2ND EDITION

ABSOLUTE OPENBSD

UNIX FOR THE PRACT*ICA*L PARANOID

MICHAEL W. LUCAS

"The definitive book on OpenBSD gets a long-overdue refresh." ***-Theo de Raadt, OpenBSD Founder***

no starch press

AdvAnce PrAise for *ABSOLUTE OPENBSD, 2ND EDITION*

"Michael W. Lucas's books are good enough to raise national productivity statistics. Every copy of OpenBSD should be bundled with this book." —RICHARD BEJTLICH, CSO OF MANDIANT, TAOSECURITY BLOGGER, AND AUTHOR OF *the practice of network security monitoring*

"After 13 years of using OpenBSD, I learned something new and useful!" —PETER HESSLER, OPENBSD JOURNAL (UNDEADLY.ORG)

"The OpenBSD world, myself included, has been waiting for an update to *Absolute OpenBSD* for years. Michael W. Lucas tackles OpenBSD topics in ways that are bound to inspire the learner and warm the hearts of Unix greybeards."

—PETER N.M. HANSTEEN, AUTHOR OF *the book of pf*

"Michael W. Lucas is a layperson's tutor, sitting next to you in front of an OpenBSD box and working through the same issues the average sys admin does." —GEORGE ROSAMOND, FOUNDING MEMBER OF THE NYC*BSD USER GROUP

"Whether you are an experienced OpenBSD user seeking a functional desk reference or a new OpenBSD user seeking to gain the knowledge necessary to become an expert, then *Absolute OpenBSD* is the book you have to have." —CHRIS SANDERS, AUTHOR OF *practical packet analysis*

"The second edition of *Absolute OpenBSD* delivers an updated tour of OpenBSD with great attention to detail and zero handwaving. Mr. Lucas and No Starch Press have once again demonstrated exemplary respect and loyalty to OpenBSD and the BSD community." —MICHAEL DEXTER, CALLFORTESTING.ORG

Absolute openbsD

2nD eDition

unix for the practical paranoid

by Michael W. Lucas

San Francisco

Printed in USA
First printing
17 16 15 14 13 1 2 3 4 5 6 7 8 9
ISBN10: 1593274769 ISBN13: 9781593274764
Publisher: William Pollock Production Editor: Alison Law Cover Illustration: Charlie Wylie Interior Design: Octopod Studios Developmental Editor: William Pollock Technical Reviewer: Peter N.M. Hansteen Copyeditor: Marilyn Smith Compositor: Susan Glinert Stevens Proofreader: Elaine Merrill Indexer: Nancy Guenther
For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly:
No Starch Press, Inc. 38 Ringold Street, San Francisco, CA 94103 phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; www.nostarch.com
Library of Congress has cataloged the first edition as follows:
Lucas, Michael W., 1967-
Absolute OpenBSD: UNIX for the practical paranoid / Michael W. Lucas. p. cm. Includes index.
ISBN: 1-886411-99-9 1. OpenBSD (Electronic resource). 2. Operating systems (Computers) 3. UNIX (Computer file) I. Title.
QA76.9.063L835 2003 005.4'32--dc21
2003000473

For Liz

About the Author

Michael W. Lucas is a network/security engineer who keeps getting stuck with the problems nobody else wants to touch. He's been using BSD since the days it came from Berkeley, and today uses OpenBSD for mission-critical infrastructure. You can find Lucas roaming around Detroit, Michigan, or teaching tutorials at tech conferences. He's the author of the critically acclaimed *Absolute FreeBSD*, *Network Flow Analysis*, *Cisco Routers for the Desperate*, and *PGP & GPG*, all from No Starch Press. Find his website and blog at *http://www.michaelwlucas.com/*, or follow @mwlauthor on Twitter.

About the Technical Reviewer

Peter N.M. Hansteen is a consultant, sysadmin, and writer from Bergen, Norway. During recent years he has been a frequent lecturer and tutor with emphasis on OpenBSD and FreeBSD, as well as the author of sev- eral articles and *The Book of PF* (No Starch Press, 2010). He writes about OpenBSD and rants about other IT topics at *http://bsdly.blogspot.com/*.

Brief Contents

Contents in Detail

4 POST-INSTALL SETUP 57

5 THE BOOT PROCESS 69

6 USER MANAGEMENT 85

17 DESKTOP OPENBSD 323

18 KERNEL CONFIGURATION 339

19 BUILDING CUSTOM KERNELS 355

Foreword

I got my OpenBSD account as a developer in 2002, more than 10 years ago. Over this time, quite a num- ber of OpenBSD-related books have been published. Some were actually good, but many were not and were full of factual errors. I kept asking myself (and oth- ers) why these authors never approached us for fact- checking before publishing.

I have known Michael for a long time as well—many, many years. Both of us frequently visit BSD-related conferences, and we often end up having a beer together, which is always fun. I did read the first edition of *Absolute OpenBSD* when it was published, a long time ago, and quite frankly, I don't remember anything from it. That's a good thing in this case, because I would have remembered if it had been bad. I have recommended it as an introduction to OpenBSD a couple of times.

So when Michael approached me asking whether I would be willing to fact-check the second edition of *Absolute OpenBSD* and provide feedback, I happily agreed.

I have done the reading on airplanes almost exclusively, and one day when I had to fly to Helsinki, I had no chapters left to read. That ended quite badly, with a WWII bomb leading to Frankfurt Airport being closed for a while, the aircraft I was supposed to fly in being identified as defective, and, of course, bad weather causing massive delays. While that was coin- cidence, of course, the rumor was out that I couldn't fly without a chapter from Michael.

Now that I am long done with reviewing, I have survived many flights without chapters to read over, but *Absolute OpenBSD* made long hours up in the sky much more enjoyable for me. Michael has a writing style that I really like—snatchy, funny, and still precise and to the point. Don't skip the footnotes!

In the end, I contributed only a tiny share to this book, but I enjoyed doing so a lot. I hope you enjoy reading it as much.

Henning Brauer OpenBSD PF developer

Acknowledgments

The world has changed in the 10 years since the first edition of *Absolute OpenBSD* came out. I used to have hair, for one thing. In 2003 OpenBSD was somewhere on the edge of open source software, known mainly for an uncompromising, fanatical view of computing security and correctness. So uncompromising that other open source proj ects didn't want to work with it. But a funny thing happened in the follow ing decade: The uncompromising fanatics turned out to be right. More than once I've heard "That's fixed in the latest Linux, and in OpenBSD 3.2." OpenBSD code trickled into other BSDs, Linux, and even some commer cial operating systems. Apple and BlackBerry products include the OpenBSD packet filter. Lots of BSDs support the OpenBSD wireless utilities. And every one runs OpenSSH. So, the first people I have to thank are those who wrote all this code. It's one thing to give a gift to the world, but when

everybody and their pet orangutan has posted their code online, it's another thing when your code is picked up and used dang near everywhere. Well done, guys.

I specifically want to thank Peter Hansteen and Henning Brauer. Henning read the early drafts of this book and pointed out innumerable errors and opportunities for improvement. Peter, the official tech reviewer, had the job of doublechecking all the facts and finding what I'd broken when trying to incorporate Henning's suggestions. While all the OpenBSD folks were friendly and open, these two sank deep into this book and didn't come up for air until it was done. When you see either of them, please buy them a beer. They've earned it.

As always, No Starch Press does a great job producing books. Their indefatigable quest for making everything both correct and pleasing has made this book more than I thought it would be—as usual. Someday I'll consider that excellence routine and, as a result, will be much less impressed when they retain their high standards. But the day their quest for perfec tion bores me has not yet come.

iXsystems provided me with hardware for testing this book. The way to really test an operating system is to push it to its limits. The only way to really find those limits is to exceed them. Preferably as greatly as possible. I used and abused that poor server, folded and spindled and mutilated it, and the blasted thing still ran. (The machine did finally fail, mind you, when I ripped out the hard drives as it was running. That's probably considered cheating, but I had to test the software RAID chapter.) I greatly appreciate iX's support. When iXsystems says their hardware runs BSD, they mean that they've actually used it. In production. For real work. Not just my puny little website and blog.

My blog readers and Twitter followers made researching this book much easier than it could have been. When I throw out a question, some one knows the answer. I try to reward them by throwing out facts, tutori als, observations, and random ranting as well as questions. Check *http:// www.michaelwlucas.com/* for links to these and more.

I considered dropping the haiku from this edition, but overwhelming reader feedback demanded that I not only retain them, but include more and better ones. As an experiment, I solicited haiku on my blog and used some in this book. Ludovic Simpson wrote the haiku for Chapter 7; Justin Sherrill, Chapter 12; and the relentless Josh Grosse, Chapters 1, 3, and 16. As a reward, each of them gets their name on the page you're reading right now. Here's your moment of glory, guys. Enjoy it before it—whoops, it's gone. Sorry.

I wanted this book to come out in 2010. Life happened. It happens. Then life kept happening, apparently with malice aforethought, for four years. My fans waited. The publisher waited. The OpenBSD folks waited. And my longsuffering wife has waited—specifically, for me to quit grum bling that I had to take time away from the crisis du jour and finish this dang book.

Thanks to everyone for your patience and support. There's a certain rightness in having this second edition come out almost exactly ten years to the day after the first edition. But six years would have done nicely, too.

Thanks, everyone.

Introductlon

I asked a psychiatric nurse practitioner about paranoia, and was told that "paranoiais the feeling that people are after you." A medical dictionary would give you a slightly different definition, but this one is actually terribly use- ful for any system administrator. It's not that everyone on the Internet is trying to attack you, but there's always *someone* who wants to break into your system. Even if you think you have nothing of value, someone wants to own your computer. And you won't realize the value of what you have until someone else has it. That's just human nature. If you're not paranoid on the Internet, you're in trouble. That's where OpenBSD comes in. This book is an introduction to the OpenBSD operating system. OpenBSD is a member of the BSD family of operating systems. It is widely regarded as the most secure operating system available anywhere, under any licens- ing terms. It's widely used by Internet service providers, embedded systems

manufacturers, and anyone who needs security and stability. If you're an experienced Unix system administrator who wants to add OpenBSD to your repertoire, this book is for you.

When you finish this book, you should be comfortable working with OpenBSD. You will understand how to configure, troubleshoot, and upgrade computers running OpenBSD and have a basic understanding of OpenBSD's software, security, and network management features.

What Is Security?

We bandy the word *security* around a whole lot, so it's worth taking a moment to talk about security itself. We all have a vague idea of what it means. "Security" means your stuff is safe, and other folks can't get it. That's fine, as far as it goes, but it doesn't go far enough. In information technology, security has three parts:

Confidentiality

This means that secret data should remain secret. Your private infor- mation must not get into the public eye. That Eastern European kiddie porn syndicate should not get your credit card number.

Integrity

This means that data on the system should not be changed without authorization. Your records should remain intact. That intruder should not change the shipping address on an order, making your staff ship a crate of really expensive stuff to an abandoned warehouse in Detroit.

Availability

This means that the system keeps running. If your business depends on your website, losing the website means losing business. Someone who can take your website down can starve your company. And all kinds of people are willing to shut you down, either to compete or just for laughs.

Having been a system administrator for longer than some of you have been alive, I have a less formal idea of security. Security means eliminating bad days caused by computer problems. Spending a day getting a piece of software to compile is not a bad day. Is it an annoying day? Sure, but it's not *bad*. A day when I need to get intruders out of my systems is bad. A day when I have a meeting due to computer intrusions is bad. A day when I realize that I cannot trust any computer on the network, and I must reinstall every blasted piece of gear I own, is really bad.[1]

While OpenBSD cannot change the fact that some of my servers are old enough to leave elementary school, it can fix the software aspects of security.

1. I still have bad days due to people, mind you, but I largely solve them by other means. Don't ask about the mounds of dirt in my backyard.

What Is BSD?

In the 1970s, AT&T needed a lot of specialized, custom-written computer software to run its business. The company was forbidden to compete in the computer industry, so it could not sell this software. Instead, AT&T licensed its software and the related source code to universities for nominal sums. Universities saved money by using this software instead of commercial equivalents with pricey licenses, and university students got access to this nifty technology and could learn how everything worked. In return, AT&T got exposure, some pocket change, and a generation of computer scientists who had cut their teeth on AT&T technology. Everyone got something out of the deal.

The best-known software distributed under this plan was UNIX. Compared with modern operating systems, the original UNIX had a lot of problems. Thousands of students had access to its source code, however, and hundreds of teachers needed interesting projects for their students. If a program behaved oddly, or the operating system itself had a problem, the people who lived with the system had the tools and the motivation to fix it. Their efforts quickly improved UNIX and created many features we now take for granted. Students added the ability to control running pro- cesses, also known as *job control*. The UNIX S51K filesystem made system administrators wail and gnash their teeth, so they replaced it with the Fast File System (FFS), which introduced a whole host of features that have crept into every modern filesystem. Over the years, many small, useful programs were added to UNIX, and entire subsystems were replaced.

The Computer Science Research Group (CSRG) at the University of California, Berkeley, acted as a central clearinghouse for UNIX code improvements from 1979 to 1994. The group collected changes from other universities, evaluated them, packaged them, and distributed the compila- tion for free to anyone with a valid AT&T UNIX license. The CSRG also contracted with the Defense Advanced Research Projects Agency (DARPA) to implement various features in UNIX, such as TCP/IP. The resulting soft- ware collection came to be known as the Berkeley Software Distribution, or BSD. Users took the CSRG's software, improved it further, and fed their improvements back into the CSRG. Today, we consider this a fairly standard way to run an open source project, but in 1979, it was revolutionary.

Fifteen years of work is a lifetime in software development. For compar- ison, Microsoft went from Windows 95 to Windows 7 in 15 years. The CSRG members collected so many enhancements and improvements to UNIX that they replaced almost all of the original UNIX with code created by the CSRG and its contributors. You had to look hard to find any original AT&T code.

Eventually, the CSRG's funding ebbed, and it became clear that the BSD project would end. After some political wrangling within the University of California, in 1992, the BSD code was released to the general public under what became known as the *BSD license*.

The BSD License BSD code is available for anyone to use under what is probably the most permissive license in the history of software development. The license can be summarized as follows:

- Don't claim you wrote this.
- Don't blame us if it breaks.
- Don't use our name to promote your product.

Taken as a whole, this means that you can do almost anything you want with BSD code. (The original BSD license did require that users be notified if a software product included BSD-licensed code, but that requirement was later dropped.) You don't even need to share any changes with the original authors! People could take BSD and include it in proprietary, open source, or free products.

Instead of a restrictive *copyright*, or the more permissive but still restricted *copyleft*, the BSD license is sometimes referred to as *copycenter*, as in "take this down to the copy center and run off a few for yourself." Not surpris- ingly, companies such as Sun Microsystems jumped right on BSD. It was free, it worked, and plenty of new graduates had experience with the tech- nology. One company, BSDi, was formed specifically to take advantage of BSD Unix.

AT&T vs. the World Back in AT&T-land, UNIX development continued. AT&T took parts of the BSD Unix distribution and integrated them with official UNIX, and then relicensed the results back to the universities that provided those improve- ments. This worked well for everyone until the US government broke up AT&T, and the resulting companies were permitted to compete in the com- puter software business.

AT&T had one particularly valuable software property: a high-end operating system that had been extensively debugged by thousands of people and had powerful features, such as a variety of small but mighty commands, a modern filesystem, job control, and TCP/IP. AT&T started a subsidiary, Unix Systems Laboratories (USL), which happily started selling UNIX to enterprises and charging very high fees for it, all the while main- taining the university relationship that had given it such an advanced oper- ating system in the first place.

The University of California, Berkeley's public release of the BSD code met with great displeasure from USL. Almost immediately, USL sued the university and the software companies that had taken advantage of BSD. The University of California claimed that the CSRG had compiled BSD from thousands of third-party contributors unrelated to AT&T, and that it was the CSRG's intellectual property to dispose of as it saw fit. Oddly enough, the lawsuit promoted BSD to thousands of people who never would have heard of it otherwise, spawning open source BSD variants such as 386BSD, FreeBSD, and NetBSD.

In 1994, after two years of legal wrangling, the University of California lawyers proved that the majority of AT&T UNIX was actually taken from BSD, rather than the other way around. To add insult to injury, AT&T had violated

the BSD license by stripping the CSRG copyright from the files it had appropriated
Only about a half-dozen files remained as the source of contention. Bruised and broken in court, USL donated some of those files to BSD while retaining others as proprietary information. BSD 4.4-Lite was released, containing everything except the proprietary files. Due to those missing files, BSD 4.4-Lite was the only formal operating system release ever that was known to not be usable or even compilable as delivered. Everyone knew this, and bought it anyway—a historic feat that modern vendors probably wish they could replicate.
A subsequent update, BSD 4.4-Lite2, is the grandfather of OpenBSD, as well as all other BSD code in use today, such as that in FreeBSD, NetBSD, and Mac OS X.

The Birth of OpenBSD

Theo de Raadt was a NetBSD developer. After many strong, broad, and long-running disagreements with other NetBSD team members on how the project should be run, he went out on his own and founded the OpenBSD Project, attracting like-minded developers. The OpenBSD team quickly established an identity as a security-focused group, and it is now one of the best-known BSD descendants.
The OpenBSD team developers have introduced several ideas into the open source operating system world that are now taken for granted, such as public read-only access to the CVS repository and commit logs. They've also created several pieces of software that have become industry standards across many operating systems, such as sudo and the ubiquitous OpenSSH.
Today, many major companies rely on OpenBSD as a reliable, secure operating system with fanatical attention to security, correctness, usability, and freedom. OpenBSD runs on many different sorts of hardware, includ- ing the standard 32-bit and 64-bit "Intel PC" (i386 and amd64), Apple's PowerPC Macintoshes (macppc), Sparc (sparc and sparc64), and obscure platforms such as the Sharp Zaurus PDA, the Lemote Yeeloong, and antedi- luvian VAXes.
OpenBSD puts almost all of its effort into security features, security debugging, and code correctness, and has demonstrated in the process that correct code has a much lower failure rate, and hence greater security. OpenBSD strives to be the ultimate secure operating system.
The OpenBSD team continually improves the operating system. New features are added only once they meet the team's code and documentation standards. Even if new software is added before it is feature-complete, it is expected to have full documentation and correct code.

The OpenBSD Community

OpenBSD is more than just a collection of bits. It's a community of users, developers, and contributors, with a single central dictator—er, coordina- tor. And this community can be a bit of a shock for anyone who doesn't know what to expect.
How can individuals scattered all over the world create, maintain, and develop an operating system, let alone build a community? Almost all dis- cussion occurs through email and online chat. The process is slower than talking face-to-face, but it's the only cost-effective way for a large group of people in every time zone to communicate in a reasonable fashion. Email and chat also offer written records of discussions. If you want to participate in OpenBSD development, you must be comfortable with email. (There are OpenBSD-dedicated web forums, but they're outside the main community.)
The OpenBSD community has four tiers: users, contributors, commit- ters, and the coordinator.

OpenBSD Users Many open source operating systems put a lot of effort into growing their user base, evangelizing, and bringing new people into the Unix fold. OpenBSD does not.
Most open source Unix-like operating system groups do a lot of pro- Unix advocacy. Again, OpenBSD does not.
The communities surrounding other operating systems actively encour- age new users and try to make newbies feel welcome. OpenBSD specifically and deliberately does not.
The OpenBSD community is not trying to be the most popular operat- ing system—just the best at it what it does.
The developers know exactly who their target market is: themselves. If you can use their work, that's great. If not, go away until you can.

The OpenBSD community generally expects newcomers to be advanced computer users. The members have written extensive OpenBSD documen- tation, and expect newcomers to be willing to read it. They're not interested in coddling new Unix users and, if pressed, will say so—often bluntly and forcefully. They will not hold your hand. They will not develop new features to please users. OpenBSD exists to meet the needs of the developers, and while others are welcome to ride along, the needs of the passengers do not steer the project.

OpenBSD Contributors Contributors are OpenBSD users who have the skills necessary to add features to the operating system, fix problems, write documentation, or accurately report problems. Problems range from typographical errors in the documentation to system crashes. Almost anyone can be a contributor. In fact, the community has even accepted problem reports from me, and resolved them within hours.

Every OpenBSD feature is present because some contributor took the time to write the code for it. Contributors who submit careful, correct fixes, or who provide useful problem reports, are welcome in the OpenBSD com- munity. And if a contributor submits enough fixes of sufficient quality, he might be offered the role of committer.

OpenBSD Committers Committers have write access to the main OpenBSD source code repository. They can make whatever changes they deem necessary for their OpenBSD projects, but are answerable to each other and to the project coordinator. Most committers are skilled programmers who work on OpenBSD during their own time. While being a committer seems glamorous, the role carries a lot of responsibility. If a committer breaks the operating system or changes some- thing so that it conflicts with OpenBSD's driving "vision," he must fix it. Committers try to avoid breaking things, and frequently make their work available on websites and mailing lists before it's integrated into the main OpenBSD source code collection, allowing interested people to preview, test, and double-check their work.

Many committers have very specific coordination roles within OpenBSD. For example, quite a few hardware architectures have a point man for issues that affect that hardware, the compiler has a maintainer, and so on. These committers have earned that position of trust in the community.

OpenBSD Coordinator Theo de Raadt started OpenBSD in 1995 and still coordinates the project. He is the final word on how the system should work, what is included in the system, and who gets direct access to the repository. He resolves all disputes that contributors and committers cannot resolve among themselves. Theo takes whatever actions necessary to keep the OpenBSD Project running smoothly. If something should ever happen to Theo, the project does have plans for replacing him.

Building the OpenBSD organization around a central benevolent dicta- tor avoids a lot of the management problems other large open source proj- ects have.

If you decide to work on OpenBSD, you must accept Theo's decisions as final. A contributor who doesn't accept the project's leader won't remain with the community for long. Theo might have a big stick, but as he is the acknowledged project leader, he doesn't need to use it nearly as often as you might think.

OpenBSD's Strengths

What makes OpenBSD OpenBSD? Why bother with yet another Unix-like operating system when there are so many out there, several closely related to OpenBSD? What makes this operating system worth a computer, let alone worthy of protecting your company's assets?

Portability OpenBSD is designed to run on a wide variety of popular processors and hardware platforms, including Intel-compatible (both 32-bit and 64-bit), Alpha, Macintosh (both PowerPC and Intel systems), and almost anything from Sun. It runs on tiny devices such as the Sharp Zaurus, hefty Hewlett- Packard HP 9000 systems, certain Silicon Graphics workstations, and whatever else grabs the developers' attention. The OpenBSD team wants to support as many interesting hardware architectures as it has the hard- ware and skills to maintain, so more are added regularly, and chances are most computers you encounter can run OpenBSD.

That said, when a hardware platform becomes too obscure, OpenBSD stops supporting it. A few MIPS systems, 68K

Macintosh hardware, and Amiga systems are examples of systems that run older versions of OpenBSD but are not supported by new releases.

Power As a matter of legacy, OpenBSD will run on hardware that has been obso- lete for decades because the hardware was in popular use when OpenBSD started, and the developers try to maintain compatibility and performance when possible. This includes platforms such as the VAX and Alpha, which were considered powerful in the 1980s and 1990s. While someone running OpenBSD on a dual-core 64-bit system might not notice a programming change in OpenBSD that increases the amount of CPU time needed to process network packets, people running OpenBSD on VAX systems will quickly notice that same change.

Of course, some performance-impacting changes cannot be avoided. For example, systems must support IPv6 in the very near future, and I sus- pect that decades-old hardware will struggle to keep up. OpenBSD cannot turn back the clock, but it will leave every scrap of computing power pos- sible for your applications. And after all, that's what's important—people use applications, not operating systems. This focus on performance means that a system running OpenBSD with a 1GB disk and a 486 CPU can still support real applications, such as a DNS or web server.

Documentation Many free software projects are satisfied when they release code. Some think that they go above and beyond by including a help function in the program itself, available by typing some command-line flag. Others really go wild and offer a grammatically incorrect and technically vague manual page.

The OpenBSD community expects the documentation to be both com- plete and accurate. The manual pages for system and library calls are exten- sive, even when compared to other BSDs, and include discussions on usage and security.

Documentation errors are considered serious bugs, and are treated as harshly as any other serious bug. This might sound extreme, but in its own internal audits, the OpenBSD team has found any number of instances where programmers used a library interface exactly as recommended in the manual page, but errors in the manual page made the usage dangerous or insecure. Documentation is important.

Free In the spirit of the original BSD license, OpenBSD is free for use in any way, by anyone, for any purpose. You can use it with any tool you like, on any computer.

Most of today's free software is licensed under terms that require software distributors to return any changes to the project's owner, but OpenBSD doesn't even carry that requirement. You can use OpenBSD in your pro- prietary system, ship that system everywhere in the world, and not pay the developers a dime.

OpenBSD is perhaps the freest of the free operating systems. Like every other free Unix-like operating system, the source code inherited from BSD originally contained a wide variety of programs that shipped under condi- tional licenses. Some were free for noncommercial use. Some were free if you changed the name once you changed the code. Others had a variety of obscure licensing terms, such as indemnifying a third party against law- suits. These programs have either been relicensed (with the permission of the original author) or ripped out and replaced with free alternatives.

The word *freedom* has been given a lot of different twists by people in the programming community. Some believe that software is free if you can download it and use it. Some believe that software is only free if the end user gets the source code. The OpenBSD idea of freedom is that its code can be used for any purpose, by anyone.

Consider this: During a discussion on an OpenBSD mailing list regard- ing licensing terms,[2] Theo de Raadt said:

We know what a free license should say. It should say

Copyright foo I give up my rights and permit others to:

distribute sell give modify use I retain the right to be known as the author/owner When it says something else, ask this:

- is it 100% guaranteed fluff which cannot ever affect anyone? - is it giving away even more rights (the author right)? If not, then it must be giving someone more rights, or by the same token—taking more rights away from someone else! Then it is _less_ free than our requirements state!

2. This is from October 24, 2002, on the openbsd-misc mailing list. It's more than a decade old, but still pretty much says it all.

The OpenBSD team works hard to ensure that every line of code it sup- ports is licensed in this manner.

The source code tree does include code under different licenses, such as the GNU C compiler gcc*, binutils, and so on. OpenBSD* runs *fine without them—you just can't* compile *OpenBSD without them.*

This is pretty straightforward. OpenBSD is a gift. You're free to use it or not. As with any gift, you can do whatever you want with it. But you're not free to bug the developers for features or support.

Correctness Every skilled programmer knows that programs written correctly are more reliable, predictable, and secure. However, many free software producers are satisfied if their code compiles and simply seems to work, and quite a few commercial software companies don't give their programmers time to write their code correctly. OpenBSD developers strive to implement solutions correctly. They make it a strict rule to write programs in a reliable and secure manner, following best current programming practices. And exposing the code to "weird" environments such as ancient VAXes is part of the discipline; OpenBSD developers insist that some subtle bugs (and a few less subtle ones) have been pinpointed only during testing on one of OpenBSD's less mainstream architectures. Fixing those bugs benefits all users, of course.

OpenBSD implementations follow UNIX standards, such as the Portable Operating System Interface (POSIX) and the American National Standards Institute (ANSI), but they are less concerned about extensions to these stan- dards created by third parties. For example, many Linux extensions do not appear in OpenBSD. When those extensions are added to standards, the OpenBSD team will add them.

OpenBSD code has been repeatedly audited for correctness through a lot of hard work. Anyone who tries to introduce incorrect code will be turned away—generally politely, and often with constructive criticism, but turned away nonetheless. And that brings us to OpenBSD's most well- known claim to fame.

Security OpenBSD strives to be the most secure operating system in the world. While it can reasonably make that claim today, maintaining that position requires constant effort. Intruders constantly try new ways to penetrate computers, which means that today's feature might be tomorrow's security problem. As OpenBSD developers learn of new classes of programming errors and security holes, they scan the entire source tree for that type of problem and make fixes before anyone even knows how these issues *might* be exploited.

Additionally, OpenBSD takes advantage of any security features offered by hardware. For example, AMD's 64-bit Intel-compatible CPUs can mark a page of memory as either executable or writable, but not both. (Intel later copied this feature.) This alleviates many buffer overflow attacks, but the operating system must use this facility. OpenBSD supported this feature in 2003, shortly after the hardware was released. In fact, OpenBSD gener- ally supports all hardware security features offered on a platform.

The history of computing shows that users cannot be expected to patch or maintain their own systems. Systems must be secure against existing and future attacks out of the box. OpenBSD's goal is to eliminate problems before they exist.

OpenBSD and Your Security

Even though OpenBSD is tightly secured, intruders still break into OpenBSD systems. This might seem contradictory, but in truth, it means that the person running the computer didn't understand computer security. OpenBSD has many integrated security features, but you cannot assume that these features secure everything running on the system. That's just not possible. No operating system can defend itself against operator error. An operating system can protect itself from software problems to a limited extent, but ultimately, the responsibility for security is the administrator's.

Consider a web server—even OpenBSD's integrated Apache server— running on OpenBSD. OpenBSD provides the web server with a stable, reli- able platform, and will provide services as the web server requests, within the limits assigned by the system administrator. If the system administrator has configured the web server correctly, a web server failure will not endan- ger the operating system. If the system administrator configures the web server to run with unlimited privileges, the web server can inflict almost unrestricted damage on the underlying system.

Or consider a less extreme case. The web server might be configured correctly, but suppose you install insecure forum software. An intruder can break into the forum and edit its data—maybe grab the username and pass- word

the forum software uses to access the local database. If that account information matches a system-level username and password, the intruder might be able to leverage them to gain access to the system. Or perhaps he can use that username and password to get administrator-level access to the database and penetrate other applications. What if those applications have elevated privileges?

Only careful, consistent, thoughtful work by a system administrator can prevent intrusions. Throughout this book, we'll discuss some basic security precautions you should take when installing and running software. We'll also discuss the advanced security features OpenBSD offers in order to pro- tect itself.

OpenBSD's Uses

Where does OpenBSD fit into your computing strategy? That ultimately depends on your strategy and your needs. OpenBSD can be used anywhere you need a solid, reliable, and secure system. I recommend OpenBSD for any of three different roles: a desktop, a server, or network management.

Desktop If you need a powerful desktop system with all the features you would expect from a complete Unix-like workstation, OpenBSD will do nicely. Graphic interfaces, office suites, web browsers, and other desktop software are available in the ports collection, OpenBSD also supports a variety of development tools, application environments, network servers, and other features that programmers and web developers need. If you're a network administrator, you'll find that OpenBSD supports packet sniffers, traffic analyzers, and all the other programs you rely on.

Server If you're serving web pages, handling email, providing Lightweight Directory Access Protocol (LDAP) or database services, or offering any other sort of network service to clients, OpenBSD can help you. It's a cheap and reliable platform. Once it's set up, it just works. And, of course, it's secure, which you cannot underestimate on the Internet.

Network Management OpenBSD makes an excellent firewall, bridge, or traffic shaper. You can use it to support intrusion detection software, web proxies, and traffic monitors. The integrated packet-filtering firewall and supporting software provides state-of-the-art network connection management and control, and can strip out many dangerous types of traffic before it reaches your servers. And its load-balancer features are competitive, with many commercial offerings that cost thousands of dollars more.

About This Book

This book is written for experienced Unix users or system administrators who want to add OpenBSD to their repertoire. I assume you're familiar with basic commands, such as tail(1), chmod(1), ping(8), and so on, and that you know why each command in this list includes a number in parentheses after the name. We'll discuss many programs that you might already be familiar with, but that might be slightly different in OpenBSD.

For maximum benefit, you should install OpenBSD on a dedicated machine. OpenBSD can coexist with other operating systems or run in a virtual machine, but if you're going to use OpenBSD in a production envi- ronment, you should run it on its own.

Many people believe that OpenBSD is not the easiest Unix-like operat- ing system, or the easiest version of BSD, or even the easiest open source BSD. OpenBSD doesn't have handy wizards that walk you through each stage of the configuration process, although it does has a few menu-driven front ends. Once you're familiar with how the system works, though, such wizards would only get in the way.

To truly understand OpenBSD, you must be willing to learn, experiment, and spend time accumulating understanding. Much of this knowledge can be directly applied to other versions of BSD, other Unix-like operating systems, and even completely foreign operating systems, such as Microsoft's Windows.

Contents Overview

While this book is designed to be read from front to back, here's a brief description of each chapter, in case you would rather skip around randomly.

Chapter 1: Getting Additional Support Discusses the OpenBSD docu- mentation available both in the installed system and on the Web. You need to understand what you're getting into before installing OpenBSD. **Chapter 2: Installation Preparations** Discusses installation on a standard amd64 (also known as the 64-bit Intel-compatible) system. Making some decisions before you install OpenBSD will ensure that you don't need to reinstall it later. **Chapter 3: Installation Walk-Through** Carries you through every step of a real OpenBSD installation. The OpenBSD installer assumes a cer- tain level of knowledge about computer hardware and OpenBSD that you might not yet possess. This walk-through will guide you through the rough spots. **Chapter 4: Post-Install Setup** Discusses the basic steps you should take after installing OpenBSD to make your system secure, stable, and usable. **Chapter 5: The Boot Process** Covers system startup. Different situa- tions require different startup methods, and we'll cover them all. We'll also discuss how OpenBSD starts its component software. **Chapter 6: User Management** Discusses how to add, remove, and restrict OpenBSD user accounts.

Chapter 7: Root, and How to Avoid It Discusses controlling user privileges and permissions. OpenBSD includes powerful tools such as classes and limits, as well as the privilege management tool sudo(8). **Chapter 8: Disks and Filesystems** Covers disk management with the standard OpenBSD filesystems. **Chapter 9: More Filesystems** Covers advanced filesystem topics such as the Network File System (NFS), working with disk images, software RAID, and encrypted disks. **Chapter 10: Securing Your System** Considers how to maintain security using tools such as file flags, securelevels, OpenBSD security announce- ments, and some basic cryptographic tools. **Chapter 11: Overview of TCP/IP** Reviews the basics of TCP/IP ver- sions 4 and 6, and covers some of OpenBSD's tools for examining and troubleshooting the network. **Chapter 12: Connecting to the Network** Takes you through configur- ing OpenBSD's network stack for Ethernet, trunks, and virtual local area networks (VLANs). **Chapter 13: Software Management** Describes OpenBSD's add-on software tools. You'll learn how to install precompiled software, com- pile your own software, and verify and remove software. **Chapter 14: Everything /etc** Describes each major file in */etc* that isn't covered elsewhere, and discusses how you might want to use those files. **Chapter 15: System Maintenance** Covers the various ways OpenBSD maintains itself and how you can make those processes fit your environ- ment and workflow. **Chapter 16: Network Servers** Covers configuring software integrated with OpenBSD. You'll learn about the system logger and log file man- agement, the DHCP server, the web server, and more. **Chapter 17: Desktop OpenBSD** Covers software useful to OpenBSD as a desktop, such as the window manager cwm(1) and Xenocara. This chapter includes coverage of important software that makes using OpenBSD with a desktop easier, such as SSH keys and tmux. **Chapter 18: Kernel Configuration** Discusses the various tools avail- able to configure a standard kernel. Unlike many other free Unix-like operating systems, OpenBSD does not expect or require the system administrator to compile a kernel. You can tune the standard kernels without recompiling. **Chapter 19: Building Custom Kernels** Discusses how to recompile a kernel in those rare instances when you must. **Chapter 20: Upgrading** Covers how to upgrade OpenBSD, either from a snapshot or from source.

Chapter 21: Packet Filtering Documents OpenBSD's integrated packet-filtering engine, PF. It includes discussions of real-world situa- tions and how to handle them. **Chapter 22: Advanced PF** Introduces things that the packet filter can do beyond just filtering packets. **Chapter 23: Customizing OpenBSD** Includes tidbits that didn't fit anywhere else but are not large enough topics to merit their own chap- ters. This includes diskless OpenBSD, building bootable USB installation media, and making custom OpenBSD installation sets.

This book won't cover everything OpenBSD can do, but it will get your feet firmly under the table. To learn the rest, you'll need to access OpenBSD's information resources, which is the subject of the first chapter.

1

GettinG AdditionAl Help

Mailing lists are rough; homework is mandatory. Love it or leave it.

You've bought this book, so you now pos- sess all the information you will ever need about OpenBSD. You hold in your hands the ultimate repository of all OpenBSD wis- dom and acumen, and once you complete it, you will be lord and master of all that OpenBSD offers. Right?

Sorry, no. No one book can possibly contain everything there is to know about OpenBSD. UNIX is pushing 40 years old, and BSD operating systems have been around for more than 30 years. OpenBSD itself is over 15 years old, and is built on decades of tradition, knowledge, and community devel- opment. You won't master it with any single book. You might master it with a room full of books and a few years of study, if you stop wasting time on trivialities like having a family and avoiding scurvy.

The OpenBSD community maintains a wide variety of information sources. Some, such as the manual, are integrated with the OpenBSD operating system. The OpenBSD team maintains additional resources, such as the main OpenBSD website and the official OpenBSD mailing lists. Users and devotees maintain additional websites, mailing lists, and

documentation. The flood of information can overwhelm experienced users and intimidate new users so badly that they don't even try to sort through it. That's why this chapter will take your hand and lead you through some of the other resources available.[1]

OpenBSD's Support Model

If you've worked with only commercial UNIX, you might find OpenBSD's support structure a little surprising. There is no toll-free number to call and no vendor to guide you. No, you may not speak to the manager of the sup- port team. There isn't one. The management is you.

Many commercial operating systems conceal their inner workings, and the only access you get is through the programs, application programming interfaces (APIs), and application binary interfaces (ABIs) they provide. If you want to learn more about how your operating system works, you can't (unless you reverse-engineer it). When something breaks, you either live with it or pay the vendor to solve the problem.

OpenBSD, on the other hand, is completely open. You can view the source code, the compiler, and the resulting binaries. You have the official manual and a whole bunch of ancillary documentation. You have access to the developers' logs—logs that describe every change ever made to every part of the system—through the same tools the developers use. You can back out of changes, understand the motivation behind the changes, and even contact the people who have most recently worked on a component you're interested in and ask them what they were thinking. You can add your own features. In other words, you have the opportunity to understand OpenBSD in

exquisite, excruciating detail.

If you want to learn about OpenBSD, you must jump from eating what you're served to reading the cookbook and creating your own meals. If you're willing to learn using the information provided, you will develop skills, and you'll probably even make some friends in the OpenBSD community along the way. If you want to use OpenBSD and don't have the time or inclination to learn, invest in a commercial support contract. Many companies and consultants around the world support OpenBSD. The OpenBSD website lists dozens.

If you don't want to learn and don't want to buy a support contract, then OpenBSD is simply not for you.

The Code Is Fine. What's Wrong with You?

Systems administrators rarely have trouble with OpenBSD itself; the soft- ware runs, and it runs well. Most problems arise from their own under- standing, or lack thereof.

When a program behaves unexpectedly, the problem is usually a gap in your expectations or understanding, and the OpenBSD community expects

1. Yes, the first chapter in this book is about getting help outside the book. I am aware of the irony; you don't need to tell me.

that you will work to improve your own knowledge so that you can make the system meet your needs. Other people make OpenBSD work correctly, and you can, too.

That said, you may still find that a problem is quite real, but you can't be certain that it was caused by OpenBSD itself until you understand correct behavior—not just how you think the system works, but how it really does work. The problem could be an OpenBSD bug, bad hardware, or an errant third-party tool. To correctly identify bugs, you must learn how the system should behave and why.

For example, before writing the first edition of this book. I had never used an OpenBSD machine to display a serial console. All of my Unix-like boxes had connections to a rusty old terminal server. Most people don't have that many serial consoles, and they want to use a null modem cable between two OpenBSD machines and have each serve as the terminal for the others' console. (We'll cover serial consoles in Chapter 5.) From read- ing the manual page (discussed in the next section), this common configu- ration seemed simple enough: Attach the cables, configure one machine to dump its console to the serial port, become root on the display machine, and enter **tip tty00**. The other machine's console should have appeared in the terminal window, but that didn't happen.

The next question is, "What's wrong?" It might have been an OpenBSD bug, a hardware failure, or a gap in my comprehension. Swapping systems around demonstrated that the command worked on other OpenBSD machines, just not my particular test box. Further tests with a serial mouse and modem showed that the serial port on the test machine was bad.

Had the serial port been in working order, I might have actually found an OpenBSD bug, but probably not.

Sources of Information

OpenBSD provides information through three primary channels: manual (man) pages, websites, and mailing lists. To understand why your system behaves in a particular way in your environment, you might need to check all three.

Man Pages Man pages are the original format for presenting documentation on Unix- like systems. While man pages have a reputation for being obtuse, difficult to read, or incomplete, the OpenBSD team expects its man pages to be readable, correct, and complete.

When man pages were first created, the average system administrator was a C programmer. As a result, man pages were written by programmers for programmers. The OpenBSD developers are programmers and con- sider man pages the final word in OpenBSD documentation. Even docu- mentation errors are considered serious bugs and are dealt with as quickly as possible. Man pages should be your first line of attack in learning how OpenBSD works.

That said, a man page is *not* a tutorial. The manual explains how things work, not what to type to achieve particular effects. You must be able to assemble the knowledge offered by the man page into the tool that you need. If you want tutorials, read articles on third-party websites, the FAQ, and this book. If you find a tutorial that tells you how to do exactly what you want, read the relevant man pages along with the tutorial. Just remember that anyone can

write a tutorial, and there's no guarantee of any particular tutorial's effectiveness or security.

Manual Sections The OpenBSD manual has nine sections, and each man page appears in only one section. These sections are sometimes called *volumes*, a name from the days when the manual was small enough to print and distribute. Each section covers a single topic. The sections are as follows:

1: General commands 2: System calls and error numbers 3: C libraries 3p: Perl libraries 4: Device drivers 5: File formats 6: Games 7: Miscellaneous 8: System maintenance and management commands 9: Kernel internals

Man pages often appear with the section number in parentheses after the command, such as ping(8) or ed(1). This gives you the name of the com- mand (ping) and the section where the command is documented (8, on system maintenance). Almost every part of OpenBSD has a man page.

Viewing Man Pages View man pages with man(1). If you know the section number, enter it before the program name, but the section number isn't mandatory. For example, to see the man page for the standard network utility ping(8), enter this:

```
$ man ping
```

You'll get something like this in response.

```
PING(8) OpenBSD System Manager's Manual PING(8)
NAME
ping - send ICMP ECHO_REQUEST packets to network hosts
SYNOPSIS
ping [-DdEefLnqRrv] [-c count] [-I ifaddr] [-i wait] [-l preload]
[-p pattern] [-s packetsize] [-T tos] [-t ttl] [-V rtable] [-w maxwait] host
DESCRIPTION
ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_REPLY from a host or gateway.
ECHO_REQUEST datagrams (``pings'') have an IP and ICMP header, followed by a "struct timeval" and then an arbitrary number of
"pad" bytes used to fill out the packet. The options are as follows:
-c count
Stop sending after count ECHO_REQUEST packets have been sent. If count is 0, send an unlimited number of packets.
-D Set the Don't Fragment bit. ...
```

You can learn more than you ever wanted to know about the lowly trouble- shooting tool ping just by reading this document. If you need more infor- mation, look at the other man pages referenced by ping(8). Read enough pages, and you'll develop an in-depth understanding of OpenBSD.

Once you're in a man page, pressing the spacebar or PGDN takes you forward one full screen. If you don't want to go that far, press ENTER or the down arrow to scroll down one line. Typing **B** or pressing PGUP takes you back one screen. To search within a man page, type **/** followed by the word you're searching for, and then press ENTER. You'll jump to the first appear- ance of the search term. Subsequently, typing **N** takes you to the next occur- rence of the word.

The man page navigation discussed here assumes that you're using the default BSD pager, more(1). If you're using a different pager, use that pager's syntax. If you know enough about Unix-like systems that you've already set your preferred default pager, you can probably skip this part of the book.

Finding Man Pages New users often say that they would be happy to read the man pages, if they could find the right ones. You can perform basic keyword searches on the manual with apropos(1) and whatis(1). The command apropos searches for any man page name or description that includes the word you specify. The command whatis does the same search, but matches only whole words. For example, if you're interested in the vi text editor, you might try the following:

```
$ apropos vi ... vmware (4) - VMware SVGA video driver
```



```
voodoo (4) - Voodoo video driver wsudl (4) - video driver for DisplayLink USB display devices xcmsdb (1) - Device Color
Characterization utility for X Color Management System ...
```

On my system, this generates 686 entries, most of which have nothing to do with vi(1). The random selection of entries shown here includes device drivers and user utilities, but no text editors. If you examine them closely, you'll

see that the letters *vi* appear in each of them, encapsulated within words like ***vi****deo* or *de****vi****ce*. Depending on what you're looking for, apropos can offer you far too much information. Try a similar search with whatis:

```
$ whatis vi
ex, vi, view (1) - text editors
```

Matching only whole words can be more useful. Experiment with apropos and whatis until you're comfortable with them, and you should be able to find just about any topic you like.

Overlapping Man Page Names Some man pages have a name in common with a man page in another sec- tion. For example, suppose someone mentions sysctl and you want to learn about it, so you search the man pages.

```
$ whatis sysctl
sysctl (3) - get or set system information
sysctl (8) - get or set kernel state
sysctl.conf (5) - sysctl variables to set at system startup
```

We have a *sysctl.conf* file and two different man pages called sysctl. Manual section 3 is for C libraries. If you're just learning about sysctl, you might find this man page intimidating.

By default, man displays the first matching page it finds. It searches commands first, then games, then programming libraries, then add-on programs such as Perl. You can change this search order in */etc/man.conf* (see Chapter 14). In this situation, manual section 3 is before manual section 8. Without specifying a section number, you'll read about the programming interface. To see the man page for the system administrator command sysctl, you must run man 8 sysctl.

Man Page Contents Manual authors try to arrange their content meaningfully, although mean- ingful varies depending on what it documents. A man page for a common user command will probably be much easier to understand than a man page for a kernel-programming interface. Even so, most man pages are divided into sections. Some of the common sections include the following:

- NAME tells you the names of a program or utility. Some programs have multiple names; for example, the vi(1) text editor is also available as view(1) or ex(1). The man page lists all of these names.
- SYNOPSIS lists the possible command-line options and their arguments, or gives examples for how programmers can call a library or interface. Once you've read the man page and used the command a few times, the synopsis might be enough to remind you of what you need.
- DESCRIPTION contains a brief synopsis of the program or feature. You'll also find detailed discussions of the command-line options.
- BUGS describes known failure conditions and weird behavior, and gen- erally discusses when a feature doesn't work as you might expect. Always look at the BUGS section; it can save you a lot of time. I've frequently had a problem with a command only to find that the behavior, and sometimes a work-around, is listed here. Honesty is a wonderful thing in computing products.
- SEE ALSO is traditionally the last section in a man page. OpenBSD is an interrelated whole, and every command has ties to other commands. In an ideal world, you would read every man page and be able to hold an integrated image of the system in your head. Because most of us can't do this, this section directs you to related man pages.

Man Pages on the Web The man pages are also available on *http://www.OpenBSD.org/* and its vari- ous online mirrors. One of the interesting things about the web-based man pages is that you can look at them for both previous releases and for other architectures. Do you want to know if there's a difference between the sysctl command for i386 and Loongson hardware? The web versions will let you compare two different man pages. (You can also do this with the integrated manual, but the web version makes it easier.)

The OpenBSD Website The OpenBSD website (*http://www.OpenBSD.org/*) contains a lot of information—from administration, installation, and management to where to find hardware. The front page links you to a general discussion about OpenBSD's goals and support, where you can get OpenBSD, available resources, and ways you can support OpenBSD. Project members keep the website updated. If you have an OpenBSD problem or question, check this website first.

Mirrors Many people across the world mirror the OpenBSD website. The main web- site is quite heavily accessed, and mirrors will often respond more quickly. You'll see links at the bottom of the main website. I recommend you

pick and bookmark an official mirror that responds quickly for you. The mirror sites are generally underused and hence faster than the official site.

The OpenBSD FAQ The OpenBSD FAQ is OpenBSD's repository of answers to frequently asked questions. While much of the information in the FAQ duplicates the man pages, the FAQ presents this information in a question-and-answer format that's often easier to understand.

Unlike many other FAQs, the OpenBSD FAQ includes extensive tutori- als. For example, Chapter 4 of the FAQ contains the full, detailed installa- tion process. If you're having a problem, or want to know how some major part of OpenBSD works, check the FAQ first!

Non-Project Websites Many people maintain websites dedicated to OpenBSD content, related to OpenBSD, or generally useful to OpenBSD users. Any time you have a problem or are trying to understand something, your search engine might lead you to articles on these sites. Read third-party documentation care- fully and skeptically, however. Tutorials and articles outside the OpenBSD Project might contain erroneous information, violate OpenBSD's best prac- tices, or work only in the author's particular environment.[2]

The only third-party website I can unconditionally recommend is *http:// www.undeadly.org/*, an OpenBSD news aggregator. When a website posts worthwhile OpenBSD-related content, the *undeadly.org* maintainers link to it.

If you want a web forum to discuss OpenBSD, you might try DaemonForums (*http://www.daemonforums.org/*). DaemonForums has discussion groups for all of the major BSD variants, including OpenBSD.

OpenBSD Mailing Lists The OpenBSD Project primarily communicates through mailing lists. All mailing lists are accessible to the public, but some welcome new users more than others. Many hardware platforms have dedicated mailing lists, but they welcome only platform-specific discussions and specifically reject prob- lem reports. The OpenBSD website contains a complete list of mailing lists. Here, I'll cover only the mailing lists useful to average users.

announce@OpenBSD.org

This low-volume, moderated list includes only important OpenBSD news. This list receives at least one message every six months, when a new version of OpenBSD comes out.

2. Of course, this doesn't apply to anything on my blog. Everything I post is the one word of truth.

security-announce@OpenBSD.org

When the OpenBSD team learns of an OpenBSD security flaw, it posts a bulletin to this list. If you are running an OpenBSD machine on the Internet, you *must* subscribe to this list. I'll say more about *security-announce* in Chapter 10.

misc@OpenBSD.org

This list contains general OpenBSD discussions. While this is the "miscellaneous" list, it still has strict rules, and the community firmly enforces its etiquette. I'll cover how to usefully post to an OpenBSD mailing list in "Using Mailing Lists" on page 11.

tech@OpenBSD.org

This list is for in-depth technical discussions, such as code reviews and protocol analysis. If you want to know what the OpenBSD folks are working on, read this list. It's not for support requests. As a good rule of thumb, if your email doesn't include a code diff, don't send it to this list.

advocacy@OpenBSD.org

This list is for promoting OpenBSD. If you want to talk about OpenBSD's inherent awesomeness in a nontechnical manner, use this list.

You'll find other lists that might interest you, such as *www@* (discussions about the website) and *ports@* (discussing the OpenBSD ports system, which we'll cover in Chapter 13), but those lists require more OpenBSD expertise than most beginners have.

The easiest way to access the mailing lists is the web interface at *http:// lists.OpenBSD.org/*. The OpenBSD team manages its mailing lists with Majordomo (*http://www.greatcircle.com/majordomo/*). If you're familiar with that package, you can access the mailing lists at *majordomo@OpenBSD.org*.

Unofficial Mailing Lists You can find a fairly complete list of all OpenBSD-related mailing lists hosted by third parties at *http://www.OpenBSD.org/mail.html*. This includes lists in languages other than English.

One unofficial list, run by an OpenBSD developer, is the PF mailing list, for users of the OpenBSD packet filter. This list is for all PF packet filter users, not only OpenBSD, but OpenBSD users dominate the list. If you want to learn more about PF, subscribe to this list. You can find more at *http://www.benzedrine.cx/*.

Read-Only Mailing Lists So *misc@OpenBSD.org* looks like the mailing list for you, and you subscribe. If you race ahead and ask all your questions, you'll immediately accomplish a couple things: You'll alienate the community, and you'll be told to shut up and go away; you certainly won't make friends. That's mainly because the mailing lists exist to be read more than posted to.

Unless you're in a truly unique situation or really on the bleeding edge of OpenBSD development, more likely than not, someone has struggled with your problem previously. That person probably got an answer, and that answer probably hasn't changed. The quickest and least intrusive way to answer your question is to find that previous message. That's where the mailing list archives come in.

Your favorite search engine has already indexed the OpenBSD mailing list. Always ask the search engine your question before going anywhere near the mailing lists. If you've looked around and found that your question is truly unique, send a message to the mailing list. But when you're first start- ing out, you're better off treating the OpenBSD mailing lists as read-only.

Using OpenBSD Problem-Solving Resources

Let's pick a common question and use the OpenBSD resources to solve it, without resorting to sending mail. One of the things OpenBSD is known for is its support for cryptography in hardware. How does that work, and what does OpenBSD do to support it? Here's how I would search for infor- mation on this topic from each information source the OpenBSD Project provides.

Using the OpenBSD Website Look at *http://www.OpenBSD.org/* and you'll see a link to Crypto. This takes you to the Cryptography page, which covers OpenBSD's cryptography sup- port. It includes algorithms and discusses how the team has integrated OpenSSL into hardware cryptographic accelerators. Read, learn, and enjoy.

Using Man Pages Let's try running apropos cryptography:

```
$ apropos cryptography RSA_public_encrypt, RSA_private_decrypt (3) - RSA public key cryptography
```

This man page isn't terribly useful as a general overview, and whatis cryptography doesn't return anything. Cryptography is often referred to as *crypto*. apropos crypto gives too many results. whatis crypto gives more reasonable results:

```
$ whatis crypto crypto (3) - OpenSSL cryptographic library crypto (4) - hardware crypto access driver crypto (9) - API for cryptographic services in the kernel
```

This is a fairly short list, and all the entries look promising. Manual section 3 is for programmer interfaces, section 4 is for device drivers,

and section 9 is for the kernel. If you're specifically looking for hardware cryptographic accelerators, section 4 should jump out at you, but start wher- ever you feel most comfortable.

Using Internet Searches Go to your favorite search engine and search for "OpenBSD crypto hardware support." On the day I wrote this, the first result led me to the official page on the OpenBSD website. The second hit was a paper on the OpenBSD cryptographic framework. You'll find old articles, archived mailing list dis- cussions, man pages, tutorials, and innumerable blog posts. You'll probably need to add the model number of a particular cryptographic accelerator card to reduce the results to a manageable number.

Using Mailing Lists If the mailing list archives, a web search, the OpenBSD FAQ, the OpenBSD website, the integrated manual, and other assorted resources do not answer your question, you can ask for help. A variety of highly knowledgeable and very skilled computing professionals read the OpenBSD mailing lists. Many of these people enjoy working with OpenBSD and want to help intelligent new users. In their minds, "intelligent" equates to

“not asking a question that has been asked before.”
Have another look at all the ways we gathered information on OpenBSD’s cryptographic hardware accelerator support in the previous section. Information about most other topics is just as readily available. People who take the time to read and answer questions on the OpenBSD mailing lists have already spent considerable time and energy creating this content and ensuring its accuracy. Now imagine their reaction when some- one asks about cryptographic accelerator support on the public mailing list. Most OpenBSD experts will assume any of the following:

- The person wants their hand held.
- The person is unwilling to read the documentation.
- The person has nothing but contempt for the OpenBSD developers.
- The person has the intelligence of a brick.

Most OpenBSD experts would conclude that the person asking the question simply isn’t ready to run OpenBSD. At best, the questioner will be ignored. At worst, some experienced OpenBSD person who wrote all this documentation will take offense at his hard work being so thoroughly dis- counted and flame the questioner badly enough that his monitor will need three months in the Mayo Clinic burn unit.
Keep this in mind before you send an email. Have you really checked everywhere for an answer? Are there any other search terms you haven’t tried? Performing a few extra searches with different keywords is much faster than composing a useful email, and there’s an excellent chance you’ll find the answer to your question.

If you’re familiar with other free Unix-like operating systems, OpenBSD’s mailing lists might give you a bit of a culture shock. OpenBSD users are advanced computer users almost by definition. If an experienced systems administrator tries to debug a piece of software, that administrator is expected to know enough to ask the responsible party. If you go to a Linux forum, you’ll find people discussing server and client programs, desktop environments, and dang near any other piece of software that runs on that platform. Those forums are manned by volunteers dedicated to providing around-the-clock support and extreme efforts to help their operating sys- tem conquer the world.
The OpenBSD folks don’t care if they take over the world or not. They don’t really care if you use their software. If other people can get use out of it, that’s great. If not, oh well. They will happily assist you with OpenBSD- specific problems, but they don’t really care about your database issues or your website. If you’re having trouble porting your preferred window manager to OpenBSD because of some subtle bug in OpenBSD’s libc, the OpenBSD people would love to talk to you. If you can’t configure your win- dow manager the way you like, then you should talk to the window manager support group instead.

Creating a Good Help Request Before you send an email, think carefully about the problem you’re trying to solve. What question should you actually be asking? Define the issue as narrowly as possible. Suppose you cannot connect to a virtual private net- work (VPN) server with OpenBSD’s IPsec client. Is the problem that you can’t actually reach the IPsec server? Does the connection work when you turn off your OpenBSD firewall, but return when you re-enable filtering? Does isakmpd(8) crash and leave a core file every time you try to start the VPN? Each of these is a very different problem. Including the precise prob- lem in your email will get you a better reception.
The first paragraph of your email should state your problem briefly and succinctly. If your first paragraph doesn’t contain enough to interest people, they’ll probably delete the email before getting to anything relevant.
After that important first paragraph, gather any and all information related to the problem. Include this information in your email. This should include the following:

- The version of OpenBSD you are running
- Your hardware platform
- Any error output (be sure to check */var/log/messages* as well as your terminal)
- The contents of */var/run/dmesg.boot*
- A complete but narrow problem description

Give your email message an appropriate subject. A subject like “Problem with OpenBSD” will get ignored. A subject like “Reproducible isakmpd
crash on newest OpenBSD snapshot” will immediately attract attention. Many OpenBSD people decide which messages to read based entirely on the subject line. Moderately advanced email-reading programs allow the recipient to delete an entire thread of discussion based on the subject line or message headers.

How to Be Ignored Many senior OpenBSD users use a text-based email reader such as Mutt (although quite a few do use more graphic email readers, mind you.) Text- based email programs are very powerful programs for handling thousands of emails a day, but they show only text, and they do not display HTML mes- sages well. If you are using a graphic mail client such as Mozilla Thunderbird or Microsoft Outlook, wrap your text at 72 columns. Sending mail in pure HTML or without readable line wrapping invites experienced recipients to discard it unread.

This might seem harsh, and it’s definitely different from mailing lists run by other open source operating systems. But most email clients are not suited to handle thousands of messages in a day, scattered across dozens of mailing lists, with several parallel discussion threads each, in a manner accessible to the human mind. I receive thousands of email messages a day, and I know many OpenBSD developers get—and process—even more. We simply cannot cope without mail tools that address our problems. HTML support is not nearly as necessary as the ability to manage, present, and pro- cess a large number of messages in a sensible manner.

On a similar note, most email attachments are unnecessary (and several of the OpenBSD mailing lists will unceremoniously strip them from incom- ing messages). You do not need to PGP sign your email, and those business card attachments just demonstrate that you really shouldn’t be running OpenBSD. If you include a signature in your email, it should be no longer than four lines. Long ASCII art signatures, even really nifty ones featuring the OpenBSD blowfish, are right out.

It’s easy to let frustration turn a simple request into a rabid demand for immediate assistance. Remember to be polite; the people who receive your message might decide to help you, but they’re under no obligation to do so. If you want someone to be obliged to help you, buy a support contract.

Sending Your Email Before sending your email, double-check your search engine. Are you *sure* this hasn’t been asked before?

Send all of your information and your narrow, specific, documented question about the OpenBSD core system to *misc@OpenBSD.org*. Yes, OpenBSD has many other mailing lists, and some of them might look more appropri- ate for your problem, but people who post to them are almost always told to go ask on *misc@* instead. People on *misc@* might refer you to another mailing list, but it’s much better to post a message to a specific list if that message starts with “So-and-so on *misc@* recommended that I ask this here.”

If you have a narrow, specific, well-documented question about a piece of add-on software (or package, as discussed in Chapter 13), you can send it to *ports@OpenBSD.org* instead.

Responding to Email The response you receive might be a brief note with a URL, or even just the words “man such-and-such.” If that’s what you get, that’s where you need to go. Remember that you’re asking because you don’t understand something, and these responses tell you where to learn the answer to your question. Don’t just email back asking someone to hold your hand.

If you don’t understand the reference you’re given, treat that as another problem. Narrow down the source of your confusion. Man pages and tutori- als are not perfect, and some parts might seem mutually exclusive or contra- dictory if you don’t fully comprehend them.

Finally, follow through. If you’re asked for more information, provide it. If you don’t know how to provide it, treat that as another problem. Go back to the beginning of this chapter and try to figure it out. If you develop a reputation as someone who doesn’t follow up on requests for more informa- tion, you won’t even get a first reply.

Now let’s get ready to actually install OpenBSD.

2

Installation Preparations

I am script kiddie. Windows is warm and tasty; blowfish goes down hard.

It's not enough to install OpenBSD and get the machine running; you want a *successful* installation. A successful installation means that the system is configured to perform the job you intend it to do. A developer's laptop has very different requirements than those of a dedicated firewall, which might look very different from a web server. Proper planning will make your OpenBSD installation quick, easy, and successful. We'll spend a great deal of time on installation planning. Once you understand what you're doing, the actual installation process is pretty simple. Many of the problems people have with OpenBSD come from not understanding their many choices. The guidelines in this chapter cover most situations, but the final word on installing OpenBSD is the install document included in the release. For example, before installing OpenBSD on an i386 system, read *i386/INSTALL.i386* for your release.

OpenBSD Hardware

OpenBSD supports a wide variety of different hardware architectures. Some platforms, such as i386 and amd64, have extensive support, and their web pages and release notes list pages and pages of supported hardware. Others, such as SGI, support only very specific hardware models.

OpenBSD's currently supported hardware platforms include i386 (stan- dard PC), amd64 (64-bit PC-style hardware), sparc64 (Sun-style hardware), SGI (Silicon Graphics), and others. It also supports old platforms such as the VAX and tiny computers like the Zaurus. The platforms that I find interesting include the following:

i386 the Intel-compatible computer that has been popular for the past couple of decades

amd64 AMD's 64-bit extensions to the 32-bit i386, copied by Intel as EM64T, and sometimes called x64, x86_64, or x86-64 (this hardware can run both the 32-bit i386 and 64-bit amd64 versions of OpenBSD)

sparc64 64-bit Sun UltraSPARC and compatibles

macppc PowerPC-based Macintosh computers, from the iMac up until Apple switched to amd64 hardware

Zaurus Sharp Zaurus personal digital assistants (PDAs)

This chapter covers installing on the i386 and amd64 platforms. These are the standard 32-bit and 64-bit PC systems available from most vendors, and are what you're most likely to find on the secretary's desk while he is at lunch. They're architecturally close and install in exactly the same way. Old systems can run OpenBSD quite well. I've run OpenBSD/i386 quite nicely on a 166 MHz processor with 128MB of memory. You probably have some old system lying around that's perfectly adequate for learning OpenBSD.

In this book, I assume that your equipment is PCI bus or newer. I do not cover EISA hardware, or ISA other than the onboard chips in modern hardware. If you have an EISA SCSI card or network interface card (NIC) that still works, OpenBSD probably supports it. I assume that you still have the original hardware configuration floppy and remember how to set the IRQ and interrupt to match that assumed by the OpenBSD kernel. If you don't, recycle that

card and buy something built this millennium.
Note that the hardware must be in working condition. If your old Pentium machine kept crashing because its RAM is bad, using OpenBSD won't fix that problem. Also, OpenBSD will be most useful if the hardware meets certain minimum levels. I make recommendations based on my own experience, but again, the documentation gives the current and definitive requirements.

You can find a full list of supported hardware platforms at *http://www .OpenBSD.org/plat.html*. This page links to a page for each hardware platform, where you can get details on support for that hardware.

Supported Hardware The good news is that OpenBSD supports most hardware. The bad news is that it doesn't support everything. Generally speaking, OpenBSD supports the most common nonproprietary hardware. It might not support the very newest hardware, as the OpenBSD team doesn't get much access to hard- ware before it's released. Hardware that's a few months old has better sup- port than bleeding-edge gear.
To verify if OpenBSD supports your hardware, read the release notes for your platform or just give it a try.

Proprietary Hardware, Blobs, and Firmware Some hardware vendors want to keep the inner workings of their equip- ment secret so that competitors can't copy their designs. They hide their hardware designs in two common ways: proprietary hardware and binary object device drivers.
Some vendors will not provide documentation for their hardware. The vendor expects that the user will use the vendor-provided driver, and they provide drivers only for the most widely used commodity operating systems (such as Windows) or for a specific target market (Apple). Without docu- mentation, writing device drivers is tedious and difficult. Some hardware can be supported well without complete documentation, but much cannot. For example, OpenBSD's sparc64 platform didn't support newer Sparc pro- cessors for several years, until Sun released documentation.
Some vendors don't want to provide documentation, but do want users of open source operating systems to buy their hardware. These vendors provide drivers for their hardware in the form of binary objects, or *blobs*. This might sound reasonable at first, but the operating system must load these blobs into the kernel. The OpenBSD team has several objections to this. First, the code is not available for audit. If the blob has a security issue, or has some subtle interaction with the kernel that destabilizes the system, there's no way for the developers to resolve the problem. The blob might only be inefficient or wasteful, but it could negatively impact other kernel subsystems or even include backdoors. Lastly, OpenBSD's philosophy requires that all code be covered under a strict BSD license. In-kernel blobs are not free, and so OpenBSD will not support them.
Note that blobs are not the same as *firmware*. Firmware is a binary object a piece of hardware needs in order to run, and is loaded into the hardware itself, rather than into the operating system. You'll find firmware in almost every computer component: CPUs, motherboards, NICs, disk controllers, and so on. Firmware is never loaded into the kernel; the kernel loads the firm- ware into the card. The OpenBSD team considers this acceptable. The firmware lets the hardware provide its documented interface to the operat- ing system, and if it wasn't on the disk, it would be on the hardware itself.

Generally speaking, if OpenBSD developers have a piece of hardware, documentation for that hardware, and any use for the hardware, they will probably implement support for it. If not, that hardware won't work. In most cases, unsupported proprietary or blob-driven hardware can be replaced with more effective (and less expensive) open hardware.

Processors Processor brand is irrelevant. OpenBSD doesn't care if it's running on a CPU from Intel, AMD, Cyrix, or any other Intel-compatible processor. OpenBSD probes the CPU on boot and uses whatever chip features it recog- nizes. I've run very effective multimegabit firewalls on 486-class processors, but you'll be happiest with a 1 GHz or faster processor.
OpenBSD's multiprocessor support is not as broad as some other oper- ating systems, however. The OpenBSD

kernel mostly runs with the Big Giant Lock method, so the kernel can run on only one processor at a time. (Some small chunks of the kernel are not under the Big Giant Lock.) In practical terms, this means that the OpenBSD kernel won't make effective use of more than two processors or cores.

Does this mean you shouldn't use OpenBSD on your dual-eight-core- processor server? That depends on your expected server load. User processes scale well as long as they don't go into the kernel. Most web log analysis software, for example, runs almost entirely in user space, and you run massively parallel analysis jobs that scale quite well with the number of processors. Tasks such as forwarding packets, however, pass through the kernel. The hardware you need depends entirely on your expected workload.

Memory (RAM) Memory is good. The more memory you have, the happier you will be. Adding RAM accelerates your system more than any other generic improvement. You should have at least 256MB of RAM, and preferably at least 512MB. If you can get a couple of gigabytes in your system, OpenBSD will take full advantage of that memory.

If you keep adding memory, you will hit a point where your system has all the memory it needs, and more memory won't further improve perfor- mance. This could be as low as 128MB for a small firewall, a couple of giga- bytes for a desktop machine, or more for a large database server.

Most weird crashes and inexplicable, irreproducible problems can be traced back to bad memory, so be certain that the memory you are using is good. Memory is a common failure point in an old machine.

Hard Drives The smallest new hard drive you can buy today will run OpenBSD with vast amounts of space to spare. On older systems, I recommend at least 40GB of disk space—not because OpenBSD won't fit in less, but because you'll want room for additional files and software. The smaller your disk, the more closely you'll need to monitor its use. It's easy to fill a small disk when building a desktop environment from source, and disks are cheap these days. If you're running a small firewall from a flash drive, I recommend at least 512MB.[1]

Virtualization Many people run new operating systems in a virtual environment while they become accustomed to those systems. Some companies even have firm poli- cies mandating that all systems be run as virtual servers. OpenBSD runs fine in common virtual environments, and even has specific device drivers for virtualization systems such as VMware.

The hardware requirements for running OpenBSD on a virtual server are similar to the requirements for running OpenBSD on real hardware. Note that no operating system running in a virtual environment is as secure as that same operating system running on real hardware. Virtual environ- ments do not precisely replicate real hardware. Emulated CPUs have their own new and interesting bugs, virtual NICs have unique errors, and so on. Additionally, the environment providing the virtual server is itself an oper- ating system. An intruder can attack that underlying operating system, and once an intruder controls the virtualization server, clients running on that machine are much more vulnerable. No operating system can protect itself against its hardware. You must consider this risk when planning OpenBSD's role in your environment.

For learning about OpenBSD, however, a virtual environment is per- fectly adequate. I run OpenBSD machines in VirtualBox, on ESXi, and on Linux's KVM hypervisor without difficulty.

Multiple Operating Systems For many years, I ran multiple operating systems on a single computer. I remember being thrilled by my new 6GB hard drive because I could run FreeBSD, OpenBSD, Windows, and Linux on one computer with plenty of space for each operating system. This was the only way to run multiple operating systems on a single desktop, but advances in virtualization tech- nology have made this approach obsolete.

Rather than carefully dividing your desktop hard disk to run multiple operating systems and hoping that some proprietary disk-partitioning pro- gram won't munch its neighbor, I recommend running one operating system that supports a virtualization server and running your secondary operating systems as guests. OpenBSD supports running virtual guests with qemu.

Getting OpenBSD

Once you have hardware, you need OpenBSD. You can get OpenBSD on CD and over the Internet.

1. Yes, that's megabytes—you know, the unit below gigabytes. Yes, megabytes can apply to disks.

Official CDs Why would you buy an official CD in the 21st century?

The OpenBSD project is funded largely by sales of official CDs, along with related books, clothing, and so on. You can download a disk image from the Internet and burn your own installation disk, but purchasing an official set helps improve OpenBSD. The OpenBSD team tries to make the official CD sets interesting pieces in and of themselves, and usually packages them in some sort of geek-themed art. To get an official CD, go to the OpenBSD website and look for the Getting OpenBSD link. You can also find a whole bunch of OpenBSD-related merchandise. You can download installation images from the Internet, but they're not the same as the official CD set. The downloaded disk images don't contain any packages, lack the fancy physical packaging, and work on only one hardware architecture. You cannot download the images used for the official disks.

The main OpenBSD distribution point is in Canada, which increases delivery costs for those living on other continents. The OpenBSD website lists a variety of resellers that offer official OpenBSD CDs. Pick a vendor in your country and save on customs duties. If that option isn't available to you, you can at least pick a vendor on your same continent and save on shipping.

Internet Downloads The other OpenBSD installation methods require network access, either to download a complete image or to download files during the installation. Start by selecting an OpenBSD mirror site close to you. You can find a full list of mirrors at *http://www.OpenBSD.org/ftp.html*.

You can install the operating system files from an ISO image, FTP, HTTP, rsync, or even the Andrew File System (AFS) or Network File System (NFS) on some platforms. We will break the task into two parts: getting the target system to boot and getting the operating system files on the machine.

Mirror Site Layout All of the OpenBSD mirrors contain files and directories much like these:

5.1, 5.2, 5.3, and 5.4 The numbered directories contain OpenBSD releases. Most mirrors contain the last four releases. This particular server contains OpenBSD releases 5.1, 5.2, 5.3, and 5.4. ***Changelogs*** This directory contains collated OpenBSD Concurrent Version System (CVS) logs for those interested in OpenBSD's development. The casual user would probably find the web-based CVS browser more useful. ***distfiles*** This directory contains the files for building third-party soft- ware included in the OpenBSD ports collection (see Chapter 13). Not all mirror sites carry this very large directory.

doc This directory contains the OpenBSD FAQ and the PF FAQ, as well as translated and obsolete versions of the documentation. ***ftplist*** This file documents the official FTP and HTTP installation mir- rors. When you install via FTP or HTTP, the installer grabs this file to allow you to choose a mirror site close to you. ***OpenBGPD, OpenNTPD,* and *OpenSSH*** These three directories contain software that originated in the OpenBSD Project, but has been ported to other operating systems. *OpenBGPD* and *OpenNTPD* are newer projects aimed at creating OpenBSD-style Border Gateway Protocol (BGP) and Network Time Protocol (NTP) daemons. *OpenSSH* is the most widely deployed Secure Shell (SSH) client and server in the world, and is ported to all major operating systems. ***patches*** This directory contains patches for each earlier OpenBSD release. These patches address critical security and stability issues. ***snapshots*** This directory contains recent experimental OpenBSD ver- sions, snapshots of development between releases. If you want an early preview of the next version of OpenBSD, install a snapshot. These are works in progress; the developers provide them so that users can help test new code and catch any bugs before a release. If you want to be helpful, use a snapshot, but be warned: A snapshot might work beauti- fully, or it might savage your hardware and subtly corrupt your data. See Chapter 20 for more information about snapshots. ***songs*** Each version of OpenBSD includes a song written for the release. The *songs* directory contains each of these soundtracks. ***timestamp*** This file contains the time this mirror was last updated. ***tools*** This directory contains add-on tools useful for the OpenBSD Project's internal workings.

Release Directories Look within any given release directory on an OpenBSD FTP site or on a CD, and

you'll see the following:

• A directory for each architecture OpenBSD supports: *amd64*, *i386*, *sparc64*, and so on (on the CD, these directories are scattered among different disks as space permits)
• A *packages* directory containing precompiled software for this release (see Chapter 13)
• A *ports.tar.gz* file containing the compressed ports tree (see Chapter 13)
• A *src.tar.gz* file containing the operating system source code (see Chapter 20)
• A *sys.tar.gz* file containing the OpenBSD kernel source code (see Chapter 19)

• A *xenocara.tar.gz* file containing the OpenBSD version of the X Window System (see Chapter 19)
• A *tools* directory with software to help installation
• Several documents such as the release announcement (*ANNOUNCEMENT*), the basic instructions (*README*), and notes on OpenBSD's support for third-party software and different hardware

Look through your CD or the mirror site and find the directory for your hardware architecture. The architecture directories contain fairly similar files for every hardware platform.

First, find the installation instructions for your hardware. These are named *INSTALL* followed by the platform name (such as *INSTALL.i386*, *INSTALL.amd64*, and so on). Always read the installation instructions for your platform. While I've made every effort for accuracy in this book, OpenBSD continually changes, and the install document for your release is the last word on installation instructions.

Boot Media The OpenBSD boot media varies by hardware platform, and each hard- ware item has its own boot media requirements. You can't expect to boot a Zaurus or a VAX from a CD.

To easily boot the OpenBSD installer on i386 or amd64 hardware, use either a floppy disk or a CD (I usually recommend the latter). You can boot the installer from a USB disk, but the standard method requires bootstrap- ping from an OpenBSD machine, and nonstandard methods vary widely depending on available equipment.

If you cannot boot from a CD, use a floppy disk. OpenBSD provides one amd64 floppy image and three different i386 floppy disk images. If you're booting i386 from a floppy, I suggest downloading all floppy images. If you cannot boot using either method, you must use the Preboot eXecution Environment (PXE) diskless booting method, as described in Chapter 23. This method works well but requires a bit more preparation.

Choosing Install Media The boot disk can format your hard drive, configure your network, and copy installation files to disk. Boot media don't include those installation files, however. Installation files for i386 and amd64 machines come on an ISO image and over the network via FTP or HTTP.

If you intend to install this release on multiple OpenBSD machines, you might download the CD image that includes the installation files. It's much larger than the boot-only installer ISO image, however, so downloading it will require some sort of broadband connection.

If you're doing a single OpenBSD installation, or you don't have a CD drive, I recommend an HTTP installation. If you install from a reasonably close mirror site and have sufficient bandwidth, OpenBSD installs from HTTP quickly and reliably, and uses only about half as much bandwidth as downloading the installation ISO image. If you prefer, you can install from FTP as well.

Advanced users can install OpenBSD via the PXE method, as men- tioned in the previous section and covered in detail in Chapter 23.

Local Installation Servers One reason CDs are so popular is that you need to download files from the Internet only once, but can reuse your downloads to install OpenBSD on many machines. But CDs are physically fragile, and not every machine has a CD drive. If you want to install OpenBSD on several machines without using up bandwidth for each installation, download all of the installation files for your architecture. If you copy these files to a local FTP or web server, you can install OpenBSD on any number of machines from these files. To install from the local FTP server, you'll need a username and password for the FTP server.

To help save the OpenBSD Project on bandwidth costs, download only the directories for the architectures you need.

If you know exactly what you want to install, download just those file sets. You might have no respect for your own bandwidth, but please respect others' bandwidth.

File Sets

The release directory for each architecture contains several compressed files with names like *comp52.tgz*, *base52.tgz*, and so on. These *file sets* contain compressed OpenBSD installation files. By choosing to install particular file sets, you can pick how much functionality your OpenBSD system will have out of the box. For example, the documentation is kept in a separate distribution set. If you have documentation elsewhere, you might choose to not install it on a particular system. Also, intruders often make use of com- pilers, so you might not want them on a system you want to protect. But if this is your experimental "learning OpenBSD" machine, install everything. Each file set has a name and a version number. For example, one distri- bution set of OpenBSD in release 5.2 is *base52.tgz*. These are the base files of release 5.2. In the next release, this same file set will be called *base53.tgz*.

All architectures include all file sets, unless otherwise noted in the architecture's release notes. If this is your first OpenBSD installation, take a moment to decide which distribution sets you need. If at all possible, install them all on your test machine. You can always trim them down later for dedicated-purpose machines.

The following file sets are available:

bsd*, *bsd.mp*, and *bsd.rd

These file sets contain only OpenBSD kernels. The kernel is the heart of the operating system, containing the device drivers and basic system functions. Without a kernel, the system will not boot. The *bsd* kernel is for single-processor machines, while the *bsd.mp* kernel supports multiple

processors. The *bsd.rd* kernel contains the OpenBSD installer, basic user- land utilities, and the live system kernel. You can run only one kernel at a time.

baseXX.tgz

This contains OpenBSD's core programs—all the things that make OpenBSD Unix-like. The contents of */bin*, */sbin*, */usr/bin*, and */usr/sbin*; the system libraries; and all the miscellaneous programs you expect to find on a minimal Unix-like system are in this file set. You must install this file set.

etcXX.tgz

You might guess that this file set contains the files from */etc*, but it also contains other required files and directories, such as */var/log* and the root user's home directory. You must install this file set.

manXX.tgz

If you need the man pages for the programs in the base and *etc* file sets, install this distribution set. The man pages for other sets are installed with their respective file sets.

compXX.tgz

This file set contains C and C++ compilers, the assembler, libraries, tools, manuals, and the toolchain for each. You need this file set to develop or compile software, or use the ports collection (see Chapter 13). You do not need this file set if you plan to use only precompiled software pack- ages. At roughly 60MB, it is the largest file set for most platforms, but it's trivial compared to the size of modern hard disks. You might choose to not install it on a secure machine.

gameXX.tgz

This file set contains several simple games, based on games originally distributed in BSD 4.4. Some of these, such as *fortune(1)*, are considered UNIX classics, and old farts won't be happy unless they're installed. Others, such as */usr/games/wargames*, assume that you're familiar with early 1980s films. You don't need the games file set (unless you want to see what passed for "computer games" back when I was in high school).

xbaseXX.tgz

This contains the core of Xenocara, the OpenBSD version of the X Window System. If you want to use X, you need

this. Although you might not have a console or monitor on this computer, remember that X allows programs on this server to display remotely.
Most OpenBSD packages assume that you have installed this file set. If you find that a package crashes with errors about missing X libraries, you need this file set.

xetcXX.tgz

This contains the X configuration files. If you're using X for more than its libraries, you need this file set.

xfontXX.tgz

This contains X fonts. If you plan to use X on this machine's console, install this file set.

xservXX.tgz

This file set contains all the X video card drivers. If you plan to use X on this machine's console, install this file set.

xshareXX.tgz

This contains the X documentation. If you plan to use X on this machine's console, install this file set.

Partitioning

Partitions are logical subsections of a hard drive. OpenBSD can handle different partitions with their own unique privileges. You might make some partitions read-only so that files on them cannot be added, moved, or changed. OpenBSD might refuse to run programs on a specified partition, and it knows that device nodes should appear only on certain partitions. User files should not have setuid or setgid permissions, so the operating system won't recognize those privileges on files on the user data partition. While many operating systems support these sorts of privilege controls, OpenBSD uses them by default.
The most difficult part of installing OpenBSD is partitioning. When you don't know how partitions work, choosing partitioning can be troublesome.
If you're familiar with other Unix-like operating systems (such as some distributions of Linux), you might be accustomed to using a single large root partition and putting everything on it. This is a bad idea for several reasons. OpenBSD uses partitions as a security tool. A single large partition eliminates per-partition security and privileges. With your log files safely contained on one partition, a process or user gone amok cannot fill your entire drive. While it could fill a partition, you could still create and edit files on other partitions, giving you the flexibility you need to address the problem.
Unlike many installers that have fancy menus and graphic tools, OpenBSD's installer expects you to know how to use low-level disk manage- ment tools such as disklabel(8). Unlike with those operating systems, how- ever, OpenBSD can be installed in a much wider variety of ways on a wider variety of systems, all with a single installer.

If this is your first OpenBSD installation, use the default partitioning offered by the installer. OpenBSD will provide all its standard partitions, but adjust their sizes based on the size of your disk. The discussion here is based on a standard i386 installation on a fairly small disk.
If you've previously installed OpenBSD and you're installing it on a special-purpose machine, you might want special partitioning. In that case, get a piece of paper and a pencil, and write down the size of your hard disk, each partition you need, and each partition's desired size. Your special- purpose OpenBSD machine should almost certainly have all the same partitions as a default installation, but their sizes will differ. A web server has very different disk space requirements than a desktop machine, which in turn has different requirements than those of a firewall.
If you have a large disk, leave some space unallocated. Having partitions the size you need accelerates filesystem integrity checks; fsck(8) doesn't spend cycles integrity-checking unused disk space. On solid-state disks, unused space gives wear-leveling algorithms more cells to play with, increasing the life span of the disk and decreasing the odds of failure. It's better to have spare disk space you never need than to need disk space you don't have.

Standard OpenBSD Partitions The standard OpenBSD partitions are */* (root), swap space, */tmp*, */var*, */usr*, */usr/X11R6*, */usr/local*, */usr/src*, */usr/obj*, and */home*. If you create a custom layout and don't include one of

these partitions, the installer will put files that go into that partition into either your root or */usr* partition, quickly fill- ing them. If you want to create a partition after installation, you must find space on your disk for it. Unless you left unallocated space on your disk, you're better off reinstalling the whole system.

Root Partition The root partition holds the main OpenBSD configuration files and the most essential software needed to get the computer into single-user mode and on the network. Your system needs fast access to the root filesystem, so if you have multiple disks, put the root partition on the fastest (or smallest) one.

The root partition is the only one whose placement on disk is vitally important. Over the years, i386 systems have been repeatedly expanded to surpass their own limits—they're based on an architecture that could origi- nally handle only up to 640KB of RAM, after all! All modern operating system kernels work around these limits in a manner mostly transparent to users, but when the system is first booting, you're trapped within the hard- ware's limits.

Many old i386 systems have limits on hard drive size. They only recognize 128GB drives, 2TB drives, or some other number. The hardware BIOS can- not access anything beyond that limit. If you're using a computer that has a 128GB limit on hard drive size, and you put the kernel somewhere beyond the first 128GB of disk space, the computer will be unable to find the kernel

and thus unable to boot the system. Check your hardware manual before you get started. If the manual refers to a disk size limit, your entire root par- tition must fit within that limit.

If you violate this limit, your system will probably appear to work. The second you change the file */bsd*, however, it's likely that your computer will refuse to boot. Save yourself much pain by putting the root partition first on the disk, and making sure it's small enough to fit within the hardware's limits.

Swap Space Swap space is used for virtual memory. When your computer runs low on RAM, it starts to move information that has been sitting idle in memory into swap space. When the computer needs that information, it's loaded from virtual memory into real memory. This isn't necessarily bad for per- formance. Many programs spend the vast majority of their time executing only a small fraction of their code. OpenBSD is pretty good about figuring out which sections of memory can be moved into swap space and which are used too frequently to be swapped. If things go well, your computer will almost never need swap space.

OpenBSD also uses swap space during system failures. If the kernel panics, the computer writes the contents of system memory to the swap par- tition. This means that the swap partition must be, at its smallest, slightly larger than the amount of physical RAM in the system.

How much swap space do you need? The short answer is, "It depends on the system." OpenBSD defaults to allocating twice as much swap space as you have physical RAM. This isn't a bad rule, as long as you understand it's very general. A swap space three or four times the size of your physical memory won't hurt. If your computer uses more swap space than that, it's overloaded and will perform poorly.

If you find yourself using swap space often, consider increasing your physical memory instead. RAM is cheap. Also consider future upgrades. If your system has 2GB of RAM when you install OpenBSD, but you intend to increase that to 8GB, assigning 16GB of swap space is a good idea. Adding a swap partition later is difficult, unless you leave unallocated disk space when you install the software. (Note that, while you can swap to a file, OpenBSD can write only crash dumps to an actual swap partition.)

/tmp Directory The */tmp* directory is temporary space for all users on the system. Space requirements for */tmp* are generally a matter of opinion—after all, you can always use a chunk of space in your */home* directory for scratch space. Automated software installers frequently extract files into */tmp*. I usually recommend at least 3GB in */tmp*, but I do horrible things to my temp space. Many people use a */tmp* directory of 256MB or 512MB and get along just fine.

/var Partition The */var* partition contains frequently changing data, such as logs, data- bases, mail spools, temporary run files, websites, and so on. OpenBSD allo- cates about 5GB to */var* by default. This should be plenty for an educational installation. If you're building a web, database, or logging server, however, */var* should get the majority of your disk space. If you're on a really tiny sys- tem, you could use as little as 10MB for */var*.

/usr Partition The */usr* partition holds the operating system programs, compilers, libraries, and add-on programs. The majority of */usr* changes only when you upgrade your system. OpenBSD assigns */usr* 2GB by default, which is more than suf- ficient, even on a desktop system.

/usr/X11R6 Partition The */usr/X11R6* partition contains the X Window System programs and documentation. OpenBSD does package software linked against the X Window System, and a lot of software you might expect to find on servers (such as ImageMagick) requires X libraries.

If you are not going to install any X software, and plan to build all your own software without X, you don't need this partition. If you're in doubt, or if this is your first installation, keep this partition.

/usr/local Partition The */usr/local* partition contains add-on OpenBSD software, usually from packages (see Chapter 13). This can be much larger than the */usr* partition containing the core OpenBSD software. OpenBSD allocates 5GB of disk space to */usr/local* by default, and I've never needed more than that.

/usr/src Partition The */usr/src* partition is dedicated to the OpenBSD source code. On a dedicated-purpose machine that doesn't have a compiler, such as a fire- wall or a secure web server, you probably don't need a local copy of the source code. If you don't plan to upgrade this machine from source code, and you don't plan to use the source code as a reference on the local machine, you don't need this partition. If you're in doubt, keep it.

/usr/obj Partition The */usr/obj* partition is where OpenBSD builds new versions of the oper- ating system and Xenocara. The files in here are temporary; once you've installed a new OpenBSD version, you don't need these files any longer. Creating a new filesystem is faster than erasing the individual files in this kind of filesystem, so */usr/obj* is configured as its own partition.

If you don't intend to build a new OpenBSD from source code, you don't need */usr/obj*. If you find that you do need this partition later, you can either create it from unused space or mount it via NFS.

/home Partition The */home* partition can be described as "everything else." User directories go into */home*, as well as any random data that's meant for users. The fam- ily MP3 and photo collections should go in */home*, as well as your personal source code, email, and anything else you want to keep.

Creating Other Partitions OpenBSD supports up to 16 partitions per disk. If you want other parti- tions, you can create them using the installer. Does your company have a policy that all add-on software must go in */opt* or */usr/companyname*? Fine, create that partition. The OpenBSD standards are not a straitjacket, but rather a starting point. You own the system. Make it behave according to your needs.

Partition Filesystems

The words *filesystem* and *partition* are often used interchangeably. They are closely related, but two different things. A filesystem is a method of allocat- ing and tracking files that are on a partition. You can back up and restore a filesystem, but if a partition is damaged, you're in much worse trouble.

OpenBSD uses the standard Fast File System (FFS) by default. FFS has been around for decades, and is both well debugged and well understood. Unfortunately, with its default settings, it can handle partitions only up to slightly less than 1TB in size. Modern disks make partitions of that size common.

If a partition is 1TB or more in size, the installer automagically formats it with FFS version 2 (FFS2). In Chapter 8, we'll cover how to adjust your filesystems to exactly fit your needs.

Multiple Hard Drives

Disk input/output is usually the slowest part of a computer. If you have more than one hard drive, you can use those drives to accelerate your system performance.

First, make sure that each drive is on its own port. SCSI and SATA drives usually accommodate one drive per port (unless you specifically use a port multiplier), but IDE drives usually attach two devices per port. Each port has a maximum throughput. It does no good to attach two fast drives to one port, as the drives compete for the one port's throughput.

In general, when you have multiple drives, you want to split the read and write activity between the drives. I usually put the data I'm serving on

one disk and the important system files on another. If I'm building a data- base server, I might dedicate one disk to swap space and */var*, while assign- ing all other partitions to the other disk.
Split your swap space between the drives. Be sure that at least one par- tition is large enough to hold the contents of your physical RAM, so that OpenBSD can do a crash dump if needed. OpenBSD cannot split a crash dump between two different swap partitions.
If you're a more experienced OpenBSD user, you can use multiple hard drives to create a redundant disk with software RAID. We'll cover how to do that in Chapter 9.
If your second drive is much slower than your main system drive, don't bother using it. A computer runs only as fast as its slowest component, so adding that old IDE drive to your SATA system will drag down the whole machine. Not only will its presence degrade performance for the whole sys- tem, but it's also probably much older than your main drive and far more likely to fail.

Understanding Partitions

As a historical accident, i386 and amd64 systems have two different types of partitions. OpenBSD refers to the first as *MBR partitions* and the second as *disklabel partitions* (or just *partitions*).

MBR Partitions MBR partitions, also known as *primary partitions*, are universally understood by operating systems that run on i386 hardware. Every hard drive has four MBR partitions. In most cases, only one partition has any space allocated to it; the other three partitions have zero size. If you want to install multiple operating systems on a single disk, then each operating system needs its own MBR partition.
Most operating systems manage MBR partitions with a program called *fdisk*. It's not the same program, mind you—OpenBSD's fdisk(8) is not the same as Microsoft's Fdisk, which is different from the program for Linux, FreeBSD, OpenSolaris, and so on. Any operating system's fdisk can see MBR partitions that belong to other operating systems, and while they might not recognize what's on the MBR partition, they will recognize that space has been allocated for *something* and will warn you about overwriting it. Unfortunately, not all fdisk programs play nicely with each other. Do not partition disks for one operating system with another operating system's tools.[2]
With the advent of cheap virtualization, installing multiple operating systems on a single disk is no longer advisable. Assign each disk a single MBR partition that fills the entire disk, and give the other three MBR parti- tions zero size. You will see an example of how to do this in Chapter 3.

2. I'm assured by OpenBSD developers that any fdisk should suffice for any operating system. Having been repeatedly savaged by buggy fdisk programs, I find myself unable to give you carte blanche to try this.

Disklabel Partitions BSD did not originate on i386 hardware; it had its own disk-partitioning system, based on labeling the disk's partitions. When BSD was ported to i386, the disklabel was nailed up inside an MBR partition. When someone speaks of "partitions" in OpenBSD, they almost certainly mean disklabel partitions.
One disklabel can support 16 partitions. If you need more than 16 par- titions, you must create a second MBR partition and add more disklabels. I would suggest that if you need more than 16 partitions on a single disk, you took a wrong turn somewhere in your decision-making process. Step back and reassess what you want to accomplish and how you're going about it.
Foreign operating systems do not recognize OpenBSD disklabels. BSD-based operating systems might appear to understand them, but the disklabel formats used on the various BSD-derived systems have diverged in the past 20 years. Use only OpenBSD disk tools to manage OpenBSD partitions.

Understanding Disklabels

The OpenBSD installer expects you to understand disklabels. You can avoid learning about disklabels by blindly accepting the default partition- ing OpenBSD offers, but that won't take you very far. Disklabels might look intimidating to the new user and require some basic math, but they aren't that difficult once you walk through them slowly. You need to understand disk geometry first.

Sectors and Lies Once upon a time, disk drives had clearly defined geometry. Each disk was actually round, and it spun inside the hard drive. The manufacturer divided each disk into tiny sections, called *sectors*. Each sector

had a num- ber, with sector 0 at the beginning of the disk and the sectors numbered sequentially until the end of the disk. Sectors were gathered into rings, or *tracks*. Stacks of tracks were aggregated into *cylinders*. Each disk drive had a number of *heads*—data-reading devices that read information from the disk as the disk spun beneath them. Taken as a whole, sectors, tracks, and cylin- ders described the disk *geometry*.

This all seems simple enough, but today you can't actually count on disk sectors to actually map to anything useful. Over the years, both hard drive manufacturers and operating systems have set and broken limits. This applies to all aspects of machine design, from the 640KB memory limit to the 504MB disk limit. Hard drive manufacturers avoided these limits by tricking the system BIOS and/or the operating system.

If you're a hard drive manufacturer making a hard drive with 126 sectors per track, but the most popular operating system can accept only 63 sectors per track, you have a problem. The easy solution is to teach your hard drive to lie. If you claim you have half as many sectors per track

but twice as many platters, the numbers still add up, and you can still provide unique sector numbers. Every hard drive manufacturer chooses to lie in a slightly different way. The most obvious examples are flash drives (which still report cylinders, sectors, and tracks, even though they're not round and don't spin[3]) and hardware RAID (which reports the same information about several disks as if they were one). If you read about the history of hard drives, you'll discover all sorts of interesting lies.

By the time disk geometry information reaches the operating system, it has been through one or more translations. Reach into your head, find the button that says "Accept What You're Told," and press it as you repeat the following: Disks are divided into sequentially numbered sectors. Partitions fill a number of consecutive sectors. Sectors are grouped into cylinders, based on the number of heads in the drive. Partitions end on cylinder boundaries.

Sectors and Disklabels The installer will display your disk's disklabel. (You can also see the disklabel once the system is installed and running, as discussed in Chapter 8.)

We'll look at the disk's physical information first. While the physical information doesn't usually directly impact the installation, you need to know how to read it if something goes wrong.

```
❶ # /dev/rsd0c: ❷ type: SCSI ❸ disk: SCSI disk ❹ label: DSA2CW120G3 ❺ duid: adb697598fa0a010
flags: ❻ bytes/sector: 512 sectors/track: 63 tracks/cylinder: 255 sectors/cylinder: 16065 cylinders: 14593 ❼ total sectors: 234441648 ❽
boundstart: 64 ❾ boundend: 234436545
drivedata: 0
```

Except for the device unique identifier (DUID), you cannot change any of these entries without changing the underlying hardware.

The first entry is the device name, /dev/rsd0c ❶. The leading /dev means that this is a device node. The rsd0c is the disk name. sd means that this drive uses the sd(4) device driver, and the 0 means that this is the first drive OpenBSD found and attached. (This is usually, but not always, BIOS drive 0.) The leading r means that we're addressing the disk in raw mode, while the tailing c means that we're examining disklabel partition *c*. Disklabel

3. Yes, you can make flash drives spin. But a flash drive doing 5400 RPM has a whole set of problems beyond the scope of normal systems administration.

partition *c* always matches the entire MBR partition containing this disklabel. Almost any disk that isn't explicitly IDE will probably show up as a SCSI disk.

The type ❷ is a general label describing the disk's physical interface. Any IDE disk will show up as ESDI (Enhanced Small Device Interface), while SCSI, SAS, SATA, and almost every other type of disk has type SCSI.

The disk field ❸ shows what sort of disk is attached to this interface. Here, it shows a SCSI disk, but we knew that already from the type.

The label ❹ displays the manufacturer's name and/or the drive model number. In the case of virtualized servers, this shows virtual drive or some- thing similar.

The duid ❺ is the DUID for this disk. If you've ever managed a system with more than a couple of disks in it, you know how easy it is to confuse disks. The hardware BIOS identifies disks by the physical port they're attached to. If

you need to replace a SATA or SCSI card, and you get the disks mixed up as you rerun cables, you will have a hard time finding your boot drive again. By using the DUID in your system configuration instead of the BIOS-assigned device name, you will always have the same disk used for the same purpose. As noted earlier, the DUID is the one editable field in the top of the disklabel information.

The bytes per sector, sectors per track, tracks per cylinder, and sectors per cylinder ❻ all describe the disk's geometry. These numbers are all lies, but the total number of sectors on the disk ❼ *is* accurate. You also see the first sector you may fill with disklabel partitions ❽, and the last sector you may use ❾. (You lose a few sectors due to the hard drive's geometry trans- formations. Don't try to hold the hardware accountable. You can't win that argument.)

The next section displays the disklabel partitions, and you can alter it as needed. Here's a disklabel from my desktop:

```
16 partitions: # ❶ size ❷ offset ❸fstype ❹[fsize bsize cpg] ❺ a: 2097121 64 4.2BSD 2048 16384 1 # / ❻ b: 4698424 2097185 swap
❼ c: 312581808 0 unused
d: 8388576 6795617 4.2BSD 2048 16384 1 # /tmp e: 16736864 15184193 4.2BSD 2048 16384 1 # /var f: 4194304 31921057 4.2BSD
2048 16384 1 # /usr g: 2097152 36115361 4.2BSD 2048 16384 1 # /usr/X11R6 h: 20971520 38212513 4.2BSD 2048 16384 1 #
/usr/local i: 4194304 59184033 4.2BSD 2048 16384 1 # /usr/src j: 4194304 63378337 4.2BSD 2048 16384 1 # /usr/obj k: 245003968
67572641 4.2BSD 2048 16384 1 # /home
```

This disklabel declares that it has 16 partitions, but lists only 11. The disklabel has space for 16 partitions, but like the MBR partition table, not all of them have space allocated to them. As with most configuration files in Unix-like operating systems, a hash mark (#) indicates the beginning of a comment. The comments here give the headers for the table above.

The first column is the partition letter. A unique letter identifies each disklabel partition. The first partition in our example is *a* ❺, the second is *b* ❻, the third is *c* ❼, and so on.

The size ❶ is the number of sectors the drive uses. In this example, partition *a* fills 2097121 sectors, partition *b* 4698424 sectors, and partition *c* 312581808 sectors.

The offset ❷ is the number of sectors from the beginning of the MBR partition where the disklabel partition begins. If a disk is bootable, it has a master boot record (MBR) flagging it as such. The MBR record takes the first 63 disk sectors, numbers 0 through 62. The first sector available for a disklabel partition is sector number 63. Partition *a* begins on sector 64 in order to correctly align with the memory cells in solid-state disks.

Take a look at partition *b*. It has an offset of 2097185, meaning it starts in sector 2097185. How do we get there? Well, partition *a* starts in sector 64 and has a size of 2097121. 2097121+64=2097185, or the first free sector after partition *a* ends. This seems perfectly sensible until you look at partition *c*. Disklabel partition *c* is magical. On every disklabel partition, *c* represents the entire disk. It has an offset of 0 and a size equal to the number of sec- tors on the disk. You cannot put a filesystem on partition *c*; it's there only for reference. Partition *d* picks up where partition *b* left off.

The fstype ❸ marks the type of filesystem on this partition. OpenBSD filesystems, such as partition *a*, are labeled as 4.2BSD. (The OpenBSD filesystem is no longer exactly the same as that from BSD 4.2, mind you.) Partition *b* is swap space.

The next two columns ❹ display the fragmentation behavior of the file- system on this partition. These values are set by the filesystem creation tool when putting the filesystem on the partition, and should not be changed by hand. If you're curious, read newfs(8) and its related man pages. The fsize is the fragment size for any file fragments on the partition. The *b* is the size of a block on disk, in bytes. We talk about FFS fragmentation in Chapter 8. All you really need to know at this point is that FFS and FFS2 are both highly fragmentation-resistant, and neither requires any sort of defragmentation process.

The last column shows the number of cylinders per cylinder group. This is almost always 1 for modern disks.

One interesting thing is that the disklabel can be considered a configu- ration file for formatting a disk. You could

save this disklabel to a file, get an identical hard drive, write this label to that new disk, and perfectly dupli- cate the partitioning of the old disk on the new.
If at any time you feel confused about your partitioning, print out your current disklabel and compare it to how you would like your system to look.