Contents

< Day Day Up > NEXT

BSD Hacks

By Dru Lavigne

Publisher: O'Reilly
Pub Date: May 2004
ISBN: 0-596-00679-9
Pages: 300

Looking for a unique set of practical tips, tricks, and tools for administrators and power users of BSD systems? From hacks to customize the user environment to networking, securing the system, and optimization, BSD Hacks takes a creative approach to saving time and accomplishing more with fewer resources. If you want more than the average BSD user--to explore and experiment, unearth shortcuts, create useful tools--this book is a must-have.

Table of Contents
Index
Reviews
Reader Reviews
Errata
Academic

BSD Hacks

By Dru Lavigne

Publisher: O'Reilly
Pub Date: May 2004
ISBN: 0-596-00679-9
Pages: 300

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Credits

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

About the Author

Dru Lavigne is the author of ONLamp.com's FreeBSD Basics column and has been an avid BSD user since FreeBSD 2.2.1. As an IT instructor, she specializes in networking, routing, and security. She is also responsible for ISECOM's Protocol Database, which can be found at http://www.isecom.org.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Contributors

The following people contributed their hacks, writing, and inspiration to this book:

-
- John Richard, known locally as JR, is a system administrator in Kingston, Ontario, Canada. His trademark in the field is his insistence on a FreeBSD box as the primary firewall on a network. He has enjoyed working with the author in the past at a private college in Kingston. In his spare time, he experiments with FreeBSD and rides his Harley-Davidson.
- [Hack #64]
-
- Joe Warner is a Technical Analyst for Siemens Medical Solutions Health Services Corporation and has been using FreeBSD as a server and desktop since October of 2000. Joe has lived in Salt Lake City, Utah for most of his life and enjoys *BSD, computing, history, and The Matrix.
- [Hacks #35 and #59]
-
- Dan Langille (http://www.langille.org/) runs a consulting group in Ottawa, Canada. He has fond memories of his years in New Zealand, where the climate is much more conducive to year-round mountain biking. He lives in a house ruled by felines.
- [Hack #41]
-
- Robert Bernier's professional career has included engineering, accident investigation, and Olympic trials. In the 1980s, his interest returned to IT when he realized he wouldn't have to use a punch card anymore. Eventually he discovered Linux and by the mid-1990s had developed a passion for all things open source. Today, Robert teaches at the local community college and writes for a number of IT publications based in North America and Europe.
- [Hack #12]
-
- Kirk Russell (kirk@qnx.com) is a kernel tester at QNX Software Systems (http://www.qnx.com/).
- [Hack #36]

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Acknowledgments

I would like to thank the many BSD and open source users who so willingly shared their experiences, ideas, and support. You serve as a constant reminder that BSD is more than an operating system—it is a community.

I would also like to thank all of my students and the readers of the FreeBSD Basics column. Your questions and feedback fuel my curiosity; may this book return that favor.

Thanks to David Lents and Rob Flickenger for reviews and advice. Special thanks to Jacek Artymiak for his invaluable input from the OpenBSD and NetBSD perspectives. And finally, special thanks to chromatic. A writer couldn't have asked for a better editor.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Preface

"What was it about UNIX that won my heart? . . . UNIX is mysterious when you first approach. A little intimidating, too. But despite an unadorned and often plain presentation, the discerning suitor can tell there's lot going on under the surface."

—*Thomas Scoville,* http://unix.oreilly.com/news/unix_love_0299.html

When the above-mentioned article was first published, I was still very much a BSD newbie. My spare hours were spent struggling with kernel recompiles, PPP connectivity (or lack thereof), rm and chmod disasters, and reading and rereading every bit of the then available documentation. Yet, that article gave voice to my experience, for, like the quoted author, I had stumbled upon operating system love. In other words, I was discovering how to hack on BSD.

Since then, I've learned that there is an unspoken commonality between the novice Unix user and the seasoned guru. It doesn't matter whether you've just survived your first successful installation or you've just executed a complex script that will save your company time and money, the feeling is the same. It's the excitement of venturing into unknown territory and discovering something new and wonderful. It's that sense of accomplishment that comes with figuring something out for yourself, with finding your own solution to the problem at hand.

This book contains 100 hacks written by users who love hacking with BSD. You'll find hacks suited to both the novice user and the seasoned veteran, as well as everyone in between. Read them in any order that suits your purpose, but keep the "onion principle" in mind. While each hack does present at least one practical solution to a problem, that's just the outer layer. Use your imagination to peel away deeper layers, exposing new solutions as you do so.

Why BSD Hacks?

The term hacking has an unfortunate reputation in the popular press, where it often refers to someone who breaks into systems or wreaks havoc with computers. Among enthusiasts, on the other hand, the term hack refers to a "quick-n-dirty" solution to a problem or a clever way to do something. The term hacker is very much a compliment, praising someone for being creative and having the technical chops to get things done. O'Reilly's Hacks series is an attempt to reclaim the word, document the ways people are hacking (in a good way), and pass the hacker ethic of creative participation on to a new generation of hackers. Seeing how others approach systems and problems is often the quickest way to learn about a new technology.

BSD Hacks is all about making the most of your BSD system. The BSDs of today have a proud lineage, tracing back to some of the original hackers—people who built Unix and the Internet as we know it today. As you'd expect, they faced many problems and solved problems both quickly and elegantly. We've collected some of that wisdom, both classic and modern, about using the command line, securing systems, keeping track of your files, making backups, and, most importantly, how to become your own BSD guru along the way.

How to Use this Book

One of the beauties of Unix is that you can be very productive with surprisingly little knowledge. Even better, each new trick you learn can shave minutes off of your day. We've arranged the chapters in this book by subject area, not by any suggested order of learning. Skip around to what interests you most or solves your current problem. If the current hack depends on information in another hack, we'll include a link for you to follow.

Furthermore, the "See Also" sections at the end of individual hacks often include references such as man fortune. These refer to the manual pages installed on your machine. If you're not familiar with these manpages, start with [Hack #89] .

How This Book Is Organized

To master BSD, you'll have to understand several topics. We've arranged the hacks loosely into chapters. They are:

Chapter 1 Customizing the User Environment

Though modern BSDs have myriad graphical applications and utilities, the combined wisdom of 35 years of command-line programs is just a shell away. This chapter demonstrates how to make the most of the command line, customizing it to your needs and preferences.

Chapter 2 Dealing with Files and Filesystems

What good is knowing Unix commands if you have no files? You have to slice, dice, and store data somewhere. This chapter explains techniques for finding and processing information, whether it's on your machine or on a server elsewhere.

Chapter 3 The Boot and Login Environments

The best-laid security plans of administrators often go out the window when users enter the picture. Keeping the bad guys off of sensitive machines requires a two-pronged approach: protecting normal user accounts through good password policies and protecting the boxes physically. This chapter explores several options for customizing and securing the boot and login processes.

Chapter 4 Backing Up

After you start creating files, you're bound to run across data you can't afford to lose. That's where backups come in. This chapter offers several ideas for various methods of ensuring that your precious data will persist in the face of tragedy.

Chapter 5 Networking Hacks

Unless you're a die-hard individualist, you're likely connected to a network. That fact presents several new opportunities for clever hacks as well as mystifying failures. This chapter illuminates ways to take

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Conventions Used in This Book

This book uses the following typographical conventions:

Italic

Indicates new terms, URLs, email addresses, filenames, pathnames, and directories.

Constant width

Indicates commands, options, switches, variables, attributes, functions, user and group names, the contents of files, and the output from commands.

`Constant width bold`

In code examples, shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values.

Color

The second color is used to indicate a cross-reference within the text.

> This icon signifies a tip, suggestion, or general note.

> This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN, for example: "BSD Hacks by Dru Lavigne. Copyright 2004 O'Reilly Media, Inc., 0-596-00679-9."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at permissions@oreilly.com.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:
O'Reilly Media, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800) 998-9938 (in the United States or Canada)(707) 829-0515 (international or local) (707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:
http://www.oreilly.com/catalog/bsdhks

To comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:
http://www.oreilly.com/

Chapter 1. Customizing the User Environment

PREV < Day Day Up > NEXT

Hack 0 Introduction

Users of open source (http://opensource.org) Unix operating systems are an interesting breed. They like to poke under the surface of things, to find out how things work, and to figure out new and interesting ways of accomplishing common computing tasks. In short, they like to "hack."

While this book concentrates on the BSDs, many of the hacks apply to any open source operating system. Each hack is simply a demonstration of how to examine a common problem from a slightly different angle. Feel free to use any of these hacks as a springboard to your own customized solution. If your particular operating system doesn't contain the tool used in the solution, use a tool that does exist, or invent your own!

This chapter provides many tools for getting the most out of your working environment. You'll learn how to make friends with your shell and how to perform your most common tasks with just a few keystrokes or mouse clicks. You'll also uncover tricks that can help prevent command-line disasters. And, above all, you'll discover that hacking BSD is fun. So, pull your chair up to your operating system of choice and let's start hacking.

Hack 1 Get the Most Out of the Default Shell

Become a speed daemon at the command line.

For better or for worse, you spend a lot of time at the command line. If you're used to administering a Linux system, you may be dismayed to learn that bash is not the default shell on a BSD system, for either the superuser or regular user accounts.

Take heart; the FreeBSD superuser's default tcsh shell is also brimming with shortcuts and little tricks designed to let you breeze through even the most tedious of tasks. Spend a few moments learning these tricks and you'll feel right at home. If you're new to the command line or consider yourself a terrible typist, read on. Unix might be a whole lot easier than you think.

> NetBSD and OpenBSD also ship with the C shell as their default shell. However, it is not always the same tcsh, but often its simpler variant, csh, which doesn't support all of the tricks provided in this hack.
>
> However, both NetBSD and OpenBSD provide a tcsh package in their respective package collections.

1.2.1 History and Auto-Completion

I hate to live without three keys: up arrow, down arrow, and Tab. In fact, you can recognize me in a crowd, as I'm the one muttering loudly to myself if I'm on a system that doesn't treat these keys the way I expect to use them.

tcsh uses the up and down arrow keys to scroll through your command history. If there is a golden rule to computing, it should be: "You should never have to type a command more than once." When you need to

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Hack 2 Useful tcsh Shell Configuration File Options

Make the shell a friendly place to work in.

Now that you've had a chance to make friends with the shell, let's use its configuration file to create an environment you'll enjoy working in. Your prompt is an excellent place to start.

1.3.1 Making Your Prompt More Useful

The default tcsh prompt displays % when you're logged in as a regular user and hostname# when you're logged in as the superuser. That's a fairly useful way to figure out who you're logged in as, but we can do much better than that.

Each user on the system, including the superuser, has a *.cshrc* file in his home directory. Here are my current prompt settings:

```
 dru@~:grep prompt ~/.cshrc

if ($?prompt) then

        set prompt = "%B%n@%~%b: "
```

That isn't the default tcsh prompt, as I've been using my favorite customized prompt for the past few years. The possible prompt formatting sequences are easy to understand if you have a list of possibilities in front of you. That list is buried deeply within man cshrc, so here's a quick way to zero in on it:

```
 dru@~:man cshrc

/prompt may include
```

Here I've used the / to invoke the manpage search utility. The search string prompt may include brings you to the right section, and is intuitive enough that even my rusty old brain can remember it.

If you compare the formatting sequences shown in the manpage to my prompt string, it reads as follows:

```
 set prompt = "%B%n@%~%b: "
```

That's a little dense. Table 1-1 dissects the options.

Table 1-1. Prompt codes

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 3 Create Shell Bindings

Train your shell to run a command for you whenever you press a mapped key.

Have you ever listened to a Windows power user expound on the joys of hotkeys? Perhaps you yourself have been known to gaze wistfully at the extra buttons found on a Microsoft keyboard. Did you know that it's easy to configure your keyboard to launch your most commonly used applications with a keystroke or two?

One way to do this is with the bindkey command, which is built into the tcsh shell. As the name suggests, this command binds certain actions to certain keys. To see your current mappings, simply type bindkey. The output is several pages long, so I've included only a short sample. However, you'll recognize some of these shortcuts from [Hack #1].

```
 Standard key bindings

"^A"           ->  beginning-of-line

"^B"           ->  backward-char

"^E"           ->  end-of-line

"^F"           ->  forward-char

"^L"           ->  clear-screen

"^N"           ->  down-history

"^P"           ->  up-history

"^U"           ->  kill-whole-line


Arrow key bindings

down           -> history-search-forward

up             -> history-search-backward

left           -> backward-char

right          -> forward-char

home           -> beginning-of-line

end            -> end-of-line
```

The ^ means hold down your Ctrl key. For example, press Ctrl and then l, and you'll clear your screen more quickly than by typing clear.

Hack 4 Use Terminal and X Bindings

Take advantage of your terminal's capabilities.

It's not just the tcsh shell that is capable of understanding bindings. Your FreeBSD terminal provides the kbdcontrol command to map commands to your keyboard. Unfortunately, neither NetBSD nor OpenBSD offer this feature. You can, however, remap your keyboard under X, as described later.

1.5.1 Creating Temporary Mappings

Let's start by experimenting with some temporary mappings. The syntax for mapping a command with kbdcontrol is as follows:

```
kbdcontrol -f number "command"
```

Table 1-2 lists the possible numbers, each with its associated key combination.

Table 1-2. Key numbers

Number	Key combination
1, 2, . . . 12	F1, F2, . . . F12
13, 14, . . . 24	Shift+F1, Shift+F2, . . . Shift+F12
25, 26, . . . 36	Ctrl+F1, Ctrl+F2, . . . Ctrl+F12
37, 38, . . . 48	Shift+Ctrl+F1, Shift+Ctrl+F2, . . . Shift+Ctrl+F12
49	Home

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 5 Use the Mouse at a Terminal

HACK #5

Use your mouse to copy and paste at a terminal.

If you're used to a GUI environment, you might feel a bit out of your element while working at the terminal. Sure, you can learn to map hotkeys and to use navigational tricks, but darn it all, sometimes it's just nice to be able to copy and paste!

Don't fret; your mouse doesn't have to go to waste. In fact, depending upon how you have configured your system, the mouse daemon moused may already be enabled. The job of this daemon is to listen for mouse data in order to pass it to your console driver.

> Of course, if you're using screen [Hack #12], you can also take advantage of its copy and paste mechanism.

1.6.1 If X Is Already Installed

If you installed and configured X when you installed your system, moused is most likely started for you when you boot up. You can check with this:

```
% grep moused /etc/rc.conf

moused_port="/dev/psm0"

moused_type="auto"

moused_enable="YES"
```

Very good. moused needs to know three things:

- The mouse port (in this example, */dev/psm0*, the PS/2 port)
- The type of protocol (in this example, auto)

Hack 6 Get Your Daily Dose of Trivia

Brighten your day with some terminal eye candy.

As the saying goes, all work and no play makes Jack a dull boy. But what's a poor Jack or Jill to do if your days include spending inordinate amounts of time in front of a computer screen? Well, you could head over to http://www.thinkgeek.net/ to stock up on cube goodies and caffeine. Or, you could take advantage of some of the entertainments built into your operating system.

1.7.1 A Fortune a Day

Let's start by configuring some terminal eye candy. Does your system quote you a cheery, witty, or downright strange bit of wisdom every time you log into your terminal? If so, you're receiving a fortune:

```
login: dru
Password:
Last login: Thu Nov 27 10:10:16 on ttyv7


"You can't have everything. Where would you put it?"
                -- Steven Wright
```

If you're not receiving a fortune, as the superuser type /stand/sysinstall. Choose Configure, then Distributions, and select games with your spacebar. Press Tab to select OK, then exit out of sysinstall when it is finished.

Then, look for the line that runs */usr/games/fortune* in your *~/.cshrc* file:

```
% grep fortune ~/.cshrc
/usr/games/fortune
```

If for some reason it isn't there, add it:

```
% echo '/usr/games/fortune' >> ~/.cshrc
```

Don't forget to use both greater-than signs; you don't want to erase the contents of your *.cshrc* file! To test your change, use the source shell command, which re-executes the contents of the file. This can come in handy if you've updated an alias and want to take advantage of it immediately:

Hack 7 Lock the Screen

Secure your unattended terminal from prying eyes.

If you work in a networked environment, the importance of locking your screen before leaving your workstation has probably been stressed to you. After all, your brilliant password becomes moot if anyone can walk up to your logged in station and start poking about the contents of your home directory.

If you use a GUI on your workstation, your Window Manager probably includes a locking feature. However, if you use a terminal, you may not be aware of the mechanisms available for locking your terminal.

As an administrator, you may want to automate these mechanisms as part of your security policy. Fortunately, FreeBSD's screen locking mechanism is customizable.

1.8.1 Using lock

FreeBSD comes with lock (and it's available for NetBSD and OpenBSD). Its default invocation is simple:

```
 % lock

Key: 1234

Again: 1234

lock /dev/ttyv6 on genisis. timeout in 15 minutes.

time now is Fri Jan 2 12:45:02 EST 2004

Key:
```

Without any switches, lock will request that the user input a key which will be used to unlock the terminal. This is a good thing, as it gives the user an opportunity to use something other than her login password. If the user tries to be smart and presses Enter (for an empty password), the lock program will abort.

Once a key is set, it is required to unlock the screen. If a user instead types Ctrl-c, she won't terminate the program. Instead, she'll receive this message:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 8 Create a Trash Directory

Save "deleted" files until you're really ready to send them to the bit bucket.

One of the first things Unix users learn is that deleted files are really, really gone. This is especially true at the command line where there isn't any Windows-style recycling bin to rummage through should you have a change of heart regarding the fate of a removed file. It's off to the backups! (You do have backups, don't you?)

Fortunately, it is very simple to hack a small script that will send removed files to a custom trash directory. If you've never written a script before, this is an excellent exercise in how easy and useful scripting can be.

1.9.1 Shell Scripting for the Impatient

Since a script is an executable file, you should place your scripts in a directory that is in your path. Remember, your path is just a list of directories where the shell will look for commands if you don't give them full pathnames. To see your path:

```
 % echo $PATH

PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/loc
al/sbin:/usr/

local/bin:/usr/X11R6/bin:/home/dru/bin
```

In this output, the shell will look for executables in the *bin* subdirectory of dru's home directory. However, it won't look for executables placed directly in my home directory, or */home/dru*. Since *bin* isn't created by default, I should do that first:

```
 % cd

% mkdir bin
```

As I create scripts, I'll store them in */home/dru/bin*, since I don't have permission to store them anywhere else. Fortunately, no one else has permission to store them in my *bin* directory, so it's a good match.

The scripts themselves contain at least three lines:

```
 #!/bin/sh

# a comment explaining what the script does
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 9 Customize User Configurations

Now that you know how to set up a useful environment for yourself, it's time to share the wealth.

It's very easy for a system administrator to ensure that each newly created user starts out with the same configuration files. For example, every user can receive the same customized prompt, shell variables, or hotkeys.

Whenever you create a new user, several default (and hidden, or dot, files) are copied into the new user's home directory. In FreeBSD, the source of these files is */usr/share/skel/*. Any customizations you make to these files will be seen by all subsequently created users. Do note that you'll have to manually copy over any modified files to existing users.

It's useful to understand these files, as they apply to every user you create. Depending upon your needs, you'll probably end up removing some of the defaults, customizing others, and even adding a few of your own.

1.10.1 Default Files

Let's take a quick tour of the default files:

```
 % ls -l /usr/share/skel

total 24

drwxr-xr-x   2 root  wheel  512 Jul 28 16:09 ./

drwxr-xr-x  27 root  wheel  512 Jul 28 16:06 ../

-rw-r--r--   1 root  wheel  921 Jul 28 16:09 dot.cshrc

-rw-r--r--   1 root  wheel  248 Jul 28 16:09 dot.login

-rw-r--r--   1 root  wheel  158 Jul 28 16:09
dot.login_conf

-rw-------   1 root  wheel  371 Jul 28 16:09
dot.mail_aliases

-rw-r--r--   1 root  wheel  331 Jul 28 16:09
dot.mailrc

-rw-r--r--   1 root  wheel  797 Jul 28 16:09
dot.profile
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 10 Maintain Your Environment on Multiple Systems

The sign of a true Unix guru is the ability to perform a task quickly when confronted with an unfamiliar shell, keyboard, terminal, window manager, or operating system.

A large part of using Unix systems effectively involves configuring a comfortable environment using familiar tools available from the Unix shell prompt. It's much easier to perform a task quickly when all of the shortcuts your fingers have learned work on the first try.

Even something as simple as setting up your prompt the way you like it can steal significant time from your productivity if you need to do it on several hosts. If you're going to spend significant time in a Unix shell, it's worth getting organized. A bit of onetime effort will reward you later, every time you sit down at the keyboard.

1.11.1 Enter unison

unison is a tool for maintaining synchronized copies of directories. I've used it to maintain a central repository of all of my dot files, shell scripts, signatures file, SpamAssassin configuration—basically any file I'd like to have available, regardless of which host I happen to be logged into.

You can install unison from the NetBSD pkgsrc collection:

```
 # cd /usr/pkgsrc/net/unison

# make install clean
```

FreeBSD and OpenBSD ports also include *net/unison*.

Even better, this utility is available for most Unix and Windows platforms. See the main unison web site for details.

1.11.2 Using unison

Whenever I configure a new Unix host or get a shell on another system, I install unison. Then, I create a directory to receive the files I've stored in the */usr/work/sync* directory at host.example.com. I call the local

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 11 Use an Interactive Shell

Save and share an entire login session.

How many times have you either struggled with or tried to troubleshoot another user through a thorny problem? Didn't you wish you had another set of eyes behind you so you could simply type your command set, point at the troublesome output, and say, "That's the problem." Well, if you can't bring another user to your output, you can still share that real-time output using an interactive shell.

1.12.1 Recording All Shell Input and Output

There are actually several ways to share what is happening on your screen. Let's start by recording all of your input and output to a file. Then we'll see how we can also allow another user to view that output from another terminal.

Your BSD system comes with the script command which, not surprisingly, allows you to script your session. This command is extremely simple to use. Simply type script:

```
 % script

Script started, output file is typescript
```

By default, script will create an output file named *typescript* in your current directory. If you prefer, you can specify a more descriptive name for your script file:

```
 % script configure.firewall.nov.11.2003

Script started, output file is
configure.firewall.nov.11.2003
```

Regardless of how you invoke the command, a new shell will be created. This means that you will see the MOTD and possibly a fortune, and your *.cshrc* will be reread.

You can now carry on as usual and all input and output will be written to your script file. When you are finished, simply press Ctrl-d. You will see this message:

```
 Script done, output file is
configure.firewall.nov.11.2003
```

If you've ended a script and decide later to append some more work to a previous session, remember the -a (append) switch:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 12 Use Multiple Screens on One Terminal

Running a graphical environment is great. You can have numerous applications and utilities running, and you can interact with all of them at the same time. Many people who have grown up with a GUI environment look down upon those poor souls who continue to work in a terminal console environment. "After all," they say, "you can only do one thing at a time and don't get the same information and control that you have in a desktop environment."

It's true; they do say those things. (I am curious to know who they are, however.)

It's also true that the utility of a graphical environment diminishes when you need to administer machines remotely. Do you really want to squander network bandwidth just to maintain a GUI session?

Here are some more questions to ask yourself regarding remote administration:

- Are you worried about making your services vulnerable just so you can administer them across the Internet?
- Do you want a secure connection?
- Do you want to run multiple terminal sessions from a single login?
- Do you want to be able to password protect your session to prevent unauthorized access?
- Do you want multiple windows with labels and of different sizes?
- Do you want to copy and paste between the windows?

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Chapter 2. Dealing with Files and Filesystems

PREV   < Day Day Up >   NEXT

PREV   < Day Day Up >   NEXT

Hack 12 Introduction

Now that you're a bit more comfortable with the Unix environment, it's time to tackle some commands. It's funny how some of the most useful commands on a Unix system have gained themselves a reputation for being user-unfriendly. Do find, grep, sed, tr, or mount make you shudder? If not, remember that you still have novice users who are intimidated by—and therefore aren't gaining the full potential of—these commands.

This chapter also addresses some useful filesystem manipulations. Have you ever inadvertently blown away a portion of your directory structure? Would you like to manipulate */tmp* or your swap partition? Do your Unix systems need to play nicely with Microsoft systems? Might you consider ghosting your BSD system? If so, this chapter is for you.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 13 Find Things

Finding files in Unix can be an exercise in frustration for a novice user. Here's how to soften the learning curve.

Remember the first time you installed a Unix system? Once you successfully booted to a command prompt, I bet your first thought was, "Now what?" or possibly, "Okay, where is everything?" I'm also pretty sure your first foray into man find wasn't all that enlightening.

How can you as an administrator make it easier for your users to find things? First, introduce them to the built-in commands. Then, add a few tricks of your own to soften the learning curve.

2.2.1 Finding Program Paths

Every user should become aware of the three w's: which, whereis, and whatis. (Personally, I'd like to see some why and when commands, but that's another story.)

Use which to find the path to a program. Suppose you've just installed xmms and wonder where it went:

```
% which xmms

/usr/X11R6/bin/xmms
```

Better yet, if you were finding out the pathname because you wanted to use it in a file, save yourself a step:

```
% echo `which xmms` >> somefile
```

Remember to use the backticks (`), often found on the far left of the keyboard on the same key as the tilde (~). If you instead use the single quote (') character, usually located on the right side of the keyboard on the same key as the double quote ("), your file will contain the echoed string which xmms instead of the desired path.

The user's current shell will affect how which's switches work. Here is an example from the C shell:

```
% which -a xmms

-a: Command not found.

/usr/X11R6/bin/xmms
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 14 Get the Most Out of grep

You may not know where its odd name originated, but you can't argue the usefulness of grep.

Have you ever needed to find a particular file and thought, "I don't recall the filename, but I remember some of its contents"? The oddly named grep command does just that, searching inside files and reporting on those that contain a given piece of text.

2.3.1 Finding Text

Suppose you wish to search your shell scripts for the text $USER. Try this:

```
 % grep -s '$USER' *

add-user:if [ "$USER" != "root" ]; then

bu-user:  echo "  [-u user] - override $USER as the
user to backup"

bu-user:if [ "$user" = "" ]; then user="$USER"; fi

del-user:if [ "$USER" != "root" ]; then

mount-host:mounted=$(df | grep "$ALM_AFP_MOUNT/$USER")

.....

mount-user:  echo "  [-u user] - override $USER as
the user to backup"

mount-user:if [ "$user" = "" ]; then user="$USER"; fi
```

In this example, grep has searched through all files in the current directory, displaying each line that contained the text $USER. Use single quotes around the text to prevent the shell from interpreting special characters. The -s option suppresses error messages when grep encounters a directory.

Perhaps you only want to know the name of each file containing the text $USER. Use the -l option to create that list for you:

```
 % grep -ls '$USER' *

add-user

bu-user

del-user
```

PREV < Day Day Up > NEXT

Hack 15 Manipulate Files with sed

HACK #15

If you've ever had to change the formatting of a file, you know that it can be a time-consuming process.

Why waste your time making manual changes to files when Unix systems come with many tools that can very quickly make the changes for you?

2.4.1 Removing Blank Lines

Suppose you need to remove the blank lines from a file. This invocation of grep will do the job:

```
% grep -v '^$' letter1.txt > tmp ; mv tmp letter1.txt
```

The pattern ^$ anchors to both the start and the end of a line with no intervening characters—the regexp definition of a blank line. The -v option reverses the search, printing all nonblank lines, which are then written to a temporary file, and the temporary file is moved back to the original.

> grep must never output to the same file it is reading, or the file will end up empty.

You can rewrite the preceding example in sed as:

```
% sed '/^$/d' letter1.txt > tmp ; mv tmp letter1.txt
```

'/^$/d' is actually a sed script. sed's normal mode of operation is to read each line of input, process it according to the script, and then write the processed line to standard output. In this example, the expression '/^$/ is a regular expression matching a blank line, and the trailing d' is a sed function that deletes the line. Blank lines are deleted and all other lines are printed. Again, the results are redirected to a temporary file, which is then copied back to the original file.

2.4.2 Searching with sed

sed can also do the work of grep:

```
% sed -n '/$USER/p' *
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 16 Format Text at the Command Line

Combine basic Unix tools to become a formatting expert.

Don't let the syntax of the sed command scare you off. sed is a powerful utility capable of handling most of your formatting needs. For example, have you ever needed to add or remove comments from a source file? Perhaps you need to shuffle some text from one section to another.

In this hack, I'll demonstrate how to do that. I'll also show some handy formatting tricks using two other built-in Unix commands, tr and col.

2.5.1 Adding Comments to Source Code

sed allows you to specify an address range using a pattern, so let's put this to use. Suppose we want to comment out a block of text in a source file by adding // to the start of each line we wish to comment out. We might use a text editor to mark the block with bc-start and bc-end:

```
% cat source.c

  if (tTd(27, 1))

    sm_dprintf("%s (%s, %s) aliased to %s\n",

        a->q_paddr, a->q_host, a->q_user, p);

  bc-start

    if (bitset(EF_VRFYONLY, e->e_flags))

  {

    a->q_state = QS_VERIFIED;

    return;

  }

  bc-end

  message("aliased to %s", shortenstring(p,
MAXSHORTSTR));
```

and then apply a sed script such as:

```
% sed '/bc-start/,/bc-end/s/^/\/\//' source.c
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 17 Delimiter Dilemma

Deal with double quotation marks in delimited files.

Importing data from a delimited text file into an application is usually painless. Even if you need to change the delimiter from one character to another (from a comma to a colon, for example), you can choose from many tools that perform simple character substitution with great ease.

However, one common situation is not solved as easily: many business applications export data into a space- or comma-delimited file, enclosing individual fields in double quotation marks. These fields often contain the delimiter character. Importing such a file into an application that processes only one delimiter (PostgreSQL for example) may result in an incorrect interpretation of the data. This is one of those situations where the user should feel lucky if the process fails.

One solution is to write a script that tracks the use of double quotes to determine whether it is working within a text field. This is doable by creating a variable that acts as a text/nontext switch for the character substitution process. The script should change the delimiter to a more appropriate character, leave the delimiters that were enclosed in double quotes unchanged, and remove the double quotes. Rather than make the changes to the original datafile, it's safer to write the edited data to a new file.

2.6.1 Attacking the Problem

The following algorithm meets our needs:

1.

1. Create the switch variable and assign it the value of 1, meaning "nontext". We'll declare the variable tswitch and define it as tswitch = 1.

2.

2. Create a variable for the delimiter and define it. We'll use the variable delim with a space as the delimiter, so delim = ' '.

3.

3. Decide on a better delimiter. We'll use the tab character, so new_delim = '\t'.

4.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 18 DOS Floppy Manipulation

Bring simplicity back to using floppies.

If you're like many Unix users, you originally came from a Windows background. Remember your initial shock the first time you tried to use a floppy on a Unix system? Didn't Windows seem so much simpler? Forever gone seemed the days when you could simply insert a floppy, copy some files over, and remove the disk from the drive. Instead, you were expected to plunge into the intricacies of the mount command, only to discover that you didn't even have the right to use the floppy drive in the first place!

There are several ways to make using floppies much, much easier on your FreeBSD system. Let's start by taking stock of the default mechanisms for managing floppies.

2.7.1 Mounting a Floppy

Suppose I have formatted a floppy on a Windows system, copied some files over, and now want to transfer those files to my FreeBSD system. In reality, that floppy is a storage media. Since it is storing files, it needs a filesystem in order to keep track of the locations of those files. Because that floppy was formatted on a Windows system, it uses a filesystem called FAT12.

In Unix, a filesystem can't be accessed until it has been mounted. This means you have to use the mount command before you can access the contents of that floppy. While this may seem strange at first, it actually gives Unix more flexibility. An administrator can mount and unmount filesystems as they are needed. Note that I used the word administrator. Regular users don't have this ability, by default. We'll change that shortly.

Unix also has the additional flexibility of being able to mount different filesystems. In Windows, a floppy will always contain the FAT12 filesystem. BSD understands floppies formatted with either FAT12 or UFS, the Unix File System. As you might expect from the name, the UFS filesystem is assumed unless you specify otherwise.

For now, become the superuser and let's pick apart the default invocation of the mount command:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 19 Access Windows Shares Without a Server

Share files between Windows and FreeBSD with a minimum of fuss.

You've probably heard of some of the Unix utilities available for accessing files residing on Microsoft systems. For example, FreeBSD provides the mount_smbfs and smbutil utilities to mount Windows shares and view or access resources on a Microsoft network. However, both of those utilities have a caveat: they require an SMB server. The assumption is that somewhere in your network there is at least one NT or 2000 Server.

Not all networks have the budget or the administrative expertise to allow for commercial server operating systems. Sure, you can install and configure Samba, but isn't that overkill for, say, a home or very small office network? Sometimes you just want to share some files between a Windows 9x system and a Unix system. It's a matter of using the right-sized tool for the job. You don't bring in a backhoe to plant flowers in a window box.

2.8.1 Installing and Configuring Sharity-Light

If your small network contains a mix of Microsoft and Unix clients, consider installing Sharity-Light on the Unix systems. This application allows you to mount a Windows share from a Unix system. FreeBSD provides a port for this purpose (see the Sharity-Light web site for other supported platforms):

```
 # cd /usr/ports/net/sharity-light

# make install clean
```

Since Sharity-Light is a command-line utility, you should be familiar with UNC or the Universal Naming Convention. UNC is how you refer to Microsoft shared resources from the command line. A UNC looks like \\NetBIOSname\sharename. It starts with double backslashes, then contains the NetBIOS name of the computer to access and the name of the share on that computer.

Before using Sharity-Light, you need to know the NetBIOS names of the computers you wish to access. If you have multiple machines running Microsoft operating systems, the quickest way to view each system's name is with nbtstat. From one of the Windows systems, open a command prompt and type:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 20 Deal with Disk Hogs

Fortunately, you no longer have to be a script guru or a find wizard just to keep up with what is happening on your disks.

Think for a moment. What types of files are you always chasing after so they don't waste resources? Your list probably includes temp files, core files, and old logs that have already been archived. Did you know that your system already contains scripts capable of cleaning out those files? Yes, I'm talking about your periodic scripts.

2.9.1 Periodic Scripts

You'll find these scripts in the following directory on a FreeBSD system:

```
 % ls /etc/periodic/daily | grep clean

100.clean-disks

110.clean-tmps

120.clean-preserve

130.clean-msgs

140.clean-rwho

150.clean-hoststat
```

Are you using these scripts? To find out, look at your */etc/periodic.conf* file. What, you don't have one? That means you've never tweaked your default configurations. If that's the case, copy over the sample file and take a look at what's available:

```
 # cp /etc/defaults/periodic.conf /etc/periodic.conf

# more /etc/periodic.conf
```

2.9.1.1 daily_clean_disks

Let's start with daily_clean_disks. This script is ideal for finding and deleting files with certain file extensions. You'll find it about two pages into *periodic.conf*, in the Daily options section, where you may note that it's not enabled by default. Fortunately, configuring it is a heck of a lot easier than using cron to schedule a complex find statement.

Before you enable any script, test it first,

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 21 Manage Temporary Files and Swap Space

Add more temporary or swap space without repartitioning.

When you install any operating system, it's important to allocate sufficient disk space to hold temporary and swap files. Ideally, you already know the optimum sizes for your system so you can partition your disk accordingly during the install. However, if your needs change or you wish to optimize your initial choices, your solution doesn't have to be as drastic as a repartition—and reinstall—of the system.

> man tuning has some practical advice for guesstimating the appropriate size of swap and your other partitions.

2.10.1 Clearing /tmp

Unless you specifically chose otherwise when you partitioned your disk, the installer created a */tmp* filesystem for you:

```
 % grep tmp /etc/fstab

/dev/ad0s1e    /tmp    ufs    rw    2    2



% df -h /tmp

Filesystem    Size   Used  Avail Capacity  Mounted on

/dev/ad0s1e   252M   614K   231M     0%    /tmp
```

Here I searched */etc/fstab* for the */tmp* filesystem. This particular filesystem is 256 MB in size. Only a small portion contains temporary files.

> The df (disk free) command will always show you a number lower than the actual partition size. This is because eight percent of the filesystem is reserved to prevent users from inadvertently

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 22 Recreate a Directory Structure Using mtree

Prevent or recover from rm disasters.

Someday the unthinkable may happen. You're doing some routine maintenance and are distracted by a phone call or perhaps another employee's question. A moment later, you're faced with the awful realization that your fingers typed either a rm * or a rm -R in the wrong place, and now a portion of your system has evaporated into nothingness.

Painful thought, isn't it? Let's pause for a moment to catch our breath and examine a few ways to prevent such a scenario from happening in the first place.

Close your eyes and think back to when you were a fresh-faced newbie and were introduced to the omnipotent rm command. Return to the time when you actually read man rm and first discovered the -i switch. "What a great idea," you thought, "to be prompted for confirmation before irretrievably deleting a file from disk." However, you soon discovered that this switch can be a royal PITA. Face it, it's irritating to deal with the constant question of whether you're sure you want to remove a file when you just issued the command to remove that file.

2.11.1 Necessary Interaction

Fortunately, there is a way to request confirmation only when you're about to do something as rash as rm *. Simply make a file called *-i*. Well, actually, it's not quite that simple. Your shell will complain if you try this:

```
 % touch -i

touch: illegal option -- i

usage: touch [-acfhm] [-r file] [-t
[[CC]Y]MMDDhhmm[.SS]] file ...
```

You see, to your shell, *-i* looks like the -i switch, which touch doesn't have. That's actually part of the magic. The reason why we want to make a file called *-i* in the first place is to fool your shell: when you type rm *, the shell will expand * into all of the files in the directory. One of those files will be named *-i*, and, voila, you've just given the interactive

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Hack 23 Ghosting Systems

Do you find yourself installing multiple systems, all containing the same operating system and applications? As an IT instructor, I'm constantly installing systems for my next class or trying to fix the ramifications of a misconfiguration from a previous class.

As any system administrator can attest to, *ghosting* or hard drive-cloning software can be a real godsend. Backups are one thing; they retain your data. However, an image is a true timesaver—it's a copy of the operating system itself, along with any installed software and all of your configurations and customizations.

I haven't always had the luxury of a commercial ghosting utility at hand. As you can well imagine, I've tried every homegrown and open source ghosting solution available. I started with various invocations of dd, gzip, ssh, and dump, but kept running across the same fundamental problem: it was easy enough to create an image, but inconvenient to deploy that image to a fresh hard drive. It was doable in the labs that used removable drives, but, otherwise, I had to open up a system, cable in the drive to be deployed, copy the image, and recable the drive into its own system.

Forget the wear and tear on the equipment; that solution wasn't working out to be much of a timesaver! What I really needed was a floppy that contained enough intelligence to go out on the network and retrieve and restore an image. I tried several open source applications and found that Ghost For Unix, g4u, best fit the bill.

2.12.1 Creating the Ghost Disk

You're about two minutes away from creating a bootable g4u floppy. Simply download *g4u-1.12fs* from http://theatomicmoose.ca/g4u/ and copy it to a floppy:

```
# cat g4u-1.12fs > /dev/fd0
```

Your only other requirement is a system with a drive capable of holding your images. It can be any operating system, as long as it has an installed FTP server. If it's a FreeBSD system, you can configure an FTP server through /stand/sysinstall. Choose Configure from the menu, then Networking. Use your spacebar to choose Anon FTP.

Choose Yes to the configuration message and accept the defaults by

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Chapter 3. The Boot and Login Environments

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Introduction

When it comes to configuring systems, many users are reluctant to change the default boot process. Visions of unbootable systems, inaccessible data, and reinstalls dance in their heads. Yes, it is good to be mindful of such things as they instill the necessary attention to detail you'll need to use when making changes. However, once you've taken the necessary precautions, do take advantage of the hacks found in this chapter. Many of them will increase the security of your system.

This chapter also includes several password hacks. You'll learn how to create an effective password policy and monitor compliance to that policy. You'll find tools designed to assist you and your users in making good password choices. You'll also learn how to configure OTP, an excellent choice for when you're on the road and wish to access your network's resources securely.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 24 Customize the Default Boot Menu

Configure a splash screen.

You're not quite sure what you did to give the impression that you don't already have enough to do. Somehow, though, you were elected at the latest staff meeting to create a jazzy logo that will appear on every user's computer when they boot up in the morning.

While you may not be able to tell from first glance, the FreeBSD boot menu supports a surprising amount of customization. Let's start by examining your current menu to see which tools you have to work with.

3.2.1 The Default Boot Menu

Your default boot menu will vary slightly depending upon your version of FreeBSD and whether you chose to install the boot menu when you installed the system. Let's start with the most vanilla boot prompt and work our way up from there. In this scenario, you'll see this message as your system boots:

```
Hit [Enter] to boot immediately, or any other key
for command prompt.

Booting [/boot/kernel/kernel] in 10 seconds...
```

FreeBSD 5.1 introduced a quasi-graphical boot menu that includes a picture of Beastie and the following options:

```
Welcome to FreeBSD!



  1. Boot FreeBSD [default]

  2. Boot FreeBSD with ACPI disabled

  3. Boot FreeBSD in Safe Mode

  4. Boot FreeBSD in single user mode

  5. Boot FreeBSD with verbose logging

  6. Escape to loader prompt

  7. Reboot


          Select option, [Enter] for default
```

Hack 25 Protect the Boot Process

Thwart unauthorized physical access to a system.

Creating a snazzy boot environment for users is one thing. However, when it comes to booting up servers, your mind automatically shifts gears to security mode. Your goal is to ensure that only a very precious few on very rare occasions ever see the boot process on a server. After all, the golden rule in security land is "physical access equals complete access."

Here's a prime example—consider recovering from an unknown or forgotten root password. Go into the server closet, reboot that system, and press a key to interrupt the boot process to change the password. A few moments later, the system continues to boot as normal. This can be a real lifesaver if an admin leaves without divulging the root password. However, consider the security implications of an unauthorized user gaining physical access to that server: instant root access!

3.3.1 Limiting Unauthorized Reboots

Let's start by ensuring that regular users can't reboot the system either inadvertently or maliciously. By default, if a user presses Ctrl-Alt-Delete, the system will clean up and reboot. Typically this isn't an issue for servers, as most administration is done remotely and the server is safely locked away in a server closet. However, it can wreak havoc on workstations, especially if the user is used to working in a Windows environment and has become accustomed to pressing Ctrl-Alt-Delete. It's also worthwhile disabling on a server, as it ensures that a person has to first become the superuser in order to issue the reboot command.

> If you're logged into a remote machine over SSH and try Ctrl-Alt-Delete, it will affect your own machine, not the remote machine. reboot works well over the network, though.

Disabling this feature requires a kernel rebuild. (See [Hack #54] for detailed instructions.) Add one of these lines to your kernel

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 26 Run a Headless System

HACK #26

For those times when you want to run a system "headless."

Sometimes it is a simple matter of economy. Perhaps you've managed to scrounge up another system, but you don't have enough monitors, keyboards, or mice to go around. You also don't have the budget to purchase either those or a KVM switch. Sometimes it is a matter of security. Perhaps you're introducing a PC to a server closet and your physical security policy prevents server closet devices from being attached to monitors, keyboards, and mice.

Before you can run a system "headless," you need to have an alternative for accessing that system. Once you've removed input and output peripherals, your entry point into the system is now either through the network card or a serial port.

Going in through the network card is the easiest and is quite secure if you're using SSH. However, you should also consider a plan B. What if for some reason the system becomes inaccessible over the network? How do you get into the system then? Do you really want to gather up a spare monitor, keyboard, and mouse and carry them into the server closet?

A more attractive plan B may be to purchase a *null modem cable* as insurance. This is a crossed serial cable that is designed to go from one computer's serial port to another computer's serial port. This type of cable allows you to access a system without going through the network, which is a real lifesaver when the system isn't responding to the network. You can purchase this type of cable at any store that sells networking cables.

Your last consideration is whether the system BIOS will cooperate with your plan. Most newer BIOSes will. Many have a CMOS option that can be configured to disable "halt on errors." It's always a good idea to check out your available CMOS options before you start unplugging your peripherals.

3.4.1 Preparing the System

I've just installed a new FreeBSD 5.1 system. Since I didn't have a null modem cable handy, I installed the old-fashioned way with the monitor

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 27 Log a Headless Server Remotely

More on headless systems, but this time from the NetBSD perspective.

We've already seen in [Hack #26] that it's important to have an alternative method for connecting to a headless server. It's also important to be able to receive a headless system's console messages. This hack will show how to configure both on a NetBSD system.

3.5.1 Enabling a Serial Console

If you have another machine close to your headless server, it may be convenient to enable the serial console so that you can connect to it using a serial communication program. tip, included in the base system, and minicom , available through the packages collection, allow you to handle the server as if you were working on a real physical console.

To enable the serial console under NetBSD, simply tell the bootblocks to use the serial port as the console; they will configure the kernel on the fly to use it instead of the physical screen. You also need kernel support for the serial port device, which is included in the default GENERIC kernel.

However, changing the bootblocks configuration is a bit tricky because you need write permissions to the raw root device. As we are talking about a server, I assume the securelevel functionality is enabled; you must temporarily disable it by adding the options INSECURE line to your kernel. While in the kernel configuration file, double-check that it includes serial port support. Then, recompile your kernel.

Once you have access to the raw partition, update the bootblocks using the installboot utility. The process depends on the NetBSD version you are using.

If you are running 2.0 or higher, use the command shown next. Replace the *bootxx_ffsv1* file with the one that matches your root filesystem type; failure to do so will render your system unbootable.

```
 # /usr/sbin/installboot -o console=com0 /dev/rwd0a
/usr/mdec/bootxx_ffsv1
```

If you are running 1.6, use the following command instead:

```
 # /usr/mdec/installboot /usr/mdec/biosboot.com0.sym
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 28 Remove the Terminal Login Banner

Give users the information you want them to receive when they log in.

The default login process on a FreeBSD system produces a fair bit of information. The terminal message before the login prompt clearly indicates that the machine is a FreeBSD system. After logging in, a user will receive a copyright message and a Message of the Day (or motd), both of which contain many references to FreeBSD.

This may or may not be a good thing, depending upon the security requirements of your network. Your organization may also require you to provide legal information regarding network access or perhaps a banner touting the benefits of your corporation. Fortunately, a few simple hacks are all that stand between the defaults and your network's particular requirements.

3.6.1 Changing the Copyright Display

Let's start with the copyright information. That's this part of the default login process:

```
 Copyright (c) 1992-2003 The FreeBSD Project.

Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989,
1991, 1992, 1993, 1994

The Regents of the University of California. All
rights reserved.
```

To prevent users from seeing this information, simply:

```
 # touch /etc/COPYRIGHT
```

3.6.2 Changing the Message of the Day

Technically, you could add your own information to */etc/COPYRIGHT* instead of leaving it as an empty file. However, it is common practice to put your information in */etc/motd* instead. The default */etc/motd* contains very useful information to the new user, but it does get rather old after a few hundred logins.

You can edit */etc/motd* to say whatever suits your purposes—anything from your favorite sci-fi excerpt to all the nasty things that will happen

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 29 Protecting Passwords With Blowfish Hashes

Take these simple steps to thwart password crackers.

All good administrators know that passwords can be a weak link in the security chain. A malicious and determined user armed with a password cracker could conceivably guess enough of your network's passwords to access unauthorized resources.

3.7.1 Protecting System Passwords in General

Fortunately, you can make a password cracker's life very difficult in several ways. First, educate your users to choose complex, hard-to-guess passwords that are meaningful enough for them to remember. This will thwart dictionary password crackers [Hack #30], which use lists of dictionary and easy-to-guess words.

Second, be aware of who has superuser privileges and who has the right to backup */etc*. This directory contains the two password databases that are required to run a brute-force password cracker. As the name implies, this type of cracker will eventually guess every password in your password databases as it systematically tries every possible keyboard combination. Your best protection from this type of cracker is to prevent access to those password databases. This includes locking up your backup tapes and monitoring their access.

It is also a good idea to increase the amount of time it would take a brute-force cracker to crack a password database. FreeBSD, like most Unix systems, adds a magic bit of randomness—known as a *salt* —to the password when it is stored in the password database. The upshot is that a password cracker may have to try up to 4,096 different combinations for each and every password it tries to guess.

Using a strong algorithm to protect your passwords can also slow down a brute-force cracker. FreeBSD supports a hard-to-crack algorithm known as Blowfish. One of the first things I do after a FreeBSD install is to configure the password database to use Blowfish. While it is easier to do this before you create your users, it is still worth your while to implement it after you've created your user accounts.

3.7.2 Protecting System Passwords with Blowfish

PREV < Day Day Up > NEXT

Hack 30 Monitor Password Policy Compliance

When to use a password cracker utility.

Now that you've tightened up your password policy to thwart password crackers, it's time to learn how to use a password cracker to monitor the effectiveness of that password policy.

You're probably thinking, "Hey, wait a minute! Isn't that some sort of oxymoron? An administrator cracking passwords?" Well, it depends upon the type of password cracker you plan on using.

A brute-force password cracker such as John the ripper or slurpie will systematically try every possible keyboard combination until it has cracked every password in the password database. Does an administrator need to know every password in his network? Definitely not.

However, an administrator does need to know if her users are choosing easy-to-guess passwords, especially if she's responsible for enforcing compliance to the network's password policy. A properly tweaked dictionary password cracker such as crack is an effective way to monitor that compliance.

It is important that a network's security policy indicates in writing who runs the dictionary cracker, when it is run, and how the results are handled. For example, if the password policy forces users to change their passwords every 30 days, the following day is an excellent time for the delegated administrator to run the cracker. Ideally, the cracker will return no results. This means all users chose a strong password. Should the cracker find some weak passwords, the security policy should clearly outline the procedure used to ensure that noncompliant users change their passwords to ones that are harder to guess.

3.8.1 Installing and Using crack

Let's take a look at the most commonly used dictionary password cracker used on Unix systems, crack. You'll have to be the superuser for this entire hack because, fortunately, only the superuser has permission to crack the passwd database. crack should build on any

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 31 Create an Effective, Reusable Password Policy

Traditionally, it has been difficult for a Unix administrator to create and enforce a reusable password policy. Fortunately, PAM addresses this.

If you're using FreeBSD 5.0 or higher, your system has a PAM (Pluggable Authentication Modules) module specifically designed to assist in the creation and enforcement of a reusable password policy. If you're running a different version of BSD, see the end of this hack for other sources for this module.

3.9.1 Introducing pam_passwdqc

Before using this module, spend some time reading man pam_passwdqc, as it thoroughly covers each option and its possible values. Any values contained within parentheses are defaults. As you read through this manpage, compare those defaults with your own network's security policy and make note of any values that will require a change.

This PAM module is fairly comprehensive, allowing you to enable many of the features expected in a password policy. Here's an overview of the configurable features:

-
- Minimum and maximum password lengths
-
- Force a mix of digits, lowercase, uppercase, symbols, and non-ASCII characters
-
- Minimum number of words in a passphrase
-
- Minimum number of characters to consider as a string (dictionary word)
-
- Ability to search for strings that are words written backwards, or are words written in a mix of upper- and lowercase
-

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 32 Automate Memorable Password Generation

Make it easier for your users to choose good passwords.

It doesn't matter whether you're an administrator responsible for enforcing a password policy or an end user trying to comply with said policy. You're struggling against human nature when you ask users to choose—and remember—hard-to-guess passwords. Passwords that aren't random are easy to guess, and passwords that are too random tend to manifest themselves on sticky notes under users' keyboards or in their top drawers.

Wouldn't it be great if you could somehow offer users random but memorable password choices? There's a standard designed for just this purpose: APG, the Automated Password Generator.

3.10.1 Installing and Using apg

If you're running FreeBSD, you can install apg from the ports collection:

```
 # cd /usr/ports/security/apg

# make install clean
```

Once the port is installed, any user can run apg to generate a list of random, but pronounceable and memorable, passwords:

```
 % apg -q -m 10 -x 10 -M NC -n 10

plerOcGot5 (pler-Oc-Got-FIVE)

fobEbpigh6 (fob-Eb-pigh-SIX)

Ekjigyerj7 (Ek-jig-yerj-SEVEN)

CaujIvOwk8 (Cauj-Iv-Owk-EIGHT)

yenViapag0 (yen-Viap-ag-ZERO)

Fiwioshev3 (Fi-wi-osh-ev-THREE)

Twomitvac4 (Twom-it-vac-FOUR)

varbidCyd2 (varb-id-Cyd-TWO)

KlepezHap0 (Klep-ez-Hap-ZERO)

Naccudhav8 (Nac-cud-hav-EIGHT)
```

Notice that each password comes with a pronunciation guide, since it's

Hack 33 Use One Time Passwords

Sometimes even a complex password may not meet your security needs.

If you are on the road and need to access the corporate network from a non-secure computer in a public place, the risk of password leakage increases. Could the person next to you be shoulder surfing, watching as you log into the network? Does the computer you're using have some sort of installed spyware or keystroke logger? Is there a packet sniffer running somewhere on the network? In such a situation, a One Time Password can be a real lifesaver.

3.11.1 Configuring OPIE

FreeBSD comes with OPIE, or One-time Passwords In Everything, a type of software OTP system. It is easy to configure and doesn't require any additional hardware or proprietary software running on a server. Ideally, you should configure OPIE before leaving your secure network. For example, if you plan on traveling with your laptop, configure OPIE while connected to the office network. Make sure you are logged in as your regular user account to the particular system you'll need to access while on the road.

Start by adding yourself to the OPIE database, or */etc/opiekeys*, using opiepasswd. If you intend to access your workstation while on the road, run this command while physically sitting at your workstation. Include the console switch (-c) to indicate you are at that station's console, so it is safe to enter a passphrase:

```
 % opiepasswd -c

Adding dru:

Only use this method from the console; NEVER from
remote. If you are using

telnet, xterm, or a dial-in, type ^C now or exit with
no password.

Then run opiepasswd without the -c parameter.

Using MD5 to compute responses.

Enter new secret pass phrase:

Secret pass phrases must be between 10 and 127
characters long.
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 34 Restrict Logins

In this chapter, we've covered many methods of securing the boot and login environments. It's probably no surprise that you can further control who can log into your system and when: Unix systems contain many built-in mechanisms, allowing you to choose the most appropriate means and policy for your network.

Furthermore, the defaults may not always suit your needs. Do you really want users to be logged into multiple terminals when they can effectively do their work from one? For that matter, do you want any user, including nonemployees, to try his hand at logging into your systems at any hour of the night and day? Here's how to tighten up some defaults.

3.12.1 /etc/ttys

Since users log into terminals, a logical file to secure is the terminal configuration file, */etc/ttys*. We briefly saw this file in [Hack #24] when we password protected single-user mode.

This file is divided into three sections, one for each of the three types of terminals. Let's concern ourselves with the virtual terminals, ttyv, which are the terminals available for users physically seated at the system's keyboard.

```
 # grep ttyv /etc/ttys

ttyv0      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv1      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv2      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv3      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv4      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv5      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv6      "/usr/libexec/getty Pc"             cons25
      on   secure

ttyv7      "/usr/libexec/getty Pc"             cons25
```

Chapter 4. Backing Up

Introduction

I began gathering contributions for this book, it soon become obvious that there would be an entire chapter on backups. Not only do BSD users follow the mantra "backup, backup, backup," but every admin seems to have hacked his own solution to take advantage of the tools at hand and the environment that needs to be backed up.

If you're looking for tutorials on how to use dump and tar, you won't find them here. However, you will find nonobvious uses for their less well-known counterparts pax and cpio. I've also included a hack on backing up over ssh, to introduce the novice user to the art of combining tools over a secure network connection.

You'll also find scripts that fellow users have created to get the most out of their favorite backup utility. Finally, there are hacks that introduce some very useful open source third-party utilities.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 35 Back Up FreeBSD with SMBFS

A good backup can save the day when things go wrong. A bad—or missing—backup can ruin the whole week.

Regular backups are vital to good administration. You can perform backups with hardware as basic as a SCSI tape drive using 8mm tape cartridges or as advanced as an AIT tape library system using cartridges that can store up to 50 GB of compressed data. But what if you don't have the luxury of dedicated hardware for each server?

Since most networks are comprised of multiple systems, you can archive data from one server across the network to another. We'll back up a FreeBSD system using the tar and gzip archiving utilities and the smbutil and mount_smbfs commands to transport that data to network shares. These procedures were tested on FreeBSD 4.6-STABLE and 5.1-RELEASE.

4.2.1 Adding NETSMB Kernel Support

Since SMB is a network-aware filesystem, we need to build SMB support into the kernel. This means adding the proper options lines to the custom kernel configuration file. For information on building a custom kernel, see [Hack #54], the Building and Installing a Custom Kernel section (9.3) of the FreeBSD Handbook, and relevant information contained in */usr/src/sys/i386/conf*.

Add the following options under the makeoptions section:

```
 options     NETSMB             # SMB/CIFS requester

options     NETSMBCRYPTO       # encrypted password
support for SMB

options     LIBMCHAIN          # mbuf management library

options     LIBICONV

options     SMBFS
```

Once you've saved your changes, use the make buildkernel and make installkernel commands to build and install the new kernel.

4.2.2 Establishing an SMB Connection with a Host System

PREV < Day Day Up > NEXT

Hack 36 Create Portable POSIX Archives

Create portable tar archives with pax.

Some POSIX operating systems ship with GNU tar as the default tar utility (NetBSD and QNX6, for example). This is problematic because the GNU tar format is not compatible with other vendors' tar implementations. GNU is an acronym for "GNU's not UNIX"—in this case, GNU's not POSIX either.

4.3.1 GNU Versus POSIX tar

For filenames or paths longer than 100 characters, GNU uses its own @LongName tar format extension. Some vendors' tar utilities will choke on the GNU extensions. Here is what Solaris's archivers say about such an archive:

```
 % pax -r < gnu-archive.tar

pax: ././@LongLink : Unknown filetype

% tar xf gnu-archive.tar

tar: directory checksum error
```

There definitely appears to be a disadvantage with the distribution of non-POSIX archives. A solution is to use pax to create your tar archives in the POSIX format. I'll also provide some tips about using pax's features to compensate for the loss of some parts of GNU tar's extended feature set.

4.3.2 Replacing tar with pax

The NetBSD and QNX6 pax utility supports a tar interface and can also read the @LongName GNU tar format extension. You can use pax as your tar replacement, since it can read your existing GNU-format archives and can create POSIX archives for future backups. Here's how to make the quick conversion.

First, replace */usr/bin/tar*. That is, rename GNU tar and save it in another directory, in case you ever need to restore GNU tar to its previous location:

```
 # mv /usr/bin/tar /usr/local/bin/gtar
```

Next, create a symlink from pax to tar. This will allow the pax utility to

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 37 Interactive Copy

When cp alone doesn't quite meet your copy needs.

The cp command is easy to use, but it does have its limitations. For example, have you ever needed to copy a batch of files with the same name? If you're not careful, they'll happily overwrite each other.

4.4.1 Finding Your Source Files

I recently had the urge to find all of the scripts on my system that created a menu. I knew that several ports used scripts named configure and that some of those scripts used dialog to provide a menu selection.

It was easy enough to find those scripts using find:

```
 % find /usr/ports -name configure -exec grep -l
"dialog" /dev/null {  } \;

/usr/ports/audio/mbrolavox/scripts/configure

/usr/ports/devel/kdesdk3/work/kdesdk-3.2.0/configure

/usr/ports/emulators/vmware2/scripts/configure

(snip)
```

This command asks find to start in */usr/ports*, looking for files -named configure. For each found file, it should search for the word dialog using -exec grep. The -l flag tells grep to list only the names of the matching files, without including the lines that match the expression. You may recognize the /dev/null { } \; from [Hack #13] .

Normally, I could tell cp to use those found files as the source and to copy them to the specified destination. This is done by enclosing the find command within a set of backticks (`), located at the far top left of your keyboard. Note what happens, though:

```
 % mkdir ~/scripts

% cd ~/scripts

% cp `find /usr/ports -name configure -exec grep -l
"dialog" \

    /dev/null {  } \;` .

% ls ~/scripts
```

Hack 38 Secure Backups Over a Network

When it comes to backups, Unix systems are extremely flexible. For starters, they come with built-in utilities that are just waiting for an administrator's imagination to combine their talents into a customized backup solution. Add that to one of Unix's greatest strengths: its ability to see everything as a file. This means you don't even need backup hardware. You have the ability to send your backup to a file, to a media, to another server, or to whatever is available.

As with any customized solution, your success depends upon a little forethought. In this scenario, I don't have any backup hardware, but I do have a network with a 100 Mbps switch and a system with a large hard drive capable of holding backups.

4.5.1 Initial Preparation

On the system with that large hard drive, I have sshd running. (An alternative to consider is the scponly shell; see [Hack #63]). I've also created a user and a group called rembackup:

```
 # pw groupadd rembackup

# pw useradd rembackup -g rembackup -m -s /bin/csh

# passwd rembackup

Changing local password for rembackup

New Password:

Retype New Password:

#
```

If you're new to the pw command, the -g switch puts the user in the specified group (which must already exist), the -m switch creates the user's home directory, and the -s switch sets the default shell. (There's really no good mnemonic; perhaps no one remembers what, if anything, pw stands for.)

Next, from the system I plan on backing up, I'll ensure that I can ssh in as the user rembackup. In this scenario, the system with the large hard drive has an IP address of 10.0.0.1:

```
 % sshd -l rembackup 10.0.0.1

The authenticity of host '10.0.0.1 (10.0.0.1)' can't
be established.
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 39 Automate Remote Backups

Make remote backups automatic and effortless.

One day, the IDE controller on my web server died, leaving the files on my hard disk hopelessly corrupted. I faced what I had known in the back of my mind all along: I had not been making regular remote backups of my server, and the local backups were of no use to me now that the drive was corrupted.

The reason for this, of course, is that doing remote backups wasn't automatic and effortless. Admittedly, this was no one's fault but my own, but my frustration was sufficient enough that I decided to write a tool that would make automated remote snapshots so easy that I wouldn't ever have to worry about it again. Enter rsnapshot.

4.6.1 Installing and Configuring rsnapshot

Installation on FreeBSD is a simple matter of:

```
# cd /usr/ports/sysutils/rsnapshot

# make install
```

I didn't include the clean target here, as I'd like to keep the work subdirectory, which includes some useful scripts.

> If you're not using FreeBSD, see the original HOWTO at the project web site for detailed instructions on installing from source.

The install process neither creates nor installs the config file. This means that there is absolutely no possibility of accidentally overwriting a previously existing config file during an upgrade. Instead, copy the example configuration file and make changes to the copy:

```
# cp /usr/local/etc/rsnapshot.conf.default
/usr/local/etc/rsnapshot.conf
```

The *rsnapshot.conf* config file is well commented, and much of it should be fairly self-explanatory. For a full reference of all the various options, please consult man rsnapshot.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 40 Automate Data Dumps for PostgreSQL Databases

Building your own backup utility doesn't have to be scary.

PostgreSQL is a robust, open source database server. Like most database servers, it provides utilities for creating backups. PostgreSQL's primary tools for creating backup files are pg_dump and pg_dumpall. However, if you want to automate your database backup processes, these tools have a few limitations:

-
- pg_dump dumps only one database at a time.
-
- pg_dumpall dumps all of the databases into a single file.
-
- pg_dump and pg_dumpall know nothing about multiple backups.

These aren't criticisms of the backup tools—just an observation that customization will require a little scripting. Our resulting script will backup multiple systems, each to their own backup file.

4.7.1 Creating the Script

This script uses Python and its ability to execute other programs to implement the following backup algorithm:

1.
1. Change the working directory to a specified database backup directory.
2.
2. Rename all backup files ending in *.gz* so that they end in *.gz.old*. Existing files ending in *.gz.old* will be overwritten.
3.
3. Clean up and analyze all PostgreSQL databases using its vacuumdb command.
4.

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Hack 41 Perform Client-Server Cross-Platform Backups with Bacula

Don't let the campy name fool you. Bacula is a powerful, flexible, open source backup program. .

Having problems finding a backup solution that fits all your needs? One that can back up both Unix and Windows systems? That is flexible enough to back up systems with irregular backup needs, such as laptops? That allows you to run scripts before or after the backup job? That provides browsing capabilities so you can decide upon a restore point? Bacula may be what you're looking for.

4.8.1 Introducing Bacula

Bacula is a client-server solution composed of several distinct parts:

Director

The Director is the most complex part of the system. It keeps track of all clients and files to be backed up. This daemon talks to the clients and to the storage devices.

Client/File Daemon

The Client (or File) Daemon runs on each computer which will be backed up by the Director. Some other backup solutions refer to this as the Agent.

Storage Daemon

The Storage Daemon communicates with the backup device, which may be tape or disk.

Console

The Console is the primary interface between you and the Director. I

Chapter 5. Networking Hacks

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Introduction

You probably spend most of your time accessing servers on the Internet or on your own network. In fact, networking has become so prevalent, it's becoming increasingly difficult to tolerate even short periods of network outages.

This chapter contains many ideas for accessing networking services when the conventional avenues seem to be unavailable. Have you ever wanted to train your system to notify you of its new network configuration when its primary link becomes unavailable? Would you like to check your email from a system that doesn't contain a preconfigured email client? How can you maintain network connectivity when your ISP's DHCP server no longer recognizes your DHCP client?

You'll also gain insight into how some of the networking services and tools we often take for granted work. Become a tcpdump guru—or at least lose the intimidation factor. Understand your DNS messages and how to troubleshoot your DNS servers. Tame your sendmail daemon.

Finally, meet two excellent open source utilities that allow you to perform routine tasks simultaneously on all of your servers.

Hack 42 See Console Messages Over a Remote Login

View a server's console messages remotely

As a Unix system administrator, you can do 99% of your work remotely. In fact, it is very rare indeed that you'll need to sit down in front of a server (assuming the server even has an attached keyboard! [Hack #26]).

However, one of the key functionalities you lose in remote administration is the ability to see the remote server's console. All is not lost, though. First, let's answer these questions: "What do you mean by the console, and why would you want to see it?"

5.2.1 The Console

If you're physically sitting at a system, the console is the virtual terminal you see when you press Alt-F1. If you've ever logged into this particular virtual terminal, you've probably noticed that error messages appear here. These messages can be rather disconcerting when you're working at the console, especially if you're fighting your way through vi and bright white error messages occasionally overwrite your text.

If you ever find yourself in that situation, Esc-Ctrl-r will refresh your screen. Better yet, don't log into Alt-F1 when you're physically sitting at a system. Instead, log into a different terminal, say, the one at Alt-F2.

However, when you access a remote system, you can't log into a virtual terminal, and the console is considered to be a virtual terminal. (You access it by pressing Alt-F1 at the local keyboard, after all). Instead, you log into a *pseudoterminal* (also known as a *network terminal*).

Here's an example. I'm sitting at a system and have logged into the virtual terminals at Alt-F2 and Alt-F3. From Alt-F3, I've used ssh to log into the localhost. If I run the w command, I'll see this:

```
 % w

12:25  up 22 mins, 3 users, load averages: 0:00,
0:00, 0:00

USER          TTY     FROM             LOGIN@
```

Hack 43 Spoof a MAC Address

Even good guys can use secret identities.

Okay, I know what you're thinking. There's never a legitimate reason to spoof any type of address, right? Even if there were, why would you bother to spoof a MAC address, other than to prove that it can be done?

Consider the following scenario. I was administrating a small network where the ISP restricted the number of IP addresses a DHCP client was allowed to receive. Their DHCP server kept track of the leased addresses by using a combination of the client's MAC address and an OS identifier. One day I needed to replace that network's external NIC. It took me a while to figure out why the new NIC refused to pick up a DHCP address from the ISP. Once the restriction was explained to me, I contemplated my available courses of action. One was to spend the afternoon listening to Musak in the hopes that I'd eventually get to speak to one of the ISP's customer service representatives. I decided my time would be better spent if I instead took 30 seconds and spoofed the old MAC address. This provided a quick solution that allowed the owner to get back online until he could make arrangements with the ISP regarding the new MAC address.

5.3.1 Spoofing on FreeBSD

Before I could accomplish the spoof, I needed two pieces of information. The first was the MAC address for the old NIC. Fortunately, I record such things in a binder. However, I initially found out that information using ifconfig. In this scenario, the interface in question was called rl0:

```
 % ifconfig rl0

rl0:
flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>
mtu 1500

        inet 192.168.2.12 netmask 0xffffff00
broadcast 192.168.2.255

        ether 00:05:5d:d2:19:b7

        media: Ethernet autoselect (10baseT/UTP)
```

The MAC address is the hex number immediately following ether.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 44 Use Multiple Wireless NIC Configurations

Take the pain out of configuring your laptop's wireless interface.

If you use a laptop and have remote sites that you visit regularly, configuring your wireless interface can be interesting. For example, every wireless network has a unique service set identifier (SSID). Each site that uses WEP will also require a unique encryption key. Some networks may use static IP addresses, while others may use a DHCP server.

You could keep a copy of each network's configuration in your wallet and reconfigure your NIC manually at each site, but wouldn't you rather automate the various network configurations and choose the desired configuration after bootup?

For the purpose of this exercise, we will assume that the wireless access points have been properly configured and activated.

5.4.1 Initial Preparation

Before you can script the network configurations, you'll need to collect the information listed next. I've associated the necessary information with ifconfig's keywords where possible. You will see these keywords in the configuration script.

-
- ssid, the name of the wireless network
-
- authmode, the network's authorization mode (none, open, or shared)
-
- nwkey, the encryption key, in hexadecimal
-
- Whether to use a static IP address or dhclient to obtain dynamic IP address information
-

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 45 Survive Catastrophic Internet Loss

Set up your network to recover from a full Internet loss.

Someday this all too common event may happen: while you're away from your network, your connection dies. Whether the ISP drops it, the cable gets unplugged or the server behind your NAT box dies, it is gone. You are now lost at sea, not knowing what is actually going on back at home. You ping, telnet, and pray to the network gods, but nothing seems to work.

Wouldn't it be better if your network could recognize that it has lost that connection and find a way for you to get back in touch? The system that I set up did just that. All it took was a well-configured OpenBSD firewall with NAT and a short Ruby program that uses the Jabber protocol to get my attention.

5.5.1 Hardware Configuration

I use OpenBSD on a 486 to make my network resistant to total connectivity failure. The computer has two network cards, one for the DSL bridge and the other for the rest of the network. In addition, I managed to find a 56k ISA modem.

Since this computer provides little more than firewall and NAT services, it's more than capable of serving a small home or business network. The DSL bridge provides the primary Internet connection with a static IP. The service through my provider is usually quite good, but there have been troubled times. The house has only one phone line, which is plugged into the 56k modem in the same computer as the DSL line. You could easily make the modem computer a different machine entirely, but I found that this 486 is quite compact and sufficient for my purposes.

5.5.2 Connectivity Software

The current OpenBSD operating system (Version 3.4 as of this writing) comes with a wonderful firewall and NAT package, named Packet Filter (PF). PF works well on a day-to-day basis moving my packets from the network to the Internet. Unfortunately, it does not handle the loss of the connection to the ISP. A full discussion for configuring PF is

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 46 Humanize tcpdump Output

Make friends with tcpdump.

One of the most useful utilities in a network administrator's tool belt is tcpdump. While you probably agree, I bet the very thought of wading through a tcpdump sniff makes you groan. Take heart: I'll walk you through some concrete examples that show how to zero in on the information you need to solve the particular network problem that prompted you to consider doing a packet sniff in the first place.

You might be thinking, "Why bother? There are much nicer utilities out there." That's true. My personal favorite happens to be ethereal. However, you don't always have the luxury of working on a system that allows you to install third-party utilities or, for that matter, even has X installed. tcpdump is guaranteed to be on your BSD system. It's there, it's quick, it's dirty, and it's darn effective if you know how to harness its power.

5.6.1 The Basics

Let's start with the basics: starting a capture. Before you can capture any packets, you need to be the superuser. You also need to have the bpf device in your kernel. If you're using the GENERIC kernel, you're set. If you've created your own custom kernel [Hack #54], double-check you still have that device. In this example, my kernel configuration file is called *CUSTOM*:

```
 # grep bpf /usr/src/sys/i386/conf/CUSTOM

# The 'bpf' device enables the Berkeley Packet Filter.

device    bpf    #Berkeley packet filter
```

You also need to know the names of your interfaces and which interface is cabled to the network you wish to sniff. You can find this with ifconfig:

```
 # ifconfig

rl0:
flags=8802<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>
mtu 1500

        inet 192.168.3.20 netmask 0xffffff00
broadcast 192.168.3.255

        ether 00:05:5d:d2:19:b7
```

Hack 47 Understand DNS Records and Tools

Demystify DNS records.

DNS is one of those network services that has to be configured carefully and tested regularly. A misconfigured DNS server can prevent the world from finding your web and mail servers. Worse, a misconfigured DNS server can allow the world to find more than just your web and mail servers.

Even if you're not a DNS administrator, you should still know some handy DNS commands. The simple truth is, if DNS isn't working, you're not going anywhere. That means no surfing, no downloading, and no email for you.

5.7.1 Exploring Your ISP's DNS

On your home system, you most likely receive your DNS information from your ISP's DHCP server. Do you know where to find your primary and secondary DNS server addresses? If not, try this:

```
 % more /etc/resolv.conf

search domain.org

nameserver 204.101.251.1

nameserver 204.101.251.2
```

Another method is to use the dig (domain information groper) utility. Here, I'll ask for the nameservers (ns) for the sympatico.ca network:

```
 % dig ns sympatico.ca



; <<>> DiG 8.3 <<>> ns sympatico.ca

;; res options: init recurs defnam dnsrch

;; got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2

;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY:
0, ADDITIONAL: 4

;; QUERY SECTION:

;;         sympatico.ca, type = NS, class = IN
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 48 Send and Receive Email Without a Mail Client

Learn to speak SMTP and POP3.

Contrary to popular belief, you don't have to go to the trouble of configuring an email client just because you want to check your email or send off a quick email message.

Normally when you use the telnet application, you use a Telnet client to attach to a Telnet server listening on port 23. Once you're connected, you can log in and do anything on that device as if you were physically there, typing at its keyboard.

The Telnet client has even more powerful capabilities than this. If you specify a port number with the telnet command, you will attach directly to the TCP server listening on that port. If you know which commands that server can respond to, and if the service understands plain text commands, you can talk directly to that server. This essentially means that you no longer require a client application specific to that server.

5.8.1 Sending Email with telnet

Whenever you send an email, you connect to an SMTP server listening on port 25. Let's use telnet to see what really happens in the background and which commands the client and the SMTP server exchange. Note that in the following examples, the names and addresses have been changed to protect the innocent.

```
 % telnet smtp.mycompany.com 25

Trying 1.2.3.4...

Connected to smtp.mycompany.com.

Escape character is '^]'.

220 smtp.mycompany.com ESMTP server (InterMail
version x) ready Sun, 2

Nov 2003 09:54:18 -0500

mail from:<moi@mycompany.com>

250 Sender <moi@mycompany.com> Ok

rcpt to:<you@mycompany.com>
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 49 Why Do I Need sendmail?

As an end user, you've probably asked yourself: "If all I'm doing is running a FreeBSD machine for personal use, why should I need to run a heavyweight MTA daemon like sendmail?"

sendmail is the standard Mail Transport Agent (MTA) on FreeBSD, as it is on most Unix systems. In fact, the majority of email passing over the Internet will probably travel through a sendmail server at some point. However, sendmail isn't the easiest software package to manage, and the configuration file syntax gives most people a headache. There are several alternative MTA packages available, but these are also industrial-strength programs suitable for demanding use.

Many modern graphical email clients, such as Netscape Mail or Evolution, can send email directly to a mail server machine across the network. So, no, you won't need an MTA on your local machine to send email. (However, you will need an MTA if you use one of the more traditional Unix mail clients, such as mail, mutt, or pine.)

Regardless of your email client, if you want to see any automatic emails the system sends—usually from the periodic scripts—then you do require an MTA. More precisely, Unix programs expect to be able to send email by piping its text into the standard input of */usr/sbin/sendmail*, and have the system take care of the rest of the work for them.

> The venerable sendmail is only one of many MTAs available. Choosing another MTA does not always mean that you need to change the habits you picked up while working with sendmail. All three major BSD systems have a translator file, */etc/mailer.conf*, that identifies which commands to execute when the user or another process executes sendmail, mailq, or newaliases.
>
> For example, if you install postfix, you still use the sendmail command, even though the real job is done by the commands from the postfix package. The existence of */etc/mailer.conf* makes it easy

Hack 50 Hold Email for Later Delivery

Control when sendmail uses an intermittent Internet connection.

The default sendmail configuration assumes that you have a constant network connection. What if you're on a dial-up system and want to be able to work on emails without causing your modem to dial up immediately? In this scenario, you want to queue your sent messages to send later, the next time you go online.

5.10.1 Configuring sendmail Queueing

Fortunately, sendmail has a "hold expensive" function designed for this purpose. To activate it, add the following lines to the */etc/mail/<hostname>.mc* file:

```
 define(`confCON_EXPENSIVE', `True')dnl

MODIFY_MAILER_FLAGS(`RELAY', `+e')dnl

MODIFY_MAILER_FLAGS(`SMTP', `+e')dnl

MODIFY_MAILER_FLAGS(`ESMTP', `+e')dnl

MODIFY_MAILER_FLAGS(`SMTP8', `+e')dnl

define(`confTO_QUEUEWARN', `12h')dnl
```

The first line enables the feature. The next four lines add the letter e to the flags for each named mailer, to indicate that it is "expensive" and that email should first be queued rather than immediately delivered. The last line just extends the length of time the system will wait before it warns you that your message hasn't been delivered yet (the default is four hours).

Now just build the configuration file, install it, and restart sendmail as usual:

```
 # cd /etc/mail

# make

# make install

# make restart-mta
```

The four mailers listed (RELAY, SMTP, ESMTP, and SMTP8) will handle the bulk of all transmissions over the network. The configuration of both local and remote mail systems will determine which one to use. However, if you send out all of your mail via your provider's smart

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 51 Get the Most Out of FTP

Get the most out of stock ftp with macros and scripts.

In this age of GUIs and feature-rich browsers, it's easy to forget how quick and efficient command-line ftp can be. That is, until you're logged into a system that doesn't have X installed, nor a browser, nor any fancy FTP programs. If it's really your lucky day, it won't even have any manpages. And, of course, you'll need to download something.

Perhaps you find yourself using ftp all the time, always going to the same FTP servers and downloading from or uploading to the same directories. Clearly, it's time for some FTP automation.

5.11.1 Automating Logins

Have you ever noticed how easy it is to use FTP from a modern browser? Simply click on a hyperlink to start a download. At the command line, though, you can't even browse the FTP directory structure until you successfully log into the FTP server. Well, guess what: you always have to log into an FTP server. It's just that your web browser hides this little detail by doing it for you in the background.

You can achieve the same transparency for command-line ftp by creating a file called *.netrc* in your home directory and placing the following line in that file:

```
% more ~/.netrc
```

```
default login anonymous password genisis@istar.ca
```

This line will work for any FTP server on the Internet that accepts anonymous logins. (Most do, unless it's a private server.) When creating your own file, use your own email address as the password.

Test your change with this command:

```
% ftp ftp.freebsd.org
```

Compare your results to the FTP output in [Hack #71] . You should receive the same banner shown there without having to first type in a username and password.

If you're a webmaster who uses FTP to upload your new files, you do

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 52 Distributed Command Execution

Use tentakel for parallel, distributed command execution.

Often you want to execute a command not only on one computer, but on several at once. For example, you might want to report the current statistics on a group of managed servers or update all of your web servers at once.

5.12.1 The Obvious Approach

You could simply do this on the command line with a shell script like the following:

```
 # for host in hostA hostB hostC

> do ssh $host do_something

> done
```

However, this has several disadvantages:

-
- It is slow because the connections to the remote hosts do not run in parallel. Every connection must wait for the previous one to finish.
-
- Managing many sets of hosts can become a complicated task because there is no easy way to define groups of hosts (e.g., mailservers or workstations).
-
- The output is provided by the program that is run remotely.
-
- The output is hard to read because there are no marks indicating when the output for a specific host begins or ends.

5.12.2 How tentakel Can Help

While you could write a shell script to address some of these disadvantages, you might want to consider tentakel, which is available in the ports collection. Its execution starts multiple threads that run independently of each other. The maximum waiting time depends on the

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 53 Interactive Remote Administration

Managing a large network can be a daunting task. Even with the Unix utilities available for remote administration, making changes on many systems can be taxing. Scripting tools make life easier to some extent, but some tasks require hands- and eyes-on interaction.

Several system utilities allow you to execute the same command on multiple hosts. This form of loosely coupled clustering is useful for information gathering and some monitoring purposes. However, on some occasions, you not only need to run a process on multiple hosts, but you must also observe it and interact with the process to resolve host-specific issues. An administration shell script will save typing and minimize mistakes, but it's hard to write a script that will work correctly on every machine on a diverse network.

Wouldn't it be nice if there were a program that allowed you to interact with your remote hosts while running parallel commands? Enter ClusterIt.

5.13.1 Why ClusterIt?

ClusterIt is a set of tools written by Tim Rightnour, designed to place all of your network hosts at your fingertips. ClusterIt includes utilities for running a single command on all of the hosts in your cluster. It also allows automatic distribution of the tasks to any available hosts in a defined group. It uses a remote login method, such as sshd on the target hosts, so you only need to install it on the control host.

Scripts can also synchronize between task completions on different hosts. For example, you can set two hosts to compile an application and install it on the other machine. Neither host should begin the installation until the other host has finished compiling, but it is impossible to predict which host will finish first. ClusterIt defines barrier operations that can be included in a script to prevent passing a synchronization point until all hosts have caught up.

In most clustering systems for Unix, once you issue a command, you cannot interact with the hosts in the cluster individually; you only see the final output of each command run on each of the hosts. ClusterIt does not have this limitation, making it ideal for dealing with processes that

PREV < Day Day Up > NEXT

Chapter 6. Securing the System

Introduction

This chapter includes several hacks that demonstrate some security mechanisms that aren't well-documented elsewhere. I've also provided some new twists on old security favorites. Everyone has heard of sudo, but are you also aware of the security pitfalls it can introduce? You're probably also well-versed in ssh and scp, but you may have yet to harness the usefulness of scponly.

You'll also find several scripts to automate some common security practices. Each provides an excellent view into another administrator's thought processes. Use their examples to fuel your imagination and see what security solutions you can hack for your own network.

Hack 54 Strip the Kernel

Don't be shy. A kernel stripped down to the bare essentials is a happy kernel.

Picture the typical day in the life of a system administrator. Your mission, if you choose to accept it, is to achieve the impossible. Today, you're expected to:

-
- Increase the security of a particular server
-
- Attain a noticeable improvement in speed and performance

Although there are many ways to go about this, the most efficient way is to strip down the kernel to its bare-bones essentials. Having this ability gives an administrator of an open source system a distinct advantage over his closed source counterparts.

The first advantage to stripping the kernel is an obvious security boost. A vulnerability can't affect an option the kernel doesn't support. The second is a noticeable improvement in speed and performance. Kernels are loaded into memory and must stay in memory. You may be wasting precious memory resources if you're loading options you have no intention of ever using.

If you've never compiled a kernel or changed more than one or two kernel options, I can hear you groaning now. You're probably thinking, "Anything but that. Kernels are too complicated to understand." Well, there is a lot of truth in the idea that you haven't really used an operating system until you've gone through that baptism of fire known as kernel compiling. However, you may not have heard that compiling a kernel isn't all that difficult. So, grab a spare afternoon and a test system; it's high time to learn how to hack a BSD kernel.

I'll demonstrate on a FreeBSD system, but you'll find resources for other systems at the end of this hack.

Before you start, double-check that you have the kernel source installed. On an Intel FreeBSD system, it lives in */usr/src/sys/i386/conf*.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 55 FreeBSD Access Control Lists

Unix permissions are flexible and can solve almost any access control problem, but what about the ones they can't?

Do you really want to make a group every time you want to share a file with another user? What if you don't have root access and can't create a group at will? What if you want to be able to make a directory available to a web server or other user without making the files world-readable or -writable? Root-owned configuration files often need to be edited by those without root privileges; instead of using a program like sudo (see [Hack #61] and [Hack #62]), it would be better just to allow certain nonowners to edit these files.

Access Control Lists (ACLs) solve these problems. They allow more flexibility than the standard Unix user/group/other set of permissions. ACLs have been available in commercial Unixes such as IRIX and Solaris, as well as Windows NT, for years. Now, thanks to the TrustedBSD project's work, ACLs are available in FreeBSD 5.0-RELEASE and beyond.

ACLs take care of access control problems that are overly complicated or impossible to solve with the normal Unix permissions system. By avoiding the creation of groups and overuse of root privileges, ACLs can keep administrators saner and servers more secure.

6.3.1 Enabling ACLs

ACLs are enabled by an option in the file system superblock, which contains internal housekeeping information for the file system.

Edit the superblock with the tunefs command, which can be used only on a read-only or unmounted file system. This means that you must first bring the system into single-user mode. Make sure there aren't any active connections to the system, then shut it down:

```
 # shutdown now


*** FINAL System shutdown message from
root@mycompany.com ***

System going down IMMEDIATELY
```

Hack 56 Protect Files with Flags

Ever feel limited when tightening up Unix permissions? Really, there's only so much you can do with r, w, x, s, and t.

When you consider the abilities of the superuser account, traditional Unix permissions become moot. That's not very comforting if you're a regular user wishing to protect your own files or an administrator trying to protect the files on a network server from a rootkit.

Fortunately, the BSDs support a set of extended permissions known as flags. Depending upon your securelevel, these flags may prevent even the superuser from changing the affected file and its flags.

6.4.1 Preventing File Changes

Let's start by seeing what flags are available. Figure 6-1 summarizes the flags, their meanings, and their usual usage.

Table 6-1. Extended permissions flags

Flag name	Meaning	Usage
arch	archive	Forces or prevents a backup
nodump	nodump	Excludes files from a dump
sappnd	system append	Applies to logs
schg	system immutable	Applies to binaries and /etc
sunlnk	system undeletable	Applies to binaries and /etc

Hack 57 Tighten Security with Mandatory Access Control

Increase the security of your systems with MAC paranoia.

Ever feel like your Unix systems are leaking out extra unsolicited information? For example, even a regular user can find out who is logged into a system and what they're currently doing. It's also an easy matter to find out what processes are running on a system.

For the security-minded, this may be too much information in the hands of an attacker. Fortunately, thanks to the TrustedBSD project, there are more tools available in the admin's arsenal. One of them is the Mandatory Access Control (MAC) framework.

> As of this writing, FreeBSD's MAC is still considered experimental for production systems. Thoroughly test your changes before implementing them on production servers.

6.5.1 Preparing the System

Before you can implement Mandatory Access Control, your kernel must support it. Add the following line to your kernel configuration file:

```
options MAC
```

You can find full instructions for compiling a kernel in [Hack #54] .

While your kernel is recompiling, take the time to read man 4 mac, which lists the available MAC modules. Some of the current modules support simple policies that can control an aspect of a system's behavior, whereas others provide more complex policies that can affect every aspect of system operation. This hack demonstrates simple policies designed to address a single problem.

6.5.2 Seeing Other Users

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 58 Use mtree as a Built-in Tripwire

Why configure a third-party file integrity checker when you already have mtree?

If you care about the security of your server, you need file integrity checking. Without it, you may never know if the system has been compromised by a rootkit or an active intruder. You may never know if your logs have been modified and your ls and ps utilities replaced by Trojaned equivalents.

Sure, you can download or purchase a utility such as tripwire, but you already have the mtree utility [Hack #54] ; why not use it to hack your own customized file integrity utility?

mtree lists all of the files and their properties within a specified directory structure. That resulting list is known as a *specification*. Once you have a specification, you can ask mtree to compare it to an existing directory structure, and mtree will report any differences. Doesn't that sound like a file integrity checking utility to you?

6.6.1 Creating the Integrity Database

Let's see what happens if we run mtree against */usr/bin*:

```
 # cd /usr/bin

# mtree -c -K
cksum,md5digest,sha1digest,ripemd160digest -s
123456789 \

          > /tmp/mtree_bin

mtree: /usr/bin checksum: 2126659563
```

Let's pick apart that syntax in Figure 6-2.

Table 6-2. mtree command syntax

Command	**Explanation**
-c	This creates a specification of the current working directory.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 59 Intrusion Detection with Snort, ACID, MySQL, and FreeBSD

How the alert administrator catches the worm.

With the current climate of corporate force reductions and the onslaught of new, fast-spreading viruses and worms, today's administrators are faced with a daunting challenge. Not only is the administrator required to fix problems and keep things running smoothly, but in some cases he is also responsible for keeping the network from becoming worm food. This often entails monitoring the traffic going to and from the network, identifying infected nodes, and loading numerous vendor patches to fix associated vulnerabilities.

To get a better handle on things, you can deploy an Intrusion Detection System (IDS) on the LAN to alert you to the existence of all the nastiness associated with the dark side of the computing world.

This hack will show you how to implement a very effective and stable IDS using FreeBSD, MySQL, Snort, and the Analysis Console for Intrusion Databases (ACID). While that means installing and configuring a few applications, you'll end up with a feature-rich, searchable IDS capable of generating custom alerts and displaying information in many customizable formats.

6.7.1 Installing the Software

We'll assume that you already have FreeBSD 4.8-RELEASE or newer installed with plenty of disk space. The system is also fully patched and the ports collection is up-to-date. It also helps to be familiar with FreeBSD and MySQL commands.

6.7.1.1 Install PHP4, Apache, and MySQL

We'll start by installing PHP4, Apache, and the MySQL client. As the superuser:

```
 # cd /usr/ports/www/mod_php4
```

```
# make install clean
```

When the PHP configuration options screen appears, choose the GD Library Support option. Leave the other default selections, and choose OK.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 60 Encrypt Your Hard Disk

Keep your secrets secret by keeping everything secret.

People often store sensitive information on their hard disks and have concerns about this information falling into the wrong hands. This is particularly relevant to users of laptops and other portable devices, which might be stolen or accidentally misplaced.

File-oriented encryption tools like GnuPG are great for encrypting particular files that will be sent across untrusted networks or stored on disk. But sometimes these tools are inconvenient, because the file must be decrypted each time it is to be used; this is especially cumbersome when you have a large collection of files to protect. Any time a security tool is cumbersome to use, there's a chance you'll forget to use it properly, leaving the files unprotected for the sake of convenience.

Worse, readable copies of the encrypted contents might still exist on the hard disk. Even if you overwrite these files (using rm -P) before unlinking them, your application software might make temporary copies that you don't know about or that have been paged to swapspace. Even your hard disk might have silently remapped failing sectors with data still in them.

The solution is simply never to write the information unencrypted to the hard disk. Rather than taking a file-oriented approach to encryption, consider a block-oriented approach—a virtual hard disk that looks just like a normal hard disk with normal filesystems, but which encrypts and decrypts each block on the way to and from the real disk.

NetBSD includes the encrypting block device driver cgd(4) to help you accomplish this task; the other BSDs have similar virtual devices that, with somewhat different commands, can achieve the same thing. This hack concentrates on NetBSD's cgd.

6.8.1 The Cryptographic Disk Device

To the rest of the operating system, the cgd(4) device looks and behaves like any other disk driver. Rather than driving real hardware directly, it provides a logical function layered on top of another block device. It has a special configuration program, cgdconfig , to create and configure a cgd device and point it at the underlying disk device that

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 61 Sudo Gotchas

Be aware of these limitations when configuring sudo.

sudo is a handy utility for giving out some, but not all root privileges to users of Unix and Unix-like systems. sudo has some limitations and gotchas, however.

> On FreeBSD, build sudo from the ports collection in */usr/ports/security/sudo*.

6.9.1 Limitations of sudo

Tools like sudo exist because the standard Unix privilege model is monolithic. That is, you are either root, with all the privileges and dangers attendant, or you aren't, in which case you lack the ability to affect the system in significant ways. sudo is a workaround of this model. As such, there are limits to what it can achieve, and many of these limitations show up in interactions with the shell. For example:

```
 % sudo cd /some/protected/dir

Password:

sudo: cd: command not found
```

Because a process cannot affect the environment of its parent, cd can't be implemented as a program external to the shell. The command is therefore built into the shell itself. sudo can confer privilege only on programs, not pieces of programs. So, the only way to cd to a protected directory using sudo is to execute the shell itself with sudo:

```
 % sudo bash

# cd /some/protected/dir

# pwd

/some/protected/dir
```

A workaround is to write a script like the following:

```
 #!/usr/local/bin/bash

cd /some/protected/dir;/bin/ls
```

Hack 62 sudoscript

sudo can help enforce strict security policies, but what about situations in which you don't want to restrict what commands your users run?

Maybe you're looking for a way to keep track of what your sysadmin team does as root, so you can quickly find out what happened when something goes wrong. Even if you're the only administrator, it's possible to make a bad error as root without realizing it. An audit trail allows you to go back and see exactly what you did type during that 3:00 AM hacking session.

As mentioned in [Hack #61], giving access to a shell with sudo means that you lose your audit trail the moment the root shell executes. One answer to this problem is sudoscript.

Another scenario where sudoscript is useful is one similar to the situation that caused me to write sudoscript in the first place. I was a sysadmin in a small startup whose engineers all had the root password. The IT crew all used sudo, but they had tried without success to convince the engineers to use it. Upon investigation, I discovered that the principal reason for this was the prohibition on running shells with sudo.

> In fact, the sysadmins used the "everything-but-shells" method the sudoers manpage warns against [Hack #61] .

It quickly became clear that I wasn't going to be able to argue that sudo, as implemented, was equivalent to having a root shell; positions had hardened long before I showed up. So, I wrote sudoscript to bring these engineers back into the IT department's supported circle. It worked, and having the audit trail saved my bacon several times.

6.10.1 sudoscript Overview

sudoscript is a pair of Perl scripts. One is called sudoshell , or just ss. Contrary to its name, sudoshell is not a shell like tcsh or bash. Instead, it is a frontend script that uses authorization from sudo to run as root

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 63 Restrict an SSH server

Control your ssh scripts by placing them in a jail.

Using SSH increases the security of file transfers and network logins. Many network tasks, however, don't really need the shell associated with a user account—remote backups, for example. After all, a shell brings with it commands and an entry point into a system's directory structure. That's somewhat scary when you consider that many of your SSH tasks are scripted.

Configuring a restricted SSH shell such as scponly can mitigate this risk. Not only does it provide noninteractive (read scripted) logins into the SSH server, it limits the set of available commands. Additionally, it provides a chroot option, allowing you to restrict the scponly user account to its own directory structure.

6.11.1 Installing scponly

Before installing this port, read through the available options in its *Makefile*:

```
# cd /usr/ports/shells/scponly

# more Makefile
```

Depending on the scripts you plan on using, consider disabling wildcard processing (which can help prevent accidents like rm -R *). You can also enable rsync support, which is ideal if you're using rsnapshot for backups [Hack #35] . If you want to restrict the account to its own directory, preventing your scripts from accessing anything else on the SSH server, include the chroot option.

Once you've chosen your desired options, pass them to the make command. Here I'll enable chroot support:

```
# make -DWITH_SCPONLY_CHROOT install
```

> If you include the chroot option, do not use the clean target at the end of your make command. make clean will remove the *work/* directory, which contains a script that will set up the chroot for you.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 64 Script IP Filter Rulesets

One firewall ruleset isn't always enough.

As a firewall administrator, you know that it takes a bit of creative genius to create a ruleset that best reflects your network's security needs. Things can get more interesting if those needs vary by time of day. For example, you may need to allow Internet access between business hours but ban it during the evening hours. This is easy to do with two rulebases, a couple of scripts, and trusty old cron.

6.12.1 Limiting Access with IP Filter

I have a FreeBSD firewall/router guarding my home network. I also happen to have a daughter who would spend her life online if she were allowed. There's a simple solution to restricting her access to the Internet to certain times of the day without having to use a proxy.

I use FreeBSD's IP Filter as my firewall software. My normal set of firewall rules, */etc/ipf.rules*, allows unrestricted access to the Internet. Here's the section of that rulebase that controls my daughter's access:

```
 # -------------------------comment area
begin------------------------------

# Internal Interface: ed0

# Allow internal traffic to flow freely.

# -------------------------- comment area end
------------------------------

pass in  on ed0 all

pass out on ed0 all
```

Note that this is not my entire rulebase, just the section controlling the interface, ed0, connected to the portion of the network containing my daughter's computer.

Also note that I did not use the normal pass in quick on ed0 all or pass out quick on ed0 all. This is because the use of the word quick in IP Filter tells the program not to look any further for rules applying to the flow of traffic on an interface. If that were the case, this hack would not work.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 65 Secure a Wireless Network Using PF

Protect your private wireless network from unauthorized use.

The abundance of 802.11 wireless networks has raised an important question. How can you secure a wireless network so that only recognized systems can use it?

Wireless Encryption Protocol (WEP) and MAC access lists offer some protection against unauthorized users; however, they can be difficult to maintain. With OpenBSD's PF, we can maintain tables of recognized clients and update those tables with a single shell command. Known clients can access the Internet; unknown clients will only ever see a web page informing them that this is a private network.

For this hack, we will use dhcpd, PF, and Apache.

6.13.1 DHCP Configuration

We'll use a simple DHCP configuration in */etc/dhcpd.conf* like this:

```
shared-network GUEST-NET {
        max-lease-time 300;
        default-lease-time 120;

        option     domain-name-servers 192.168.0.1;
        option     routers 192.168.0.1;

   subnet 192.168.0.0 netmask 255.255.255.0 {
          range 192.168.0.101 192.168.0.254;
     }
}
```

In this case, we're using the subnet 192.168.0.0/24. Our firewall and NAT gateway is 192.168.0.1, and it's also configured as the DNS server for our network.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 66 Automatically Generate Firewall Rules

Easily protect any FreeBSD workstation with a fully configured firewall.

You know the importance of being protected by a firewall. You know where to look in the manpages for details. Given enough time and trouble, you could write a firewall configuration for any situation. They're all reasonably similar, though, so why not generate the configuration by answering a few questions?

That's the purpose of the IPFilter setup script: to generate configuration rules for typical SOHO firewalls using FreeBSD and IPFilter. Even novice users can retain the full benefits of a firewall without first having to learn syntax. In fact, with this script, you should be able to set up a typical firewall with no FreeBSD configuration knowledge at all.

Even if you're not a novice user, this is a great script to refer friends to as they discover FreeBSD. Now you can rest easy in the thought that your friends are protected—and you didn't even have to find the time to show them how to set up their systems.

6.14.1 What the Script Does

The script uses a simple question and answer text interface. It has four main parts:

Network settings and IPFilter firewall and IPNAT configuration

This configures internal and external network card interface IP address settings either manually or via DHCP. It creates stateful firewall rules on the external network interface and configures NAT to provide Internet connection sharing on the internal network interface.

ADSL PPPOE configuration

This prompts for a login name, password, and Ethernet NIC to generate the */etc/ppp/ppp.conf* file. It then inserts the required PPP variables in */etc/rc.conf*. This starts userland PPP at bootup.

PREV < Day Day Up > NEXT

Hack 67 Automate Security Patches

Keep up-to-date with security patches.

We all know that keeping up-to-date with security patches is important. The trick is coming up with a workable plan that ensures you're aware of new patches as they're released, as well as the steps required to apply those patches correctly.

Michael Vince created quickpatch to assist in this process. It allows you to automate the portions of the patching process you'd like to automate and manually perform the steps you prefer to do yourself.

6.15.1 Preparing the Script

quickpatch requires a few dependencies: perl, cvsup, and wget. Use which to determine if you already have these installed on your system:

```
% which perl cvsup wget

/usr/bin/perl

/usr/local/bin/cvsup

wget: Command not found.
```

Install any missing dependencies via the appropriate port (*/usr/ports/lang/perl5*, */usr/ports/net/cvsup-without-gui*, and */usr/ports/ftp/wget*, respectively).

Once you have the dependencies, download the script from http://roq.com/projects/quickpatch and untar it:

```
% tar xzvf quickpatch.tar.gz
```

This will produce an executable Perl script named quickpatch.pl. Open this script in your favorite editor and review the first two screens of comments, up to the #Stuff you probably don't want to change line.

Make sure that the $release line matches the tag you're using in your cvs-supfile [Hack #80] :

```
# The release plus security patches branch for
FreeBSD that you are

# following in cvsup.

# It should always be a long the lines of RELENG_X_X
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 68 Scan a Network of Windows Computers for Viruses

Regardless of the size of your network, the cost of annual subscriptions for antivirus software can quickly become a pain in the . . . checkbook. Using FreeBSD's strength as a network server, how hard could it be to hack an easier and cheaper way to administer the antivirus battle?

The solution I found uses a combination of FreeBSD and ClamAV and Sharity-Light, both of which are found in the ports collection. As seen in [Hack #19], Sharity-Light can mount Windows shares. Once the shares are mounted, ClamAV will scan them for viruses.

6.16.1 Preparing the Windows Systems

For the systems you wish to virus scan, share their drives as follows:

1.

1. Open My Computer and right-click on the drive you wish to share.

1. Select Sharing from the list of options that appear.

> If Sharing is not available, you will need to activate file sharing in the Network setting in Control Panel. Use Help if you're unsure of where to find this setting.

2.

2. In the Sharing tab of the Properties window, assign a name to the new share. I'll use cdrive in this example. Choose a name that is both useful to you and not already in use. (If a share already exists, click on New Share.)

3.

3. Unless your network is completely closed to the outside world, click on Permissions and limit the access to your user. You should only need read access for scanning purposes.

4.

4. If you need further assistance, search for "sharing" in Windows

Chapter 7. Going Beyond the Basics

Introduction

Have you ever wondered what modifications a web or mail administrator makes to her servers? Maybe you're curious about what policies other administrators use to implement bandwidth control? How do busy administrators manage the log data from a server farm?

Perhaps you've contemplated using the Expect scripting language. However, there's a good chance you've never thought of using eesh, a totally undocumented but useful scripting utility.

This chapter also includes two hacks on the emergency repair process, as many users prefer to hope that they'll never need an emergency repair kit. Instead, learn to overcome your fear of the inevitable and master the art of repairing before the emergency.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 69 Tune FreeBSD for Different Applications

Know how to tune and what to tune on your FreeBSD system

As an administrator, you want to tune your server systems so they work at peak efficiency. How do you know what to tune? The answer depends heavily upon the system's function. Will the system perform a lot of small network transactions? Will it perform a small number of large transactions? How will disk operations factor in?

How you answer these and other questions determines what you need to do to improve the performance of your systems. This hack starts with general optimizations and then looks at function-specific tunables.

7.2.1 Optimizing Software Compiling

A good place to start is with software compiling, as you want to compile software and updates as efficiently as possible. Whenever you compile, your compiler makes assumptions about your hardware in order to create binaries. If you have an x86-compliant CPU, for example, your compiler will create binaries that can run on any CPU from a 386 onward. While this allows portability, it won't take advantage of any new abilities of your CPU, such as the extended MMX, SSE, SSE2, or 3DNow! instruction sets. This is also why using precompiled binaries on your system is a surefire way to reduce your overall performance.

To ensure that software will be compiled efficiently, update your compiler flags in */etc/make.conf* . This file does not exist on new systems, but you can copy it from */usr/share/examples/etc/defaults/make.conf*.

Start by editing the CPUTYPE= line to reflect your CPU type; you'll find supported types listed as comments just before this line. While this will take advantage of your CPU's features, the disadvantage is that your compiled binaries may not run on different CPU types. However, if all of your systems run the same CPU platform, any optimizations you make to shared binaries will affect all of your systems equally well.

Next, change the CFLAGS line to CFLAGS= -O2 -pipe

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 70 Traffic Shaping on FreeBSD

Allocate bandwidth for crucial services.

If you're familiar with your network traffic, you know that it's possible for some systems or services to use more than their fair share of bandwidth, which can lead to network congestion. After all, you have only so much bandwidth to work with.

FreeBSD's dummynet may provide a viable method of getting the most out of your network, by sharing bandwidth between departments or users or by preventing some services from using up all your bandwidth. It does so by limiting the speed of certain transfers on your network—also called *traffic shaping*.

7.3.1 Configuring Your Kernel for Traffic Shaping

To take advantage of the traffic shaping functionality of your FreeBSD system, you need a kernel with the following options:

```
 options IPFIREWALL

options DUMMYNET

options HZ=1000
```

dummynet does not require the HZ option, but its manpage strongly recommends it. See [Hack #69] for more about HZ and [Hack #54] for detailed instructions about compiling a custom kernel.

The traffic-shaping mechanism delays packets so as not to exceed the transfer speed limit. The delayed packets are stored and sent later. The kernel timer triggers sending, so setting the frequency to a higher value will smooth out the traffic by providing smaller delays. The default value of 100 Hz will trigger sends every 10 milliseconds, producing bursty traffic. Setting HZ=1000 will cause the trigger to happen every millisecond, resulting in less packet delay.

7.3.2 Creating Pipes and Queues

Traffic shaping occurs in three stages:

1.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 71 Create an Emergency Repair Kit

The Boy Scout and system administrator motto: "Be prepared!"

As a good administrator, you back up on a regular basis and periodically perform a test restore. You create images [Hack #23] of important servers so you can quickly recreate a system that is taken out of commission.

Are you prepared if a system simply refuses to boot?

Some parts of your drives are as important as your data, yet few backup programs back them up. I'm talking about your partition table and your boot blocks. Pretend for a moment that these somehow become corrupted. The good news is that your operating system and all of your data still exist. The bad news is that you can no longer access them.

Fortunately, this is recoverable, but only if you've done some preparatory work before the disaster. Let's see what's required to create an emergency repair kit.

7.4.1 Inventory of the Kit

When you install a system, particularly a server, invest some time preparing for an emergency. On a FreeBSD system, your kit should include:

-
- The original install CD (or two floppies containing *kern.flp* and *mfsroot.flp* or one floppy containing *boot.flp*)
-
- A floppy containing additional drivers, *drivers.flp*
-
- A fixit floppy, *fixit.flp (*or a CD containing the live filesystem; this will be the second, third, or fourth CD in a set, but not the first CD)
-

A printout of your partition table, */etc/fstab*, and

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 72 Use the FreeBSD Recovery Process

Learn how to use your emergency repair kit before the emergency.

Now that you have an emergency repair kit, it's worth your while to do a dry run so you know ahead of time what options will be available to you. You may even decide to modify your kit as a result of this test.

Let's go back to that sysinstall Main Menu screen [Hack #71] and see what happens when you choose Fixit. You'll be presented with the following options:

```
 Please choose a fixit option

   There are three ways of going into "fixit" mode:

   - you can use the live filesystem CDROM/DVD, in
which case there will be

     full access to the complete set of FreeBSD
commands and utilities,

   - you can use the more limited (but perhaps
customized) fixit floppy,

   - or you can start an Emergency Holographic Shell
now, which is

     limited to the subset of commands that is already
available right now.



   X Exit        Exit this menu (returning to previous)

   2 CDROM/DVD   Use the "live" filesystem CDROM/DVD

   3 Floppy      Use a floppy generated from the fixit
image

   4 Shell       Start an Emergency Holographic Shell
```

If you choose the Shell option, you'll find that they weren't kidding when they warned you'd be limited to a subset of commands. Nearly all of the commands you know and love will result in a *not found* error message. This is why you went to the trouble of either creating that *fixit* floppy or purchasing/burning a CD-ROM/DVD that contains the live filesystem.

7.5.1 Using the fixit Floppy

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 73 Use the GNU Debugger to Analyze a Buffer Overflow

You don't have to be a programmer to use a debugger.

As an end user, you may not realize that you have the ability to analyze security exploits. After all, the organization that distributes your operating system of choice or the provider of a given application will deal with security issues and make updates available.

However, keep in mind that Security Officers apply the same tools and techniques that end users use for debugging programs. Knowing how to analyze a problem will help you to troubleshoot any misbehaving process in a Unix environment.

7.6.1 An Example Exploit

Analyzing a malfunctioning process starts with basic information, such as error messages and return values. Sometimes those aren't enough, though. Some error messages are unclear. In the case of security vulnerabilities, there may not be an error code or return value, because the program may crash or misbehave silently.

The BSDs provide several tools to analyze a program's execution. You can monitor system calls with ktrace and resources with fstat. You can run a debugger such as GDB, the GNU Debugger, and watch your operating system's internal operation.

In some cases, a program must run in a particular environment, which may make it difficult to analyze due to the limitations of some tools. For example, a telnetd advisory from 2001 (http://www.cert.org/advisories/CA-2001-21.html) affected most Unix operating systems. This particular vulnerability came to light when a group called TESO released an example exploit for it.

On Unix systems, telnetd runs as root, so that once the system authenticates the user, the process has the privileges required to set the user ID of the login shell to that of the user who logged in. This means that a remote entity who can cause telnetd to misbehave by sending it carefully designed input could execute processes as root on your system.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 74 Consolidate Web Server Logs

Automate log processing on a web farm.

As the administrator of multiple web servers, I ran across a few logging problems. The first was the need to collect logs from multiple web servers and move them to one place for processing. The second was the need to do a real-time tail on multiple logs so I could watch for specific patterns, clients, and URLs.

As a result, I wrote a series of Perl scripts collectively known as logproc. These scripts send the log line information to a single log host where some other log analysis tool can work on them, solving the first problem. They also multicast the log data, letting you watch live log information from multiple web servers without having to watch individual log files on each host. A primary goal is never to lose log information, so these scripts are very careful about checking exit codes and such.

The basic model is to feed logs to a program via a pipe. Apache supports this with its standard logging mechanism, and it is the only web server considered in this hack. It should be possible to make the system work with other web servers—even servers that can only write logs to a file—by using a named pipe.

I've used these scripts on production sites at a few different companies, and I've found that they handle high loads quite well.

7.7.1 logproc Described

Download logproc from http://www.peterson.ath.cx/~jlp/software/logproc.tar.gz. Then, extract it:

```
 % gunzip logproc.tar.gz

% tar xvf logproc.tar

% ls -F logproc

./    ../    logserver.bin/    webserver.bin/



% ls -F logserver.bin
```

PREV < Day Day Up > NEXT

Hack 75 Script User Interaction

Use an expect script to help users generate GPG keys.

There are occasions when you can take advantage of Unix's flexibility to control some other tool or system that is less flexible. I've used Unix scripts to update databases on user-unfriendly mainframe systems when the alternative was an expensive mainframe-programming service contract. You can use the same approach in reverse to let the user interact with a tool, but with a constrained set of choices.

The Expect scripting language is ideal for creating such interactive scripts. It is available from NetBSD pkgsrc as *pkgsrc/lang/tcl-expect* or *pkgsrc/lang/tk-expect*, as well as from the FreeBSD ports and OpenBSD packages collections. We'll use the command-line version for this example, but keep in mind that expect-tk allows you to provide a GUI frontend to a command-line process if you're willing to write a more complex script.

In this case, we'll script the generation of a GPG key. Install GPG from either *pkgsrc/security/gnupg* or the appropriate port or package.

7.8.1 The Key Generation Process

During the process of generating a GPG key, the program asks the user several questions. We may wish to impose constraints so that a set of users ends up with keys with similar parameters. We could train the users, but that would not guarantee correct results. Scripting the generation makes the process easier and eliminates errors.

First, let's look at a typical key generation session:

```
% gpg --gen-key

gpg (GnuPG) 1.2.4; Copyright (C) 2003 Free Software
Foundation, Inc.

This program comes with ABSOLUTELY NO WARRANTY.

This is free software, and you are welcome to
redistribute it

under certain conditions. See the file COPYING for
details.
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 76 Create a Trade Show Demo

I frequently represent NetBSD at trade shows. It's challenging to attract attention because there are many booths at a show—people will walk by quickly unless something catches their eye. You also need to balance eye-candy with functionality so that you can attract and keep a visitor's attention. I needed an enticing demo to run on one of the computers in the booth.

I wanted to show off several applications, such as office productivity tools, video, and games, and have music playing, but there's only so much screen real estate. Cramming all of those things on the screen at once would clutter the screen, and the point would be lost.

Most X window managers have some concept of virtual desktops, separate work spaces that you can flip between. For example, Enlightenment (*pkgsrc/wm/enlightenment*) not only has the concept of virtual desktops, but as an added bonus for the trade show environment offers a nice sliding effect as you transition from one desktop to the next.

7.9.1 Introducing eesh

Normally in Enlightenment, to switch from one virtual desktop to the next, you move the mouse pointer to the edge of the screen and then push past it, or you use a key sequence to move to an adjacent desktop. For an unattended demo, we need to automate this process. Enlightenment provides an undocumented utility called eesh that can control most aspects of the Enlightenment window manager. You can write scripts to move windows, resize them, or flip between desktops.

Note that eesh isn't a friendly utility; it doesn't even produce a prompt when you run it. Type help for the menu or exit to quit:

```
 % eesh

help

Enlightenment IPC Commands Help

commands currently available:

use "help all" for descriptions of each command

use "help <command>" for an individual description
```

PREV < Day Day Up > NEXT

Chapter 8. Keeping Up-to-Date

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Introduction

One of the distinguishing characteristics of the BSDs is the ease with which you can keep your operating system source and installed software up-to-date. In fact, each of the BSDs provides multiple alternatives, allowing users to choose the approaches that best match their time and bandwidth requirements.

This chapter provides a plethora of ways to maintain an updated system. While many are written from the FreeBSD perspective, don't let that stop you from hacking your own customized NetBSD or OpenBSD solutions. In fact, this chapter concludes with one user demonstrating how to enjoy the benefits of the BSD ports and packages collections on Mac OS X!

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 77 Automated Install

If you're responsible for installing multiple systems, hopefully you've discovered the art of automating installs.

Most operating systems have some sort of scripting mechanism that allows you to predefine the answers to the questions asked by the install program. Once you've started the actual install, you can leave and return to a fully installed system. The alternative is to sit there, answering every prompt when it appears. No, thank you!

Even as a home user, it's well worth your while to spend a few minutes customizing the install script that comes with FreeBSD. Try this hack once and you'll never want to sit and watch an install again.

8.2.1 Preparing the Install Script

Before installing any system, you need to know the following:

-
- The IP settings and hostname of the host you're installing
-
- The FreeBSD name of that host's NIC
-
- Which distributions, or parts of the OS, to install
-
- Your desired partitioning scheme
-
- Which packages (applications) to install

Of course, it's always a good idea to record this information and include it with the documentation for the system.

FreeBSD's install mechanism lives in */stand/sysinstall*. Not surprisingly, man sysinstall describes all of the scriptable bits of this program. I'll go over some useful parameters, but you'll definitely want to skim through the manpage to see if there are additional parameters suited to your particular environment.

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Hack 78 FreeBSD from Scratch

For those who prefer to wipe their disks clean before they upgrade their systems.

Have you ever upgraded your system with make world? If you have only one system on your disks, you may run into a problem: if the installworld fails partway through, you may end up with a broken system that might not even boot. It's also possible that the installworld will run smoothly, but the new kernel will not boot.

What if you're like me and believe in the "wipe your disks when upgrading systems" paradigm? Reformatting ensures there is no old cruft left lying around. It also means you have to recompile or reinstall all your ports and packages and then redo all your carefully crafted configuration tweaks.

FreeBSD From Scratch solves all these problems. The strategy is simple: use a running system to install a new system under an empty directory tree, mounting new partitions in that tree as appropriate. Many config files can copy straight across, and mergemaster can take care of those that cannot. You can perform arbitrary post-configuration of the new system from within the old system, up to the point where you can chroot to the new system.

This upgrade has three stages, where each stage either runs a shell script or invokes make:

stage_1.sh

Creates a new bootable system under an empty directory, merges or copies as many files as are necessary, and then boots the new system

stage_2.sh

Installs your desired ports

stage_3.mk

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 79 Safely Merge Changes to /etc

Use a three-way merge to deal with upgraded configuration files.

Even though you probably run cvsup on a daily basis, you likely run make world only a few times a year, whenever a new version of the OS is released. The steps required to upgrade your system are well documented and fairly straightforward. That is, it's easy until it's time to run mergemaster.

mergemaster is an important step, as it integrates changes to */etc*. For example, occasionally a core utility such as Sendmail will require a new user or group in */etc/passwd*. Problems can occur if those changes aren't integrated.

If you've used mergemaster before, you know it's not the most user-friendly utility out there. Misinterpret a diff, and you might lose your configuration file changes or, worse, miss a necessary change. You might even end up blowing away your own users in */etc/passwd* —not the most convenient way to start off a new upgrade.

8.4.1 Initial Preparations

An alternative is to use etcmerge (*/usr/ports/sysutils/etcmerge*). This utility does most of the work for you. Unlike the two-way diff used by mergemaster, this utility can compare the changes between three sets of edits:

- The */etc* from your original version of FreeBSD
- Any changes you've made to */etc* since then
- The */etc* for your new version of FreeBSD

> Before any upgrade, you definitely want a fresh, tested backup of all of your data, including */etc*.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 80 Automate Updates

FreeBSD provides many tools to make software upgrades as painless as possible. In fact, the entire process is fully scriptable. Simply choose the pieces you want and how up-to-date you want to be.

End users and administrators alike share a desire to keep their operating systems and applications as up-to-date as possible. However, if you're an operating systems veteran, you're well aware that this desire doesn't always translate into foolproof, easy execution. For example, do you have to scour the far corners of the Internet to find the latest updates? Once you find them, is it possible to upgrade safely without overwriting the dependencies required by other applications?

8.5.1 Assembling the Pieces

The cvsup process provides the latest updates to the FreeBSD operating system, ports collection, and documents collection. You no longer have to scour the Internet looking for the latest sources. Simply run cvsup!

Since our intention is to script the whole process, install the *cvsup-without-gui* port:

```
 # cd /usr/ports/net/cvsup-without-gui

# make install clean
```

If you've never used cvsup before, take the time to read its section in the FreeBSD Handbook so you have an overview of how the process works.

When the install finishes, copy */usr/share/examples/cvsup/cvs-supfile* to a location that makes sense to you (e.g., */root* or */usr/local/etc*). Use the comments in that file and the instructions in the handbook to customize the file so it reflects your closest mirror, operating system (tag), and what you would like to update.

Here's my *cvs-supfile*. It uses a Canadian mirror and updates all sources, ports, and documents on a FreeBSD 5.1-RELEASE system:

```
 # more /root/cvs-supfile

#use the Canadian mirror

*default host=cvsup.ca.freebsd.org
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 81 Create a Package Repository

Combine the advantages of compiling from source and installing packages.

We saw in [Hack #69] that compiling applications from source, i.e., by making their ports, has several advantages. You can tune */etc/make.conf* to take advantage of your architecture. You can also customize the installation by passing various arguments to make.

However, if you're responsible for maintaining software on multiple machines, do you always want to install from source? If your systems run similar hardware, why not create your own customized packages on one machine and make them available to your other systems via a package repository?

Creating your own custom packages allows you to retain all the benefits of make. Even better, the resulting package installs the desired software very quickly. This can be a real time-saver when you maintain multiple systems.

> The experienced hacker may prefer to use */usr/ports/devel/distcc* to provide multiple builds.

8.6.1 Creating Custom Packages

Pick a machine in your network to contain the package repository, and install the ports collection on that system. The rest of your systems won't need the ports collection, which saves their disk space for other purposes.

On the system containing the ports collection, create a directory to store the packages:

```
# mkdir /usr/ports/packages
```

Then, decide which packages you'd like to create. I'll start with Exim. Before creating the package, I'll search through the port's *Makefile* to see if there are any make options:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 82 Build a Port Without the Ports Tree

While the ports tree is one of the most useful FreeBSD directory structures, you may have systems where it's not appropriate to maintain the entire ports structure.

On some of your systems, disk space may be an issue. The ports tree tarball itself is a 21 MB download. Once untarred, it will occupy around 500 MB of disk space. That space will continue to grow as you install ports since, by default, source files download into */usr/ports/distfiles*.

Does this mean that installing packages is your only alternative? Packages are convenient, but since they are precompiled, you don't have the option of providing your own make arguments to optimize the install for your environment.

One alternative is the anonymous CVS system. Even a minimal install of FreeBSD includes the cvs command. This allows you to check out only the particular port skeleton you need. You'll still have the convenience of the ports collection without actually having to install it.

8.7.1 Connecting to Anonymous CVS

The first time you use cvs, create an empty CVS password file, as CVS will complain if this file is missing:

```
# touch ~root/.cvspass
```

Then, ensure your present working directory is */usr*:

```
# cd /usr
```

When using cvs to maintain your ports, be sure you are in */usr*. cvs downloads the requested files to your current working directory and will overwrite any files of the same name.

Then, use the cvs login command to connect to a CVS server. There are five FreeBSD anonymous CVS servers; see the Handbook

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 83 Keep Ports Up-to-Date with CTM

Keep your ports up-to-date without using cvsup.

If you have a slow Internet connection, it can take a while to download the ports tree; the current tarball is over 21 MB in size. Once you have the ports collection, keeping up-to-date with cvsup might not be such an attractive option if it involves tying up your phone line.

Perhaps bandwidth isn't the problem. Perhaps you're just looking for an alternative way to stay current, without having to install and configure cvsup. After all, why install additional software if you can achieve the same results using commands that come with the base system?

Regardless of which category you fall into, CTM may be what you're looking for.

CTM was originally CVS Through Email, meaning you could receive the changes you usually receive through cvsup via email. (In the case of numerous changes, you'd receive several, smaller mails instead of one monolithic message.) This can be a cheaper alternative to cvsup if you're charged for the amount of time you are connected to the Internet.

However, it's even easier to retrieve these changes with ftp. FreeBSD maintains several CTM servers that contain the changes, or deltas, to the FreeBSD source and the ports collection. This hack will concentrate on keeping your ports up-to-date using ftp and the CTM servers.

8.8.1 Using ftp and ctm to Stay Current

Let's start with a system that doesn't have the ports collection installed. First, I'll create an empty *ports* directory for ctm to work with:

```
 # mkdir /usr/ports/

# cd /usr/ports
```

Then, instead of downloading and untarring the ports tree tarball, I'll ftp into a CTM server and download the latest ports tree delta. The Handbook's section on CTM includes the addresses of the CTM mirrors.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 84 Navigate the Ports System

Use built-in commands to keep abreast of the FreeBSD ports collection.

What first attracted me to FreeBSD—and what has definitely kept my attention since—is the ports collection. Over 10,000 applications are a mere make install clean away. For a software junkie like myself, it is indeed Nerdvana to no longer scour the Internet for software or fight my way through dependency hell just to convince an application to install.

Admittedly, it's easy to get lost in a sea of ports. How do you choose which application best suits your needs? How do you keep track of which ports have been installed on your system? How do you make sure you don't inadvertently delete a dependency? Read on to see how to get the most out of the built-in utilities for managing ports.

8.9.1 Finding the Right Port

You know you want to install some software to add functionality to your system. Wouldn't it be great if you could generate a list of all the ports that are available for your specific need? Well, you can, and it's almost too easy with the built-in port search facility. In this example, I'll look for ports dealing with VPN software:

```
 % cd /usr/ports

% make search key=vpn | more

Port:         poptop-1.1.4.b4_2

Path:         /usr/ports/net/poptop

Info:         Windows 9x compatible PPTP (VPN) server

Maint:         ports@FreeBSD.org

Index:         net

B-deps:         expat-1.95.6_1 gettext-0.12.1
gmake-3.80_1 libiconv-1.9.1_3

R-deps:

<snip>
```

I snipped the results for brevity as this command gives the details of each port associated with VPNs. The format of the output is quite

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 85 Downgrade a Port

It doesn't happen often, but occasionally portupgrade will upgrade a port to a newer version that doesn't sit well with your system.

It can be very frustrating when an application that was working just fine an hour ago suddenly stops working after an upgrade. Now what?

At first glance, the solution isn't obvious. Because ports don't contain revision labels, you can't just cvsup back to an earlier version. However, the commits or changes to each port are tracked in the CVS repository. You could learn the syntax of the cvs command and use it to connect to the CVS repository, manually review the port's commit history, find an earlier version that worked on your system, check out that version, and rebuild the port. Whew! There must be an easier way.

That's what Heiner Eichmann thought when he created portdowngrade . His script does all of the work for you; you only need to choose which version of the port to use.

8.10.1 Using portdowngrade

Installing portdowngrade is easy enough:

```
 # cd /usr/ports/sysutils/portdowngrade
# make install clean
```

A few moments later, you'll have the script and an informative manpage. To run the script, simply specify which port you'd like to downgrade. Here, I'll demonstrate an arbitrary port:

```
 # portdowngrade apinger
portdowngrade 0.1 by Heiner Eichmann
Please note, that nothing is changed in the ports tree
unless it is explicitly permitted in step 6!

Seeking port apinger ... found: net/apinger

Step 1: Checking out port from CVS repository
CVS root directory:
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 86 Create Your Own Startup Scripts

Ensure your favorite installed applications start at boot time.

Some ports are nice enough to create their own startup scripts in */usr/local/etc/rc.d* when you install them. Unfortunately, not all ports do. You may wonder why you're not receiving any email, only to discover a week later that your mail server didn't start at your last bootup!

In those cases, you'll have to write your own startup script. Fortunately, that's easy.

8.11.1 Was a Script Installed?

Every port comes with a packing list of installed executables, files, and manpages. To see if a particular port will install a startup script, search its *pkg-plist* for the word rc. Here, I'll check the packing lists for the stunnel and messagewall ports:

```
 % grep -w rc /usr/ports/security/stunnel/pkg-plist

etc/rc.d/stunnel.sh.sample



% grep -w rc /usr/ports/mail/messagewall/pkg-plist

%
```

Use the -w switch so grep searches for the full word rc, not just words containing those two characters. If there isn't a startup script, as is the case for messagewall, you'll just get your prompt back.

If the startup script ends with *.sample*, you'll need to copy it to a new file without that extension. This is often the case with applications that expect you to change the sample configuration file to suit your system's requirements.

Also, note the relative path. The packing list knows that, by default, the files installed by a port will start with the prefix */usr/local*. That is, in the previous example, you'll find stunnel's startup script in */usr/local/etc/rc.d*, not in */etc/rc.d*.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 87 Automate NetBSD Package Builds

Use a sandbox to build applications that play nicely within your network.

Many NetBSD users are responsible for multiple systems running on different architectures. Instead of rebuilding the same package on machine after machine, it's often desirable to build packages for all of these machines from the most powerful one, delivering the appropriate binary packages across the network. However, problems can arise when not all machines run the same version of NetBSD or when you want different optimizations or build settings on each box.

The solution to this dilemma is simple: create a sandbox with the version of NetBSD used in the target machine and build the necessary binary packages inside it. This sounds easy, but it can be a very tedious and error-prone task. It is even more complex if you want to automate periodic package rebuilding. Fortunately, that's our final goal in this hack.

To simplify things, I assume that you have a relatively fast desktop machine running NetBSD-current, where you will build binary packages, and a server machine running the stable version of NetBSD (1.6.2 at the time of this writing).

8.12.1 Installing pkg_comp

pkg_comp (also known as Package Compiler) can simplify the creation of these sandboxes: it handles any version of NetBSD inside a chroot jail and automates the build process of binary packages inside it. Its only restriction is that both the builder and the destination machine share the same architecture.

Let's begin by installing pkg_comp on the builder machine (make sure you have Version 1.15 or greater):

```
 # cd /usr/pkgsrc/pkgtools/pkg_comp

# make install && make clean
```

After installation, spend some time reading man 8 pkg_comp and getting familiar with its structure because you will be using it as a reference guide during the configuration. Also ensure that your kernel

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 88 Easily Install Unix Applications on Mac OS X

Many Mac users often seem a little surprised when I tell them I run XChat and other Unix applications on Mac OS X alongside native Aqua applications (such as Safari, Finder, and iPhoto). What they don't realize is that it's simple to install such applications thanks to the Fink and DarwinPorts projects. This hack is dedicated to installing and using DarwinPorts.

This hack assumes you have a basic understanding of *Terminal.app* and the underlying Unix bits of Mac OS X. You also need to have the Developer Tools installed.

8.13.1 Installing DarwinPorts

Before you can use DarwinPorts, you must install the build system and the actual ports tree. The easiest way to accomplish this is by using CVS. Before checking the project out of CVS, you'll need to decide where you'd like it to exist on your hard drive. I usually use *~/work*.

Open *Terminal.app* (or an xterm if you have X11 installed), and change to the directory where you'll install DarwinPorts. Then type the following commands at the prompt (when the server asks for a password, just press Return):

```
 % alias dcvs cvs -d \


:pserver:anonymous@anoncvs.opendarwin.org:/Volumes/src
/cvs/od

% dcvs login

% dcvs co -P darwinports
```

You should now see a bunch of output scrolling past in the terminal window. If you do, good; the project is checking out of CVS and onto your hard disk. If you don't, double-check the three commands just shown to make sure you typed everything correctly. Once you've fetched the project, it's time to install it.

Run ls in the terminal window; you should see a *darwinports* directory. cd to it and rerun ls:

```
 % cd darwinports
```

PREV [< Day Day Up >] NEXT

PREV [< Day Day Up >] NEXT

Chapter 9. Grokking BSD

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Introduction

Heinlein fans will recognize the word grok as the Martian word for "to be one with" or "thorough understanding." Indeed, you will sometimes feel like a stranger in a strange land when learning Unix. As any Unix guru can attest, however, the rewards far outweigh the initial learning curve.

This final chapter is a hodgepodge of useful and sometimes amusing tidbits. A sure sign you're on the right road to grokking BSD is when you're able to see both the usefulness and the quirky humor that is inherent in all Unix systems.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 89 How'd He Know That?

Make the most of your available resources.

Unless you've achieved Unix guru status, you probably find yourself asking "how did he know that?" whenever you're around other Unix users or read a really cool snippet in a book. Here's a little secret: he probably had to look it up. As I tell my students, "No one knows everything. Make sure the one thing you do know is where to go to get the information you need."

9.2.1 Online Resources

If you're using FreeBSD, there is no shortage of well-written documentation. If you haven't already, bookmark the FreeBSD Documentation page at http://www.freebsd.org/docs.

There you'll find hyperlinks to the four handbooks, the FAQ, how-to articles, online manpages, as well as other sources of information. There's a very good chance that someone else has already documented what you want to do.

9.2.2 Keeping Offline Resources Up-to-Date

Online resources are great, but what if you don't always have access to an Internet connection? If you installed the doc distribution, you already have most of those resources on your hard drive. You'll find the handbooks, FAQ, and articles in */usr/share/doc*. That directory contains symlinks so you can quickly navigate to the desired resource.

> If you haven't installed the doc directory structure, you can do so through /stand/sysinstall. Enter Configuration, then Distributions, and use your spacebar to select doc.

The online resources receive daily updates, so be sure to update your docs when you use cvsup. Make sure your *cvsup* file includes this line:

```
doc-all tag=.
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 90 Create Your Own Manpages

As a Unix administrator, the one word of sage advice you can give to any user that is guaranteed to solve any problem is RTFM.

What's an administrator to do when informed by a user that there is no manpage to read? Perhaps the application in question is a custom application or script, or perhaps it's a third-party program that didn't come with a manpage. Why not create the missing manual yourself?

9.3.1 Manpage Basics

Creating a manpage isn't all that difficult. After all, a manpage is simply a text file—more specifically, a gzipped text file sprinkled with groff macros. (I'm quite sure groff gets its name from the choking sound you make as you try to decipher its manpage.) For man to do its magic, which starts with being able to find the page, the manpage must live in a directory manpath can see.

Not surprisingly, manpath's configuration file, */etc/manpath.config*, contains those paths:

```
 % grep MAP /etc/manpath.config

# MANPATH_MAP           path_element
manpath_element

MANPATH_MAP            /bin
/usr/share/man

MANPATH_MAP            /usr/bin
/usr/share/man

MANPATH_MAP            /usr/local/bin
/usr/local/man

MANPATH_MAP            /usr/X11R6/bin
/usr/X11R6/man
```

Basically, manpages to programs that come with the system live in */usr/share/man*, third-party applications use */usr/local/man*, and X applications use */usr/X11R6/man*. If you ls any of these directories, you'll find directory names that go from *man1* to *man9*. If you're rusty on the function of each manpage section, run:

```
 % whatis intro

intro(1)                 - introduction to general
commands (tools and

                              utilities)
```

Hack 91 Get the Most Out of Manpages

Now that you know how to create your own manpages, you'll want to know how to get the most out of your manpage viewing.

Since most documentation on Unix systems lives within manpages, it pays to know how to get the most out of your manpage-reading experience. How do you make sure you're aware of all of the manpages installed on a system? How do you zero in on the information you need, without having to read an entire manpage? Yes, it's a great experience to read all of man tcsh at least once in your life, but you don't want to do that when you're only interested in a certain shell variable.

9.4.1 Finding Installed Manpages

You may have noticed that, by default, whatis [Hack #13] doesn't find custom manpages or those installed by third-party applications. Not only is this inconvenient, but it can also prevent your users from getting the most out of the applications installed on a system.

Remember */etc/manpath.config* from [Hack #90] ?

```
 % grep MAP /etc/manpath.config

# MANPATH_MAP     path_element         manpath_element

MANPATH_MAP       /bin                 /usr/share/man

MANPATH_MAP       /usr/bin             /usr/share/man

MANPATH_MAP       /usr/local/bin       /usr/local/man

MANPATH_MAP       /usr/X11R6/bin       /usr/X11R6/man
```

The makewhatis command actually creates the whatis database and, by default, makewhatis reads only */usr/share/man*. It'll skip any manpages in */usr/local/man* and */usr/X11R6/man*, because it doesn't know they exist!

To gather in those missing manpages, pass these extra directories to makewhatis:

```
 # makewhatis /usr/local/man /usr/X11R6/man

#
```

Hack 92 Apply, Understand, and Create Patches

Sometimes only the little differences matter.

Despite all your best efforts, eventually you'll end up with multiple versions of a file. Perhaps you forgot to keep your *.vimrc* in sync between two machines [Hack #10] . Alternatively, you may want to see the changes between an old configuration file and the new version. You may even want to distribute a bugfix to a manpage or program.

Sending the entire changed file won't always work: it takes up too much space and it's hard to find exactly what changed. It's often easier and usually faster to see only the changes (see [Hack #80] for a practical example). That's where diff comes in: it shows the differences between two files.

As you'd expect, applying changes manually is tedious. Enter patch, which applies the changes from a diff file.

9.5.1 Finding Differences

Suppose you've shared a useful script with a friend and both of you have added new features. Instead of printing out both copies and marking differences by hand or, worse, trying to reconcile things by copying and pasting from one program to another, use diff to see only the differences between the two programs.

For example, I've customized an earlier version of the *copydotfiles.pl* script from [Hack #9] to run on Linux instead of FreeBSD. When it came time to unify the programs, I wanted to see the changes as a whole. diff requires two arguments, the source file and the destination. Here's the cryptic (at first) result:

```
 $ diff -u copydotfiles.pl copydotfiles_linux.pl

--- copydotfiles.pl        2004-02-23
16:09:49.000000000 -0800

+++ copydotfiles_linux.pl        2004-02-23
16:09:32.000000000 -0800

@@ -5,8 +5,8 @@

 #    - change ownership of those files
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 93 Display Hardware Information

If you're new to FreeBSD, you may be wondering where to find information about your system's hardware and the resources it uses.

You've probably noticed that your FreeBSD system didn't ship with a Microsoft-style Device Manager. However, it does have plenty of useful utilities for gathering hardware information.

9.6.1 Viewing Boot Messages

When you boot your system, the kernel probes your hardware devices and displays the results to your screen. You can view these messages, even before you log in, by pressing the scroll lock key and using your up arrow to scroll back through the message buffer. When you're finished, press scroll lock again to return to the login or command prompt.

You can type dmesg any time you need to read the system message buffer. However, if it's been a while since bootup, it's quite possible that system messages have overwritten the boot messages. If so, look in the file */var/run/dmesg.boot*, which contains the messages from the latest boot. This is an ASCII text file, so you can send it to a pager such as more or less.

You may find it more convenient to search for something particular. For example, suppose you've added sound support to your kernel by adding device pcm to your kernel configuration file. This command will show if the PCM device was successfully loaded by the new kernel:

```
 % grep pcm /var/run/dmesg.boot

pcm0: <Creative CT5880-C> port 0xa800-0xa83f irq 10
at device 7.0 on pci0

pcm0: <SigmaTel STAC9708/11 AC97 Codec>
```

In this example, the kernel did indeed probe my Creative sound card at bootup.

9.6.2 Viewing Resource Information

Sometimes you just want to know which devices are using which system resources. This command will display the IRQs, DMAs, I/O

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 94 Determine Who Is on the System

As a system administrator, it pays to know what's happening on your systems.

Sure, you spend time reading your logs, but do you take advantage of the other information-gathering utilities available to you? Silently, in the background, your system tracks all kinds of neat information. If you know enough to peek under the system hood, you can get a very good view of what is occurring on the system at any given point in time.

> For the experienced hacker, the output from these commands may suggest interesting scripting possibilities.

9.7.1 Who's on First?

Have you ever needed to know who logged into a system and for how long? Use the users command to see who's logged in now:

```
 % users

dru biko
```

Perhaps you prefer to know who is on which terminal. Try who. Here, the H includes column headers and the u shows each user's idle time:

```
 % who -Hu

NAME             LINE     TIME         IDLE   FROM


dru              ttyv1    Jan 25 08:59 01:00

biko             ttyv5    Jan 25 09:57   .

dru              ttyp0    Jan 25 09:58 00:02
(hostname)
```

Feel free to experiment with who's switches to find an output that suits your needs. Here, dru and biko have logged in physically at this system's keyboard using virtual terminals 1 and 5. dru has also logged in over the first psuedoterminal (over the network) from the specified hostname.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 95 Spelling Bee

For those who edit their text at the command line.

Like most computer users, you probably find yourself spending a fair bit of time typing, whether responding to email, navigating the web, or working on that résumé or thesis. How often do you find yourself looking at a word, wondering if you've spelled it correctly? How often do you rack your brain trying to find a more interesting or descriptive word?

You've probably discovered that Unix doesn't come with a built-in dictionary or thesaurus. Sure, you can install a feature-rich GUI office suite, but what alternatives are there for users who prefer less bloat on their systems or are accessing systems from the command line?

9.8.1 Quick Spellcheck

If you're in doubt about the spelling of a word, try using look. Simply include as much of the word as you're sure about. For example, if you can't remember how to spell "bodacious" but you're pretty sure it starts with "boda":

```
% look boda

bodach

bodacious

bodaciously
```

If you don't have access to a GUI, see [Hack #12] .

I find look especially helpful with suffixes. It's very handy if you can't remember when to use "ly", "ally", or "ily". For example:

```
% look mandator

mandator

mandatorily

mandatory
```

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 96 Leave on Time

Use your terminal's built-in timers and schedulers.

You know how it is. You sit down in front of a keyboard and quickly become absorbed in your work. At some point you remember to look up, only to notice that everyone else is gone for the day. If that doesn't describe you, I bet you can think of at least one person it does describe.

9.9.1 Don't Forget to Leave

Fortunately the leave command can save you from the embarrassment of forgetting important appointments. Use it at any time by typing:

```
 % leave

When do you have to leave?
```

There are three ways to respond to that question:

-
- Press Enter to abort.
-
- Type *hhmm*, where *hh* represents the hour and *mm* represents the minute.
-
- Type +*number*, where *number* represents how many hours or minutes from now you'd like to leave.

For example, to leave at 5 PM:

```
 % leave 500

Alarm set for Tue Dec 30 17:00:00 EST 2003. (pid
50097)
```

leave 1700 will achieve the same results.

Or, to leave in 45 minutes:

```
 % leave +45

Alarm set for Tue Dec 30 9:52:00 EST 2003. (pid 50108)
```

Be sure to include the + if you're not specifying an actual time.

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

Hack 97 Run Native Java Applications

Until recently, running Java applications on FreeBSD meant using the Linux compatibility mode.

Linux programs can sometimes be problematic on FreeBSD. Java uses threading very heavily, and that's probably the poorest-emulated part of Linux binary compatibility. Some Java applications or class libraries just don't work correctly under Linux emulation. Native versions of the Java distribution had restrictive licenses, and it required a great deal of work to download and compile them. Fortunately, the FreeBSD Foundation has negotiated a FreeBSD Java license with Sun Microsystems. This hack demonstrates how to configure the FreeBSD version of Java.

> What about native Java on NetBSD or OpenBSD? At the time of writing, neither system had a native Java port. You can run Java on a Linux emulator or via Tomcat.

9.10.1 Choosing Which Java Port to Install

The first requirement for running Java applications is a Java Virtual Machine (JVM) and the associated runtime support libraries. There are several Java Runtime Environments (JREs) or Java Development Kits (JDKs) available in ports.

> A JRE contains everything necessary for an end user to run Java applications. A JDK contains all that, plus various extra bits required for developing, compiling, and debugging Java code.

The main criteria for choosing a port are:

-

Hack 98 Rotate Your Signature

End your email communications with a short witticism.

We all seem to know at least one geek friend or mailing-list poster whose emails always end with a different and humourous bit of random nonsense. You may be aware that this is the work of her *~/.signature* file, but have you ever wondered how she manages to rotate those signatures?

While there are several utilities in the ports collection that will randomize your signature, it is easy enough to roll your own signature rotator using the fortune program and a few lines of shell scripting.

9.11.1 If Your Mail Program Supports a Pipe

Your approach will vary slightly, depending on whether your particular mail user agent (MUA) supports pipes. If it does, it's capable of interpreting the contents of a file as command output, just like when you use a pipe (|) on the command line.

I use pine, which supports both static signature files and signatures that come from the piped output of a signature rotation program.

When configuring pine, choose Setup from the main menu, then C for the configuration editor. Find the signature-file option and give it this value:

```
 .signature |
```

The pipe character tells pine to process that filename as a program instead of inserting its contents literally.

Also enable the signature-at-bottom option found in the Reply Preferences to ensure your signature is placed at the bottom of your emails, even when replying to an email.

Next, create a file called *~/.signature* containing these lines:

```
 echo "Your random fortune:"

/usr/games/fortune -s
```

Hack 99 Useful One-Liners

Unix is amazing. Only your imagination limits the usefulness of the built-in commands. You can create your own commands and then pipe them together, allowing one utility to work on the results of another.

If you're like me, you've run across dozens of useful combinations over the years. Here are some of my favorite one-liners, intended to demonstrate useful ideas as well as to prime your pump for writing your own one-liner hacks.

9.12.1 Simultaneously Download and Untar

Have you ever downloaded an extremely large archive over a slow connection? It seems to take forever to receive the archive and forever to untar it. Being impatient, I hate not knowing how many of the archived files are already here. I miss the ability to work on those files while the rest of the archive finishes its slow migration onto my system.

This one-liner will decompress and untar the files as the archive downloads, without interfering with the download. Here's an example of downloading and untarring the ports collection:

```
 # tail -f -b=1m ports.tar.gz | tar -zxvf ports.tar.gz

ports/

ports/Mk/

<snip>
```

Here I've asked tail to stream up to one megabyte of the specified file as it is received. It will pipe those bytes to the tar utility, which I've directed to decompress (-z) and to extract (x) the specified file (f) while displaying the results verbosely (v).

To use this command, download the archive to where you'd like to untar it—in this example, */usr*. Simply replace the filename *ports.tar.gz* with the name of your archive.

9.12.2 When Did I Change That File?

Do you ever need to know the last modification date of a file? Consider a long listing:

PREV < Day Day Up > NEXT

PREV < Day Day Up > NEXT

9.13 Fun with X

Use the utilities that come with the core X distribution.

There are so many GUI utilities, available either as part of your favorite Window Manager or as a separate installation, that you can forget that the core X distribution also provides several useful and lightweight programs. Do you need to monitor console messages, manage your clipboard, send pop-up messages, or create and view screenshots? Before you hit the ports collection, give the built-in utilities a try.

9.13.1 Seeing Console Messages

In [Hack #42], we saw how to redirect console messages. If you're using an X session, the xconsole utility fulfills this purpose. To start this utility, simply type its name into an xterm or use the Run command provided by your window manager.

By default, only the superuser can start xconsole. A regular user will instead receive a Couldn't open console message. This is a safety precaution on multiuser systems, preventing regular users from viewing system messages. If you're the only user who uses your system, remove the comment (#) from this line in */etc/fbtab*:

```
#/dev/ttyv0    0600    /dev/console
```

If you spend a lot of your time at an X session, consider adding xconsole to your *~/.xinitrc* file so it will start automatically (see [Hack #9]).

9.13.2 Managing Your Clipboard

If you do a lot of copying and pasting, xclipboard is another excellent candidate for automatic startup. This utility stores each of your clipboard selections as a separate entity, allowing you to scroll through them one at a time in a simple GUI window. In addition to the Next and Prev buttons, a Delete button lets you remove unwanted items and a Save button allows you to save all of your items as a file.

9.13.3 Sending Pop-up Messages

Do you find yourself starting a command that takes a while to execute, continuing your work in an X session, then returning periodically to the

PREV < Day Day Up > NEXT

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O
] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O
] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV < Day Day Up >

PREV < Day Day Up >

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O
] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

PREV