories. But the way a number of software manufacturers include some kind of "self-healing" intelligence into their programs could change operating systems in a non-deterministic way by downloading updates from the manufacturer's site without asking the user or the IT department for permission. The resulting confusion, including the loss of valuable data, is blamed on the IT department afterward.

One of the primary causes of non-working computers has not even been mentioned until now: the installation of untested software, obtained from unknown sources, on the company-supplied computers. And a lack of stability regarding the systems is the least of all problems in that respect. The installation of untested software is the quickest way to introduce a virus, Trojan, or spyware into the company network. As funny as the latest PowerPoint presentations containing cartoons might be, from a security point of view, they can be nightmare.

A consistent security policy, taking the mentioned problems into account, is a step in the right direction for every company, provided there are not too many exceptions to the rules. To make one thing very clear, a security policy is not a tool to hassle the users; it is a tool to improve the security. A good sense of proportion allows a sound security policy without tormenting the users. You need rules governing the allowed and the forbidden actions concerning the computers. These rules need to be applied to the manufacturers of the operating systems and the programs as well. Patches and upgrades installed behind the back of the users and the IT department destabilize the infrastructure you, as system administrators, are responsible for.

A sound security policy should consist of different modules, and it may well be that a monolithic block of rules is not needed or wanted. The sane approach to the subject is the same as that used for building a good network infrastructure. You define what services should be available and create the interfaces. Next, you determine how to secure those services and what types of use are acceptable. Using this information as a base for a sound security policy is a good starting point. You should never try to tackle all issues at once:

- Policy for outside access to the company network
- Policy for using e-mail
- Policy for securing the place of work
- Policy for the installation of software on the workstations

This list could go on forever. You should start with topics that are already governed by some sort of rules and try to formulate them more precisely. The creation of new policies should be easier after tackling the existing topics, if you learn from experience.

7.5.1 Why a Good Security Policy Needs to be Platform Independent

The structure of a security policy should be formulated as platform independent as possible. For example, software, updates, and patches have to be cleared with the IT department prior to installation. If a simple policy like that gets watered down with exceptions for seemingly "secure" systems or platforms, then Pandora's box is already wide open, and arguments like "But the manufacturer says that the patch needs to be applied at once to increase security" are hard to conquer. You have to draw the line somewhere. Adhering strictly to the stated rules means more work for the IT department. On the other hand, if the IT department knows what it is doing and has a firm grip on the infrastructure, it should be possible for them to deal with a threat from the outside while the testing is finalized.

The goal to be platform independent holds equally true for the software suppliers. Including the manufacturer of a certain antivirus software into the policy will make that policy unnecessarily complicated. It would be better to create a list of requirements that can be used to measure the offers made by the manufacturers and suppliers. For example:

- The AV software has to be available on all workstations used in the company; failing to provide the same release for all used workstations terminates the agreement.
- The distribution of virus signatures has to take place from a system inside the company. The supplier has to ensure that the software allows you to disable the update from sources outside the company.
- The client software needs to be protected against tampering.
- The necessary checks for e-mail and attachments should take place on the e-mail server and should be repeated on the client system for security reasons.

Again, the list shown will not apply to every company and is to be considered only as an example. The author would add to the last point: "The AV software on the server and the AV software on the client should come from different manufacturers." Four eyes will see more than two. Including such detailed requirements in the policy increases the workload, but also enables the company to continually test the systems used, especially if you have made compromises based on the information about future developments promised by a manufacturer during the implementation.

If the security policy contains loopholes to make it fit to a certain product, then that security policy needs to have the loopholes reworked when you change the supplier. The policy should reflect the needs of the company and not the shortcomings of the supplier.

You can find good information pertaining policies at *http://www.sans.org/resources/policies.*

7.5.2 How to Calculate the Risk

You need to perform periodical risk assessments. This means you need to calculate the costs of a security breach for the company. It should contain questions like:

- Is it possible to determine which data was or could have been manipulated during the incident?
- How long will it take to establish the agreed upon service level if a server is compromised?
- Is it possible to run the services of the compromised server on another system while the original system is reinstalled?
- Are there employees trained in computer forensics or do you have to use external help? If you need external help, how long does it take for the external contractor to reach the company?
- Is it possible to recover from the lost consumer confidence if a security breach is publicized?

The last point is especially dire for companies doing business over the Internet. The financial impact of the loss in confidence cannot be calculated. These questions seem to be of a more abstract nature at first glance, because no one likes to think about these subjects. However, imagine the following scenario: a file server marketing and sales used to store their spreadsheets is compromised. How long will it take to make sure that no one changed the location of a point in all these numbers? If these numbers are used to create quotes for the customers, what will happen if a customer receives a ridiculous quote? If the spreadsheets contain e-mail addresses, will the customers like the idea that their addresses are sold to e-mail marketing companies? And you are not even to the worst outcomes yet: if the files contained the customers and conditions, what would happen if these numbers were publicized? Or worse, sold to the competition?

You need to have a “Plan B” for events like the ones described. This is not about scaring people; this is about the creation of a sound “Plan B” should the worst-case scenario manifest itself. If you know what you need to do, at least in theory, then you are a little less unprepared when dealing with the actual situation. Planning and calculating such scenarios costs time and money. Many companies try their best to avoid the expense. Even worse for some companies, a risk assessment has the potential to uncover flaws in the current planning. It is possible to minimize the uncovered risk, but even that comes at a price.

7.6 SUMMARY

This chapter discussed the various aspects of security; it started with general thoughts about security. It covered the three entities that need to be secured: the file, the system, and the network. It closed with a discussion about security policies and why they are important to the well being of the company. Another aspect of a sound security policy, the risk assessment, was covered as well.

Finally, thinking about worst case scenarios is not an undertaking to be entered into lightly. All required parties should be aware of the fact that dealing with those scenarios is frustrating at best. On the upside, a bit of creative thinking can lead to improvements that can be implemented right away, giving you the feeling that you have covered another base and made your environment a little bit more secure. This is a rewarding feeling; it helps you to see the future in a brighter light. Honestly, it is your job to improve the systems you are responsible for every day. You can make this a rewarding job.

7.7 REFERENCES

[Cheswick03] Cheswick, William, Bellovin, Steven and Rubin, Aviel. *Firewalls and Internet Security: Repelling the Wily Hacker, Second Edition*. Addison-Wesley, 2003.

8 That's It (Almost)

In This Chapter

- Ethics for System Administrators
- Closing Remarks

There are two more things to say: one is a serious matter, and the other is a good-bye to you.

8.1 ETHICS FOR SYSTEM ADMINISTRATORS

You have been granted far-reaching rights and options to read, write, and manipulate (in a positive sense) data on the systems to be administered. So, a huge responsibility comes with the job as system administrator. Your superiors show great trust in your integrity by granting you this kind of global access to the company's data. The "System Administrators Guild" (SAGE) publishes a "Code of Ethics" on their Web site (*http://www.sage.org/ethics.mm*). Take the time to read it. Systems administration is much more than just "keeping things running."

Frustration sometimes comes with the job description. That's just the way it is. If you should feel low, though, take a look at *http://www.theregister.co.uk/odds/bofh/*. Just keep in mind that this site is supposed to be satire and not a lecture in how to behave.

8.2 CLOSING REMARKS

The author has managed to hit most of the roadblocks regarding *NIX administration during the last 18+ years. You will probably find new ones. He has tried to give you hints and pointers to ease your daily work a bit. Trying to teach someone all the necessary things to be considered a *NIX guru from now on is, most likely, impossible. Perfect system administrators aren't born once in a blue moon, either. A great deal regarding systems administration is instinct and gut feeling. Both can't be taught in books; they can be learned, not trained. Try to gain as much knowledge as possible and take the time to experiment and test. Sometimes documenting the tests doesn't hurt, by the way. In the beginning, the author learned a lot through hints and pointers where to look and what to try. This book contains some of these hints and pointers; it is up to you to make use of them. The author hopes you will have as much fun as he had and still has while trying to find out how things work on a *NIX system.

At the beginning of this book, the author wrote that systems administration is a creative process; hopefully throughout the book you were convinced that this statement is true. And now for the most important hint in this book: *never* forget to have fun while working as a system administrator!

Appendix A One Liners

One liners are a collection of handy commands and programs, some longer than one line. The commands shown are extracts taken from the author's files containing interesting or handy uses of the commands. Some entries are in these files because the author cannot or will not memorize the syntax. Every system administrator should create files containing interesting programs and commands, giving the right wording in the explanation. The files are even searchable with `grep`.

"*NIX isn't about knowing every option in every command, but knowing where to look," one of the author's teachers used to say.

One other thing—some of the actions and examples found in the following sections can be executed via some sort of GUI. The beauty of knowing how to achieve the same results on the command line not only translates into a certain independence from those GUIs but also enables the system administrator to execute them from a PDA or cell phone with a vt100 emulation.

A.1 SHELL COMMANDS

The following commands should work more or less on every brand of *NIX.

A.1.1 Finding and Deleting Core Files

If a program on a *NIX system should crash—this can happen, even on *NIX—then the kernel creates a so-called core file. It contains an image of the kernel during the time of the crash. This information is very interesting for the developers of the program, but of limited use for a system administrator. The command shown in Listing A.1 searches the disk for these files and removes them after asking for permission.

LISTING A.1 Finding and Removing Core Files

```
root@unixhost> find . -name core -exec file {} \; -exec rm -i {} \;
```

The chain of commands shown starts to search from the current directory downward for files named `core`, shows which program created them, and asks for permission to delete the files.

A.1.2 Finding Directories Consuming Disk Space

If a volume starts to fill up, then it is interesting to find out which directories consume the most disk space. Listing A.2 shows how to display and sort the directories.

LISTING A.2 Display Directories by Size, Sorted

```
root@unixhost> du -sk * | sort -nr | more
1143704 usr
1033478 export
407870  proc
381438  var
17689   sbin
14846   platform
13566   kernel
4246    etc
1529    dev
1       opt
1       nsr
...
```

The command `du` uses the following options in Listing A.2: `-s` for summary, to sum up the underlying directories instead of displaying them one by one, and `-k` to use KB as size unit. The use of `-h` (nicknamed human readable) would render the result useless for sorting because `sort` would not know that a number with an appended "G" is bigger than number with an appended "M." The command `sort` uses the options `-n` to switch from alphabetical sorting to numerical sorting and `-r` to reverse the result. This means the highest numbers are the first to show up in the output. Whether or not the command `more` needs to be used is dependent on the anticipated number of directories in the output. If the result contains more lines than the maximum number of lines displayable in the terminal window, then the biggest consumers of disk space scroll off the top of the window. The directory names `/opt` and `/nsr` are symbolic links; they point to other locations on the same or another hard disk. Using the command as shown allows you to display a quick rundown of which directory uses the most space on the hard disk.

A.1.3 Who Is Accessing This File?

As long as a file or directory is accessed by a user or a process, that directory or file cannot be deleted, nor can the volume it is stored on be unmouted. So, one question often asked by system administrators is, "Who is using this file or directory at the moment?" Most *NIX systems provide the command `fuser` to answer this question (see Listing A.3). It allows you to identify the number of the process accessing the file or the directory at the moment. If the number of the locking process is known, then using a command like `ps` reveals the user account running said process.

LISTING A.3 Using `fuser`

```
# fuser admintrans
admintrans:      951c     938c
# ps -ef | grep 951 | grep -v grep
 silvied   951   938  0 12:13:49 pts/5    0:00 vi monthlyreport.txt
# ps -ef | grep 938 | grep -v grep
 silvied   938   929  0 12:13:13 pts/5    0:00 /bin/sh
 silvied   951   938  0 12:13:49 pts/5    0:00 vi monthlyreport.txt
```

Two processes are accessing the directory `admintrans` in Listing A.3: `951` and `938`. The letter `c` means that `admintrans` is also the current directory (`$PWD`) of these processes. If you need to find out who owns theses processes, you can use the command `ps` filtered with `grep`. It becomes obvious that the user account `silvied` used `vi` to access the file called `monthlyreport.txt` and that the process was started from `/bin/sh`. Listing A.4 demonstrates the use of `fuser` in a device context.

LISTING A.4 Using `fuser` with a Device

```
root@unixhost> df -h
Filesystem           Size  Used Avail Use% Mounted on
/dev/dsk/c0t3d0s0     1.5G  1.2G  279M  81% /
/dev/dsk/c0t3d0s1     482M  373M   61M  86% /var
swap                  533M   32K  533M   1% /var/run
swap                  533M  312K  533M   1% /tmp
/dev/dsk/c0t3d0s7     1.5G 1010M  404M  72% /export/home
root@unixhost> fuser /dev/dsk/c0t3d0s7
/dev/dsk/c0t3d0s7:     1047c    1002c    1001c     951co
...
```

The use of the command `fuser` allows a system administrator to find out who is using a file or directory at the moment or which account currently holds a lock on a certain device. The command is very versatile; you should study the program's manpage to learn more about it.

A.1.4 How to Truncate a Log File?

Sometimes you need to truncate a file. This means to empty the file without deleting it from the hard disk. A possible reason for this requirement might be the fact that a program writing to the file cannot be terminated and deleting the file would interrupt the program. That program would lose the link to the file if it gets deleted and would thus stop writing to it. You can use the command shown in Listing A.5 to truncate a file.

LISTING A.5 Truncate a File

```
root@unixhost> cat /dev/null > /var/log/authlog
```

You should be aware of the fact that the contents of the file are lost, though. If you copy the contents into another file first, then the command could be used to manage the size of log files, for example. You just need to make sure that the command to copy the contents and the command to truncate the file are executed in close succession; you could lose valuable data otherwise.

A.1.5 How to Find the Libraries a Program Requires?

If you should need to know which libraries a program needs to work, to copy that program to another system, for example, or in case you receive an error message about missing libraries, you could use the command `ldd` to determine what is required. Listing A.6 demonstrates the use.

LISTING A.6 Identifying the Required Libraries with `ldd`

```
root@unixhost> ldd /usr/sbin/fuser
        libkstat.so.1 =>        /usr/lib/libkstat.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libdl.so.1 =>    /usr/lib/libdl.so.1
        /usr/platform/SUNW,Ultra-1/lib/libc_psr.so.1
```

Listing A.6 lists all required libraries used by the program `fuser`.

A.1.6 How to Change the Case in Filenames

If you should need to change the names of files or directories to be all lowercase or all uppercase, for whatever reason, then the command shown in Listing A.7 could be used.

LISTING A.7 Changing All Names to Lowercase

```
erikk@unixhost:> for i in `ls`; do mv $i `echo $i | tr '[A-Z]'
'[a-z]'`; done
```

The `for` loop uses the command `ls` to get a list of all the names, be it a file or directory. The output of `ls` is fed into `i` with the help of the backticks. For every name found, the command `mv` is used to change all characters to lowercase by using the output of `echo` fed into `tr` (translate) for changing the case of the letters A to Z. A file named `ThisTExt.txt` will be changed to `thistext.txt`. The regular expression `'[A-Z]' '[a-z]'` takes care of all uppercase characters and transforms them into lowercase.

A.2 SOLARIS SPECIFIC

This section contains two hints for Solaris administrators.

A.2.1 `prstat`

A program named `prstat` is part of all newer Solaris distributions. Its output is similar to `top`, but it cannot be used in an interactive way like `top`, unfortunately. If you should have no means to install `top` on a Solaris machine, then you could use `prstat` instead. The program is mentioned here because not too many Solaris users are aware that the program even exists.

A.2.2 Registering Solaris Systems

One remark regarding this hint: *you should register your Solaris systems with Sun Microsystems.* But if you have to roll out a considerable number of systems, then registering every single system while installing the operation system is time consuming at best. You should register all systems in one go with the manufacturer instead. Listing A.8 shows how to switch off the registration program.

LISTING A.8 Disabling the Registration in Solaris

```
root@unixhost> cat DISABLE=1 >> /etc/default/solregis
```

Again, *the systems should be registered*; the command in Listing A.8 shows how to avoid doing this on every single system during a rollout of 200+ systems, for example.

A.3 THE sed STREAM EDITOR

An often neglected tool on every *NIX system is the sed stream editor. Its capabilities are not that obvious and sometimes well hidden. The program is meant to process text files by using the file as a stream and to output the modified file to stdout again. The size of the file is more ore less irrelevant because the output can be sent to a file again. This means there are almost no restrictions regarding the size of the files the program can process.

A.3.1 Creating Blank Lines after Search Results

If you should need to edit an XML file with an editor, then depending on the size and complexity, the process could be very cumbersome and error prone. Assume you want to edit a file structured in DocBook format (this means the hierarchy of sections consists of nested <sect1>, <sect2>, <sect3>, and <sect4> tags), and you want to ease the navigation inside the file by inserting a blank line above every <sect1> tag. You could use an XSLT stylesheet to create this kind of formatting, but the result does not justify the time needed to create the stylesheet. Listing A.9 demonstrates the use of sed to achieve the same result.

LISTING A.9 Insert a Blank Line Using sed

```
...
 but there will be times when proper documentation really
 saves your day.</para>
<sect1 id="environment">
<title>Environment</title>
<para>There are a lot of settings concerning the environment
 variables on a system, which likely will give you a headache,
 to say the least.</para>
<sect2 id="env_path">
<title>The <userinput>$PATH</userinput> variable</title>
<para>If the <envar>$PATH</envar> variable looks like
 this:</para>
<screen>
...
erikk@unixhost> sed '/<sect1/{x;p;x}' exchap1.xml
...
```

```
 but there will be times when proper documentation really
 saves your day.</para>

<sect1 id="environment">
<title>Environment</title>
<para>There are a lot of settings concerning the environment
 variables on a system, which likely will give you a headache,
 to say the least.</para>
<sect2 id="env_path">
<title>The <userinput>$PATH</userinput> variable</title>
<para>If the <envar>$PATH</envar> variable looks like
 this:</para>
<screen>
...
```

In Listing A.9, `sed` is instructed to search for lines containing `<sect1`. If the string is found, then `sed` outputs a blank line and the line found afterwards. Because the processing is performed on the files as a stream, the size of the file is of no concern to you. If the program you use to process the XML file later should not be able to cope with the additional blank lines, then you could use the command `sed '/^$/d'` to erase those blank lines. You just need to be aware of the fact that this command will delete *all* blank lines in the stream.

A.3.2 Removing Leading Blanks and Tabs

If you should receive a text file that contains leading blanks or Tab characters and you need to get rid of them, `sed` can be used to do so. Listing A.10 demonstrates how. Please note that the character string between the "`[]`" characters needs to be typed in as space, Ctrl-V, Tab.

LISTING A.10 Removing Leading Whitespace

```
erikk@unixhost> cat leadwhite.txt
This is line 1.
   This is line 2.
                This is line 3.
This is line 4.
        This is line 5.
This is line 6.

erikk@unixhost> sed 's/^[    ]*//' leadwhite.txt
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
This is line 6.
```

The instruction to `sed` reads as follows: substitute (`s`) at the beginning of the line (`^`) blank and Tab, regardless of how many there are (`*`) with nothing (`//`). To achieve the same result at the end of a line, you could use the instruction `sed 's/[    ]*$//'`.

Another use of this functionality is the removal of whitespace from certain lines; this means you want them removed only if a certain condition is true. Listing A.11 shows an example of how to do this.

LISTING A.11 Conditional Removal of Whitespace

```
erikk@unixhost> cat sedblockaddr.txt
<book>
    <chapter>
        <title>Whitespace inside tags</title>
        <para>This text contains tabs and spaces to make it more
        readable by clarifying the structure through indentation.
        There are programs that can't cope with the whitespace
        inside the paragraph tag.</para>
        <para>The DocBook XSLT is able to remove the additional
        whitespace, but finding how they did it can take a
        considerable amount of time.</para>
    </chapter>
</book>

erikk@unixhost> sed '/<para>/,/<\/para>/ {s/^[  ]*//}' sedblockaddr.txt
<book>
    <chapter>
        <title>Whitespace inside tags</title>
<para>This text contains tabs and spaces to make it more
readable by clarifying the structure through indentation.
There are programs that can't cope with the whitespace
inside the paragraph tag.</para>
<para>The DocBook XSLT is able to remove the additional
whitespace, but finding how they did it can take a
considerable amount of time.</para>
    </chapter>
</book>
```

This time the example instructed `sed` to process only lines situated between `<para>` and `</para>`. The instruction used resembles some sort of loop, activated only if the string `<para>` is found and deactivated again if the string `</para>` appears. The Tab character was input with Ctrl-V and Tab, as usual. The same result could be achieved with an XSLT stylesheet, but the header alone you would need to use in that stylesheet would be longer than the `sed` command used.

A.3.3 Inserting Characters at the Beginning of a Line

If you want to insert characters at the beginning of a line, then you need to "replace" the beginning of the line with the desired characters, as shown in Listing A.12.

LISTING A.12 Inserting Characters at the Beginning of the Line

```
erikk@unixhost> cat leadwhite2.txt
This is line 1.
This is line 2.
This is line 3.

erikk@unixhost> sed 's/^/xxxxx/' leadwhite2.txt
xxxxxThis is line 1.
xxxxxThis is line 2.
xxxxxThis is line 3.
```

The string "`xxxxx`" can be replaced by blanks or Tab characters as needed, obviously.

A.4 THE VI EDITOR

The `vi` editor is a powerful program, but it is rarely used to its full potential because of the lack of a user interface.

A.4.1 Cursor Movement

If you need to move the cursor inside a file you opened previously with `vi`, then you could use the cursor keys. The downside here is as follows: the size of the largest document to be edited with `vi` is limited only by the available space in the `/tmp` directory. This means going from line 1 to line 4078 will take a considerable time if the movement is done by using the cursor keys.

You could use the `G` command in `vi`'s command mode to speed things up a little. If you use `4078G` in command mode, `vi` will jump to line 4078 from every location in the document. If you want to go to line 1, `1G` is used, and to go to the end of the document, `G` alone initiates the jump. If you want to jump to the beginning of the current line, you could use `0` instead of the cursor keys and `$` to jump to the end of the current line.

A.4.2 Delete and Change

The `vi` editor interprets `w` (*word*) as extension of a command. If you input `dw` (*delete word*), `vi` will delete a word starting from the current position of the cursor until it reaches the next blank or Tab character. If you would use `d$`, the contents of the current line starting again from the position of the cursor to the end of the line will be erased. `d0` does the same but to the beginning of the line. Therefore, you can even use the jump commands to delete parts of a document. The command `dG` deletes everything from the current position of the cursor to the end of the document, and `d1G` deletes everything from the current position of the cursor to the beginning of the document. If you want simply to delete the current line, you could use the command `dd`. This command can be extended by using a number; `5dd` deletes the current line and the next four lines, if they exist.

If you just need to change a word, you could use the command `c` (*change*). You need to position the cursor on the first character of the word or at the position inside the word where the change should start and use the command `cw` (*change word*). The editor will switch to input mode, and you will be able to overwrite the word starting from the current position of the cursor up to the next blank or other delimiter recognized by `vi`. If you want to overwrite the current line starting from the cursor position to the end, you could use `c$`, and if you want to overwrite the current line starting from the cursor position to the beginning of the line, you could use `c0`. Pressing the Esc key ends the replacement mode.

Because all the mentioned commands are destructive, you should always remember that you are able to press `u` (*undo*) if you should change your mind after the replacement. The undo function works on almost every command of `vi` that changes the document.

A.4.3 Global "Search and Replace"

One thing needs a more exhaustive explanation when using `vi` and following the logic of the program. The command `s/old_word/new_word/g` substitutes `old_word` with `new_word` in the current line only and not in the whole document as the `g` (global) switch seems to indicate. To do a real global "search and replace," you need to use the command shown in Listing A.13.

LISTING A.13 "Search and Replace" within the Whole Document

```
%s/old_word/new_word/g
```

The `%` character at the beginning of the command does the trick.

A.4.4 Using Abbreviations

The `vi` editor was created by programmers *for* programmers. It contains some features to speed up the insertion of text blocks due to its heritage. One of the features is the use of so-called *abbreviations*. If you start `vi` and type the string in Listing A.14 without changing into insert mode, you create such an abbreviation. If you are done creating the abbreviation, you close the input by hitting the Enter key.

LISTING A.14 Definition of an Abbreviation in `vi`

```
:ab FROBOZZ FroBozz: We make programming less work more pleasure
```

If you would switch to input mode via `i` now and would type the string `FROBOZZ` and a blank, `vi` would replace that string with the string `FroBozz: We make programming less work more pleasure`. You can create shortcuts for phrases and sentences you need to use more than a couple of times in a document. The system also works for programming constructs or XML tags: `:ab PARA <para></para>` will replace all occurrences of `PARA` with `<para></para>`. The downside of this system is the fact that you will not be able to use the string `PARA` in the document anymore. You need to exercise care when choosing the handle for the abbreviation for that reason and should not use words or acronyms you might actually need to write in the document.

This poses the question of how this system could save time—if you have to retype the abbreviations every time `vi` is started again. The `vi` editor provides two ways to make the abbreviations available without the need to type them in repeatedly. The program searches for a file named `.exrc` in the `$HOME` directory of the current user, or you load such a file by using the command `:so <filename>` after launching the program. Listing A.15 shows how such a file could look.

LISTING A.15 Abbreviations for Perl

```
"
"    Perl abbrs for vi
"
:ab WLOOP while(<>)^M{^M^M}
:ab FORI for(i=0; i <= ; i++)^M{^M^M}
```

If you would load the file shown in Listing A.15 with the command `:so <filename>`, then you would be able to create a *while* loop by inputting the string `WLOOP`. The same goes for `FORI`; this abbreviation creates a *for* loop with the variable `i` as counter. To include the Enter key in the abbreviation, you have to input the key combination Ctrl-v followed by the Enter key. Using this system is a matter of taste, though. If you have the feeling the system speeds things up considerably, then you should use it or at least give it a try.

A.4.5 Joining Lines

If you need to join two consecutive lines in `vi`, you could use the command `J`. You would have to position the cursor on the first of the two lines to be joined and press `J` to join them into one single line. Listing A.16 shows an example.

LISTING A.16 Joining Lines in `vi`

```
This is a line
split in two.

Another line.
```

If you would position the cursor in the first line and type `J` in command mode, the two lines would look like so:

```
This is a line split in two.

Another line.
```

This command seems to answer one of the more frequently asked questions regarding `vi`.

A.4.6 A Short Table with Important `vi` Commands

To close the hints regarding `vi`, a short—and by no means complete—table (Table A.1) of useful commands follows for easy reference.

TABLE A.1 Important `vi` Commands

Command	Description
`vi <filename>`	Opens a file with `vi`; if the file does not exist, it is created.
`CTRL+f`	Jumps one screen forward.
`CTRL+b`	Jumps one screen backward.
`a`	Appends to the right of the cursor.
`i`	Inserts at the current position of the cursor.
`o`	Inserts a new line below the current and switches to input mode.
`O`	Inserts a new line above the current and switches to input mode.
`cw`	Changes the current word starting from the cursor position; Esc to end.
`J`	Joins the current line and the one below.
`u`	Undo, reverses the last change.
`x`	Deletes the character at the current position of the cursor.
`dw`	Deletes the word at the current position of the cursor, starting with the cursor position.
`dd`	Deletes the current line.
`dG`	Deletes from the current position of the cursor to the end of the document.
`d1G`	Deletes from the current position of the cursor to the beginning of the document.
`:7,18d`	Deletes lines 7 to 18.
`yy`	Copies (yanks) the current line.
`p`	Inserts the last copied or deleted line below the current line.
`1G`	Jumps to line 1 of the document.
`G`	Jumps to the last line of the document.
`17G`	Jumps to line 17 of the document.
`/<string>`	Finds the next occurrence of `<string>`.
`:r <filename>`	Loads (reads) the contents of `<filename>` and inserts them starting with the current position of the cursor.
`:w`	Writes the current document to the disk.

→

`:w <filename>`	Writes the current document to the disk and names it `<filename>`.
`:wq`	Saves the current document and quits `vi`.
`:q!`	Terminates `vi` without saving the changes since the last `:w` command.

A.5 CALENDARS

The program `cal` is part of every *NIX system, but it is rarely used. The program `gcal` is a replacement for `cal` published under the GNU License. The latter can be obtained for almost every *NIX system (see Appendix B for download addresses).

A.5.1 `cal`

The program `cal`, called without any arguments, displays the current month. If called with a month and a year, it displays the indicated month in the indicated year, as shown in Listing A.17.

LISTING A.17 Using `cal`

```
erikk@unixhost> cal 5 2005
   May 2005
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

If you would start the program with the `-y` parameter, you would get a calendar for the current year.

A.5.2 `gcal`

The program `gcal` by Thomas Esken acts like `cal` in the previous section. The program comes with a lot more capabilities, though. If you would call it with the `-K` parameter, for example, it would display the week numbers, see Listing A.18.

LISTING A.18 Using `gcal`

```
erikk@unixhost> gcal -K 5 2005

       May 2005
 Su Mo Tu We Th Fr Sa CW
  1  2  3  4  5  6  7 18
  8  9 10 11 12 13 14 19
 15 16 17 18 19 20 21 20
 22 23 24 25 26 27 28 21
 29 30 31             22
```

If you should want to schedule a phone conference between the office in Boston, Massachusetts, and Tokyo, Japan, you could use the command shown in Listing A.19.

LISTING A.19 Using `gcal` to Schedule a Conference

```
gcal -n -qus_ma+jp 7 2005

      July 2005
 Su Mo Tu We Th Fr Sa
                 1  2
  3  4  5  6  7  8  9
 10 11 12 13 14 15 16
 17 18 19 20 21 22 23
 24 25 26 27 28 29 30
 31

Eternal holiday list:

Independence Day (US_MA)                + Mon, Jul   4th 2005 =  +91
days
Star Festival (JP)                      - Thu, Jul   7th 2005 =  +94
days
Marine Day (JP)                         + Wed, Jul  20th 2005 = +107
days
Parent's Day (US_MA)                    - Sun, Jul  24th 2005 = +111
days
```

The + sign in front of the 4th of July and the 20th of July means that these are non-working days in the respective countries. The remaining days shown in the “Eternal holiday list” are days that might be not a good choice for one reason or the other. The -n option instructs the program to display the list of special days, and -qus_ma+jp means to include holidays and special occasions in Massachusetts in the U.S. and in Japan. This example really just scratches the surface regarding the capabilities of this program. It is possible, among other things, to use the Chinese or Japanese calendar and to display the phases of the moon and other astronomical events (precise enough to be used as information, *not* for rocket launches, though), and the program is able to calculate the holidays for most of the existing countries. If you need to deal with information like this, you should certainly take the time to get acquainted with gcal.

A.6 ORACLE

The following commands are not "real" *NIX commands, but because many system administrators have to deal with the Oracle Relational Database Management System (RDBMS), here they are.

A.6.1 Adding a Data File to Tablespace

If you need to add a new *data file* to a *tablespace* with low capacity, then you could use the command in Listing A.20 in sqlplus or a similar program.

LISTING A.20 Adding a Data File

```
alter tablespace <tablespacename> add datafile
    <path_to_datafile/datafilename> size 250M;
```

You could add the file datafilename in the indicated path to the tablespace. The file would have a size of 250 MB. Depending on the version of Oracle, you could have used k (kilobyte), m (megabyte), or g (gigabyte) to indicate the desired size. You have to make sure, though, that there is enough room on the volume.

A.6.2 Displaying Data Files Belonging to Tablespace

If you need a list of all data files belonging to a certain tablespace, you could issue the command in Listing A.21 in sqlplus or a similar program.

LISTING A.21 Listing the Data Files Belonging to a Certain Tablespace

```
select t.name, d.name from v$tablespace t, v$datafile d
    where t.ts# = d.ts#;
```

A.6.3 Moving or Renaming a Data File

If you have to move or rename a certain data file, you must follow the order of the steps shown in Listing A.22. If you need to do this on a very busy database, you should make sure to execute all commands quickly because all write operations will go into a temporary buffer during the move or the rename. The best way of doing this would be to either perform this action during off-hours or ask the users to suspend all database operations for the time needed. Failing to follow the steps as shown will almost certainly result in a corrupt database.

LISTING A.22 Moving or Renaming a Data File

```
alter tablespace <tablespacename> offline normal;
alter tablespace <tablespacename> rename datafile
        <oldlocation/oldname> to <newlocation/newname>;
---- rename/move datafile with OS commands
alter tablespace <tablespacename> online;
```

If you have never done this before, please use a database for testing to get a feeling for the results and the duration of this operation.

A.6.4 Forcing a Log Switch

This entry is in this book because the author cannot seem to remember that a log switch is a *system* item and not a *database* item. Listing A.23 shows the command.

LISTING A.23 Forcing a Log Switch

```
alter system switch logfile;
```

This command obviously requires a database with `ARCHIVELOG` enabled. The command `select * from v$log;` enables you to display the current status before and after.

This page intentionally left blank

Appendix

B Software Used in This Book and Where to Get It

There are many ways to get hold of the open source software mentioned in this book. The most efficient way is to get the source, compile it, test it thoroughly, and deploy it. Sadly, this approach to deployment is normally out of the question for system administrators because of time constraints. So the other option, getting precompiled binaries, is the way to go. Fortunately, most of the distributors/manufacturers of your brand of *NIX have already created or are supporting repositories containing more or less everything that is needed. That the packages on these sites are usually tested is an added bonus. One word of caution, though: everyone needs to make sure to use the provided way of testing the integrity of the downloaded packages, often by using md5 checksums.

There is one thing you should keep in mind—the packages found on the following sites are usually maintained on a voluntary basis. Most of the people involved there owe you nothing; you, on the other hand, owe them big time. Therefore, you should try to help them save bandwidth by using mirrors if available and be patient if the latest version of the package you want to install is not available yet. There might be a reason.

B.1 SOFTWARE FOR SOLARIS

Open source software for Solaris comes in two ways. Sun supplies you with a solid foundation via their *companion CD-ROM*. Included is the installation media, and if you should miss something, you can always head to *http://www.sunfreeware.com* and get it from there. The packages originating from the CD-ROM will be installed in `/opt/sfw`; the packages you download from *http://www.sunfreeware.com* will end up in `/usr/local`. The downside here is obvious: the packages in one directory tree have no way of knowing about the libraries in the other. To rectify this situation, you should simply make `/usr/local` a symbolic link to `/opt/sfw`. This way, you get the best of both worlds: packages supplied on the CD-ROM have access to the ones downloaded from *http://www.sunfreeware.com,* and replacing the supplied packages with newer versions from the Web site simply requires the installation of the newer version.

B.2 SOFTWARE FOR AIX

Open source software for AIX can be found on *http://www.bullfreeware.com*—Bull's Large Open Source Software Archive for AIX—and *http://aixpdslib.seas.ucla.edu*—the UCLA Public Domain Software Library for AIX. Installing the software is usually a question of deflating the archive and running `smit` in case of the usual `.exe` files (these files are the self-extracting archives).

B.3 SOFTWARE FOR HP-UX

Open source software for HP-UX can be found at *http://hpux.cs.utah.edu*—The HP-UX Porting and Archive Center. This archive is mirrored all over the world, so you should use a mirror server closest to your location. Please make sure to use *SAM* to install the packages; otherwise, the system won't be able to use them.

B.4 SOFTWARE FOR IRIX

SGI® distributes open source software for IRIX® at *http://freeware.sgi.com*—SGI IRIX Freeware—so you get the packages directly from the mothership, where your hardware originated. The downloaded packages can be installed via `swmgr`.

Appendix

C Internet

Because the Internet is always changing, there is a slight possibility that some of the links listed in Table C.1 might cease to exist. Nevertheless, what follows is a list of links, in no particular order, the author uses on a day-to-day basis.

TABLE C.1 Useful Internet Links

URL	Remarks
http://www.sage.org	The System Administrators Guild
http://www.usenix.org	The Advanced Computing Systems Association
http://www.sun.com/bigadmin/	The top admin portal for all Solaris users
http://isc.sans.org	The Internet Storm Center at *sans.org*
http://slashdot.org	The place to be for news about IT and science in general. Their motto is "news for nerds, stuff that matters." Expect to find lively discussions about the incoming news, not always polite in wording, but very interesting. Some of the people who wrote the operating systems covered in this book are present as readers and sometimes even raise their voices to give their opinion on a certain subject.
http://www.securityfocus.com	Many interesting articles about security; the level spans from beginner to pro.
http://www.macosxhints.com	Hints and tips for the users of Apple's BSD-based operating system.
http://www.iana.org/assignments/ipv4-address-space	Information on who maintains which IPv4 address range in the Internet.
http://www.iana.org/assignments/port-numbers	Well-known (0–1023) and registered ports (1024–49151)

This page intentionally left blank

Appendix

D About the CD-ROM

The CD-ROM contains the more elaborate scripts shown in the book. Thus, they do not need to be typed. Although, keep in mind that it is sometimes easier to understand a script when you type it instead of just copying the contents to the disk.

OVERALL SYSTEM REQUIREMENTS

Perl version 5.x or greater (the scripts should run on earlier versions but were not tested on them).

Java Runtime Environment (JRE) 1.3 or newer for the DocBook example.

The scripts should run on any more or less current *NIX-based operating system because they use functions and commands that are used by the system itself. In short: if *NIX runs on a system, so will the scripts.

The following sections contain lists of the files to be found in the corresponding directory on the CD-ROM.

FIGURES

All of the figures from the book.

CHAPTER 2 (*CH02*)

ssg_scripts/ch02/crontemplate.txt
ssg_scripts/ch02/df_check.sh
ssg_scripts/ch02/perlgetopt.pl
ssg_scripts/ch02/ulist1.pl

CHAPTER 3 (*CH03*)

ssg_scripts/ch03/baselines
ssg_scripts/ch03/baselines/base_checklog
ssg_scripts/ch03/baselines/base_checklog/chkauth.pl
ssg_scripts/ch03/baselines/base_cpu
ssg_scripts/ch03/baselines/base_cpu/base_cpu.pl
ssg_scripts/ch03/baselines/base_cpu/base_runq.pl
ssg_scripts/ch03/baselines/base_disk
ssg_scripts/ch03/baselines/base_disk/base_df.pl
ssg_scripts/ch03/baselines/base_disk/dirwatch.pl
ssg_scripts/ch03/baselines/base_netconn
ssg_scripts/ch03/baselines/base_netconn/base_netconn.pl
ssg_scripts/ch03/baselines/base_netconn/base_syn.pl
ssg_scripts/ch03/baselines/base_netconn/base_timewait.pl
ssg_scripts/ch03/baselines/base_netconn/linux_base_retrans.pl
ssg_scripts/ch03/baselines/base_netconn/run_scripts.sh
ssg_scripts/ch03/baselines/base_netconn/solaris_base_retrans.pl
ssg_scripts/ch03/baselines/housekeeping
ssg_scripts/ch03/baselines/housekeeping/logrotate.sh
ssg_scripts/ch03/baselines/housekeeping/logrotatedate.sh
ssg_scripts/ch03/baselines/html
ssg_scripts/ch03/baselines/html/baselining.css
ssg_scripts/ch03/baselines/html/images
ssg_scripts/ch03/baselines/html/index.html
ssg_scripts/ch03/baselines/plotcmds
ssg_scripts/ch03/baselines/plotcmds/base_cpu.plot
ssg_scripts/ch03/baselines/plotcmds/base_netconns.plot
ssg_scripts/ch03/baselines/plotcmds/graph_base_netconns.pl
ssg_scripts/ch03/baselines/plotcmds/graphcreate.sh
ssg_scripts/ch03/baselines/var

CHAPTER 4 (*CH04*)

ssg_scripts/ch04/backup_conf.sh

CHAPTER 5 (*CH05*)

ssg_scripts/ch05/chkauth.pl
ssg_scripts/ch05/loadtest.pl
ssg_scripts/ch05/newsysdoc.txt

CHAPTER 6 (*CH06*)

ssg_scripts/ch06/conv_tab.bc
ssg_scripts/ch06/DocBook
ssg_scripts/ch06/DocBook/getcommands.xslt
ssg_scripts/ch06/DocBook/Makefile
ssg_scripts/ch06/DocBook/maketest.sh
ssg_scripts/ch06/DocBook/testarticle.xml
ssg_scripts/ch06/DocBook/testbook.xml
ssg_scripts/ch06/RSS
ssg_scripts/ch06/RSS/articles
ssg_scripts/ch06/RSS/articles/backchange.rss
ssg_scripts/ch06/RSS/articles/cpustats.rss
ssg_scripts/ch06/RSS/articles/newcpucluster.rss
ssg_scripts/ch06/RSS/articles/newraid.rss
ssg_scripts/ch06/RSS/articles/rssmand1.rss
ssg_scripts/ch06/RSS/demo.rss
ssg_scripts/ch06/RSS/genrss1.sh

CHAPTER 7 (*CH07*)

ssg_scripts/ch07/dirlister1.sh

Please note that there are some empty folders on the disc as a result of the way the script structure is set up. This is intentional.

This page intentionally left blank

Index