®

# FREEBSD

# FREEBSD ABSOLUTE ABSOLUTE ®

**THE COMPLETE GUIDE TO FREEBSD**

PRAISE FOR *ABSOLUTE FrEEBSD*

"Even longtime users of FreeBSD may be surprised at the power and features it can bring to bear as a server platform, and *Absolute BSD* is an excellent guide to harnessing that power." —UNIXREVIEW.COM

" . . . provides beautifully written tutorials and reference material to help you make the most of the strengths of this OS." —LINUXUSER & DEVELOPER MAGAZINE

" . . . packed with a lot of information." —DAEMON NEWS

"When was the last time you could physically feel yourself getting smarter while reading a book? If you

are a beginning to average FreeBSD user, *Absolute FreeBSD* . . . will deliver that sensation in spades." —RICHARD BEJTLICH, TAO SECURITY

"By far the best FreeBSD book I have ever owned is *Absolute FreeBSD*, 2nd Edition by No Starch Press." —BSD ZEALOT

"Master practitioner Lucas organizes features and functions to make sense in the development environment, and so provides aid and comfort to new users, novices, and those with significant experience alike." —SCITECH BOOK NEWS

# Absolute

# FreebsD® 3rD eDition

## the Complete Guide to FreebsD

by Michael W. Lucas

San Francisco



ISBN-10: 1-59327-892-6 ISBN-13: 978-1-59327-892-2

Publisher: William Pollock Production Editor: Janelle Ludowise Cover and Interior Design: Octopod Studios
Developmental Editor: William Pollock Technical Reviewers: John Baldwin, Benno Rice, and George V.
Neville-Neil Copyeditor: Julianne Jigour Compositor: Susan Glinert Stevens Proofreader: James Fraleigh Indexer:
Nancy Guenther

For information on distribution, translations, or bulk sales, please contact No Starch Press, Inc. directly: No Starch Press, Inc. 245 8th Street, San Francisco, CA 94103 phone: 1.415.863.9900; info@nostarch.com www.nostarch.com

*Library of Congress Cataloging-in-Publication Data*

Lucas, Michael, 1967-
Absolute FreeBSD : the complete guide to FreeBSD / Michael W. Lucas. -- 2nd ed.
p. cm. Includes index. ISBN-13: 978-1-59327-151-0 ISBN-10: 1-59327-151-4 1. FreeBSD. 2. UNIX (Computer file) 3. Internet service providers--Computer programs. 4. Web servers--Computer programs. 5. Client/server computing. I. Title. QA76.76.O63L83 2007
004'.36--dc22

2007036190



## About the Author

After using Unix since the late ’80s and spending twenty-odd years as a network and sytem administrator specializing in building and maintain- ing high-availability systems, Michael W. Lucas now writes about them for a living. He’s written more than 30 books, which have been translated into nine languages. His critically acclaimed titles include *Absolute OpenBSD*, *Cisco Routers for the Desperate*, and *PGP & GPG*, all from No Starch Press. Learn more at *https://mwl.io/*.

## About the Technical Reviewers

**John Baldwin** joined the FreeBSD Project as a committer in 1999. He has worked in several areas of the system, including SMP infrastructure, the network stack, virtual memory, and device driver support. John has served on the Core and Release Engineering teams and organized several FreeBSD developer summits.

**Benno Rice** has been using FreeBSD since 1995 and has been a com- mitter since 2000 when he started the PowerPC port. Since then he has worked in a variety of areas and for a number of FreeBSD-using companies. He has also served on the Core Team and presented on FreeBSD-related topics at several conferences.

**George V. Neville-Neil** works on networking and operating system code for fun and profit. His areas of interest are code spelunking, operating systems, networking, and time protocols. He is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System* (Addison-Wesley Professional, 2004).

# Brief Contents

# Contents in Detail

## 3 InstAlIIng 29



## 4 stArt Me up! tHe Boot process 49

## 15 MAKIng Your sYsteM useFul 335



## 16 custoMIZIng soFtwAre wItH ports 361

## 20 sMAIl sYsteM servIces 491





## 21 sYsteM perForMAnce And MonItorIng 525

**22 JAIls 563**



**23 tHe FrInge oF FreeBsd 583**

# Foreword

I am happy to write the foreword to Michael Lucas's third edition of *Absolute FreeBSD*. For 15 years, Michael's *Absolute* series has provided the definitive guide to BSD software, filling in the whats and whys left unex- plained by the detailed but largely factual documenta- tion. And, as its name implies, it

distills to its essence the enormous volume of FreeBSD documentation so that those new to the system can get up to speed quickly.

Michael is an important contributor to the FreeBSD community. He has filled many of the roles that contributors can take: answering questions, fill- ing in pieces of missing documentation, helping to make connections in the community, and generally identifying and facilitating the things that need to be done. Michael has interacted with thousands of people: hobbyists,

professional software developers, system administrators, and university professors. Much of his real-world experience and understanding of what people are trying to get done has been distilled into this book.

I have been involved with the BSD software since its beginning in 1977 as a student project of my office mate, Bill Joy, at the University of California at Berkeley. By 1980, the BSD distributions had grown from a few programs that could be added to an AT&T UNIX system to a complete system coordinated by four people

who called themselves the Computer Systems Research Group (CRSG). By 1983, the socket interface had been designed and TCP/IP had been implemented underneath it, allowing a small set of trusted external contributors to log into the CSRG development machines over the ARPAnet (which later became the internet) and directly update the sources using SCCS, a very early source code control system. The CSRG staff could then use SCCS to track changes and verify them before doing distributions. This structure formed the basis for the current BSD-based projects once BSD was spun off from the university as open source in 1992.

Starting with the open-source distribution, FreeBSD initially ran on only the early PC computers. Over the past quarter century, thousands of developers have contributed to FreeBSD to make it into a powerful net- work operating system with state-of-the-art features that runs on all the modern computing platforms. FreeBSD powers core internet companies worldwide. From Netflix movie distribution to WhatsApp messaging, from Network Appliance and Dell/Isilon storage products to Juniper routers, from the foundation of Apple's iOS to the base libraries and services of Google's Android, it is hard to throw a rock at the internet without hit- ting FreeBSD. However, FreeBSD is not the product of any one company, but of a large open source community: the FreeBSD Project, made up of developers, users, and countless supporters and advocates. While you can, as many people do, use FreeBSD simply as a piece of software without ever interacting with that community, you can significantly enrich your FreeBSD experience by becoming a part of that community.

Whether you are a first-time user or a kernel hacker, the resources avail- able via the *http://www.freebsd.org/* website, countless mailing lists, regional user groups, and conferences can be invaluable. Have a question? Just email *questions@FreeBSD.org*, and one or more of the hundreds of volunteers will undoubtedly answer it. Want to learn more about the exciting new features coming in future FreeBSD versions? Read the Project's quarterly status reports or development mailing lists, or attend one of the many regional BSD conferences taking place around the world.

These resources are a product of the FreeBSD Project and its commu- nity, a large number of collaborating individuals and companies, as well as the FreeBSD Foundation, a nonprofit organization coordinating funding, legal resources, and support for development work and community activi- ties. Michael's easy-to-use book provides a gateway for newbies to benefit from this community's expertise and to become active users of FreeBSD themselves.

FreeBSD is open source software, available for you to use and dis- tribute at no charge. By helping to support, advocate, or even develop FreeBSD, you can give back to the FreeBSD Project and help this commu- nity grow.

Whether you are a new user of FreeBSD or an experienced one, I am confident you will find Absolute FreeBSD a book you want to keep close at hand.

Marshall Kirk McKusick FreeBSD Committer Treasurer,
FreeBSD Foundation Berkeley, California January 2018

# Acknowledgments

This book would not exist without decades of support from the FreeBSD community. Many people have told me that they reach for my books to learn how to accomplish something. What they don't see is how many times I've reached out to mailing lists, forums, and user groups to get that same sort of help—not to mention all the times I've used other people's archived discussions to fig- ure out where I went horribly wrong. In addition to all those folks who've gone before me, though, I need to name those who helped me on this particular book.

Gavin Atkinson, Diane Bruce, Julian Elischer, Lars Engels, Alex Kozlov, Steven Kreuzer, Ganael Laplanche, Greg "Groggy" Lehey, Warner Losh, Remko Lodder, Ruslan Makhmatkhanov, Hiren Panchasara, Colin Percival, Matthew Seaman, Lev Serebryakov, Carlo Strub, Romain Tartière, and Thomas Zander all provided vital feedback on earlier versions of this book. Some of them read individual chapters that they have special expertise in, while others read the whole blasted book whether they knew the topic or not. Both kinds of feedback are invaluable. John Baldwin, Benno Rice, and George Neville-Neil collaborated on performing a final technical review, catching errors that ranged from the subtly horrific to the blatantly appalling. Any errors that remain in this book were introduced by myself, despite all these people's best efforts.

I've also received years of support from Allan Jude and Benedict Reuschling of the BSDNow (*https://www.bsdnow.tv/*) podcast, along with alumnus Kris Moore. They've backed my work even when they had no idea what the heck I was doing. Their show is a great source of BSD-related news, education, and gossip. (It's a community. There's always gossip.) Just this week, they walked me through understanding the scheduler in a way I never have before.

Bert JW Regeer donated $800 to the FreeBSD Foundation for the dubi- ous privilege of being abused in this book. I sincerely thank Bert for being a good sport, and handling all the indignities I heap upon him with grace and aplomb.
Of all the folks who back me on Patreon, I must especially thank Stefan Johnson and Kate Ebneter. Because that's what their Patreon reward levels say I'll do. So: thank you!
Janelle over at No Starch Press had the unenviable job of shepherding this book through production, which is kind of like herding cats except the cats are angry and have switchblades. Thank you for dragging this tome across the finish line. I also need to thank the rest of the No Starch staff, who suffered through transforming my meandering babblings into a real book.
And as always, my gratitude to my amazing wife Liz.

# Introduction

Welcome to *Absolute FreeBSD*! This book is a one-stop shop for system administrators who want to build, configure, and manage FreeBSD servers. It will also be useful for those folks who want to run FreeBSD on their desktops, embedded devices, server farms, and so on. By the time you finish this book, you should be able to use FreeBSD to provide network services. You should also understand how to manage, patch, and maintain your FreeBSD systems and have a basic understanding of network- ing, system security, and software management. We'll discuss FreeBSD ver- sions 11 and 12, which are the most recent versions at the time this book is being released; however, most of this book applies to earlier and later ver- sions as well.

## What Is FreeBSD?

*FreeBSD* is a freely available Unix-like operating system popular with inter- net service providers, in appliances and embedded systems, and anywhere that reliability on commodity hardware is paramount. One day last week, FreeBSD miraculously appeared on the internet, fully formed, extruded directly from the mutant brain of its heroic creator's lofty intellect. Just kidding—the truth is far more impressive.

FreeBSD is a result of almost four decades of continuous development, research, and refinement. The story of FreeBSD begins in 1979, with BSD.

***BSD: FreeBSD's Granddaddy*** Many years ago, AT&T needed a lot of specialized, custom-written com- puter software to run its business. It wasn't allowed to compete in the computer industry, however, so it couldn't sell its software. Instead, AT&T licensed various pieces of software and the source code for that software to universities at low, low prices. The universities could save money by using this software instead of commercial equivalents with pricey licenses, and university students with access to this nifty technology could read the source code to see 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 licensing 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 if the operating system itself had a problem, the people who lived with the system on a day-to-day basis 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 run- ning processes, also known as *job control*. The Unix S51K filesystem made system administrators bawl like exhausted toddlers, so they replaced it with the Fast File System (FFS), whose features have spread into every modern filesystem. Many small, useful programs were written over the years, gradu- ally replacing entire swaths of Unix.

The Computer Systems Research Group (CSRG) at the University of California, Berkeley, participated in these improvements and also acted as a central clearinghouse for Unix code improvements. CSRG collected changes from other universities, evaluated them, packaged them, and distributed the compilation 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 collection of software came to be known as the *Berkeley Software Distribution*, or *BSD*.

BSD users took the software, improved it further, and then fed their enhancements back into BSD. Today, we consider this to be a fairly standard



way for an open source project to run, but in 1979 it was revolutionary. BSD was also quite successful; if you check the copyright statement on an old BSD system, you'll see this:

> Copyright 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
> The Regents of the University of California. All rights reserved.

Yep, 15 years of work—a lifetime in software development. How many other pieces of software are not only still in use, but still in active devel- opment, 15 years after work began? In fact, so many enhancements and improvements went into BSD that the CSRG found that over the years, it had 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 liberal 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.

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.) There's not even a requirement that you share your changes with the original authors! People were free to take BSD and include it in proprietary products, open source products, or free products—they could even print it out on punch cards and cover the lawn with it. You want to run off 10,000 BSD CDs and distribute them to your friends? Enjoy. Instead of *copyright*, 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 surprisingly, com- panies such as Sun Microsystems jumped right on it: it was free, it worked, and plenty of new graduates had experience with the technology—including Bill Joy, one of Sun's founders. One company, BSDi, was formed specifically to take advantage of BSD Unix.

***The AT&T/CSRG/BSDi Iron Cage Match*** At AT&T, UNIX work continued apace even as the CSRG went on its merry way. AT&T took parts of the BSD Unix distribution, integrated them with its UNIX, and then relicensed the result back to the universities that pro- vided those improvements. This worked well for AT&T until the company

was broken up and the resulting companies were permitted to compete in the computer software business. AT&T had one particularly valuable property: a high-end operating system that had been extensively debugged by thousands of people. This operating system had many useful features, such as a variety of small but powerful 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 maintaining the university relationship that had given it such an advanced operating system in the first place.

Berkeley's public release of the BSD code in 1992 was met with great displeasure from USL. Almost immediately, USL sued the university and the software companies that had taken advantage of the software, particu- larly BSDi. The University of California claimed that the CSRG had com- piled BSD from thousands of third-party contributors unrelated to AT&T, and so it was the CSRG's intellectual property to dispose of as it saw fit.

This lawsuit motivated many people to grab a copy of BSD to see what all the fuss was about, while others started building products on top of it. One of these products was 386BSD, which would eventually be used as the core of FreeBSD 1.0.

In 1994, after two years of legal wrangling, the University of California lawyers proved that the majority of AT&T UNIX was actually taken in its entirety from BSD, rather than the other way around. To add insult to injury, AT&T had actually violated the BSD license by stripping the CSRG copyright from files it had assimilated. (Only a very special company can violate the world's most generous software license!) A half-dozen files were the only sources of contention, and to resolve these outstanding issues, USL donated some of them to BSD while retaining some as proprietary information.

Once the dust settled, a new version of BSD Unix was released to the world as BSD 4.4-Lite. A subsequent update, BSD 4.4-Lite2, is the grand- father of the current FreeBSD, as well as ancestor to every other BSD vari- ant in use today.

***The Birth of FreeBSD*** One early result of BSD was 386BSD, a version of BSD designed to run on the cheap 386 processor.[1] The 386BSD project successfully ported BSD to Intel's 386 processor, but it stalled. After a period of neglect, a group of 386BSD users decided to branch out on their own and create FreeBSD so they could keep the operating system up to date. (Several other groups started their own branches off of 386BSD around the same time, of which only NetBSD remains.)

386BSD and FreeBSD 1 were derived from 1992's BSD release, the sub- ject of AT&T's wrath. As a result of the lawsuit, all users of the original BSD were requested to base any further work on BSD 4.4-Lite2. BSD 4.4-Lite2 was not a complete operating system—in particular, those few files AT&T had retained as proprietary were vital to the system's function. (After all, if those files hadn't been vital, AT&T wouldn't have bothered!) The FreeBSD development team worked frantically to replace those missing files, and FreeBSD 2.0 was released shortly afterward. Development has continued ever since.

1. At the time, several thousand dollars for a computer was dirt cheap. You young punks have no idea how good you have it.

Today, FreeBSD is used across the internet by some of the most vital and visible internet-oriented companies. Netflix's content delivery system runs entirely on FreeBSD. IBM, Dell/EMC, Juniper, NetApp, Sony and many other hardware companies use FreeBSD in embedded systems where you'd never even know it unless someone told you. The fact is, if a company needs to pump serious internet bandwidth, it's probably running FreeBSD or one of its BSD relatives.

FreeBSD also finds its way into all sorts of embedded and dedicated- purpose devices. Do you have a PlayStation 4? Congratulations, you're run- ning FreeBSD. I hear a root shell is hard to get on one of them, though.

Like smog, spiders, and corn syrup, FreeBSD is all around you; you simply don't see it because FreeBSD just works. The key to FreeBSD's reli- ability is the development team and user community—which are really the same thing.

## FreeBSD Development

There's an old saying that managing programmers is like herding cats. Despite the fact that the FreeBSD development team is scattered across the world and speaks dozens of languages, for the most part, the members work well together as parts of the FreeBSD community. They're more like a pride of lions than a collection of house cats. Unlike some other projects, all FreeBSD development happens in public. Three groups of people are responsible for FreeBSD's progress: committers, contributors, and users.

***Committers*** FreeBSD has about 500 developers, or committers. *Committers* have read-and- write access to the FreeBSD master source code repository and can develop, debug, or enhance any piece of the system. (The term *committer* comes from their ability to *commit* changes to the source code.) Because these commits can break the operating system in both subtle and obvious ways, commit- ters carry a heavy responsibility. Committers are responsible for keeping FreeBSD working or, at worst, not breaking it as they add new features and evaluate patches from contributors. Most of these developers are vol- unteers; only a handful are actually paid to do this painstaking work, and most of those people are paid only as it relates to other work. For example, Intel employs committers to ensure that FreeBSD properly supports its net- work cards. FreeBSD has a high profile in the internet's heavy-lifting crowd, so Intel needs its cards to work on FreeBSD.

To plug yourself into the beehive of FreeBSD development, consider subscribing to the mailing list *FreeBSD-hackers@FreeBSD.org*, which contains

most of the technical discussion. Some of the technical talk is broken out into more specific mailing lists—for example, fine details of the networking implementation are discussed in *FreeBSD-net@FreeBSD.org*.

Every few years, the committer team elects a small number of its mem- bers to serve as a core team, or *Core*. Core's work is simultaneously vital, underrated, and misunderstood. Core is theoretically responsible for the overall

management of FreeBSD, but in practice, it manages little other than resolving personality disputes and procedural conflicts among com- mitters. Core also approves new committers and delegates responsibility for large parts of FreeBSD to individuals or groups. For example, it delegates authority over the ports and packages system to the ports management team. Core does not set architectural direction for FreeBSD, nor does it dictate processes or procedures; that's up to the committers, who must agree en masse. Core does suggest, cajole, mediate, and inspire, however.

Core also experiences the worst part of management. Some of the key functions of management in a company are oversight, motivation, and handling problems between people. Oversight is provided by the millions of users who will complain loudly when anything breaks or behaves unexpect- edly, and FreeBSD committers are self-motivated. The ugly part of manage- ment is settling squabbles between two people, and that's the part Core gets stuck with. The status one gets from saying "I'm in Core" is an insuf- ficient reward for having to manage the occasional argument between two talented developers who've gotten on each other's nerves. Fortunately such disagreements are rare and usually resolved quickly.

***Contributors*** In addition to the committer team, FreeBSD has thousands of contributors. *Contributors* don't have to worry about breaking the main operating system source code repository; they submit their patches for consideration by com- mitters. Committers evaluate contributor submissions and decide what to accept and what to reject. A contributor who submits many high-quality patches is often asked to become a committer themselves.

For example, I spent several years contributing to FreeBSD whenever the urge struck me. Any time I feel that I've wasted my life, I can look at the FreeBSD website and see where my work was accepted by the committers and distributed to thousands of people. After I submitted the first edition of this book to the publisher, I spent my spare time submitting patches to the FreeBSD FAQ. Eventually, some members of the FreeBSD Documentation Project approached me and asked me to become a committer. As a reward, I got an email address and the opportunity to humiliate myself before thousands of people, once again demonstrating that no good deed goes unpunished.

If I had never contributed anything, I'd remain a user. Nothing's wrong with that, either.

***Users*** *Users* are the people who run FreeBSD systems. It's impossible to realisti- cally estimate the number of FreeBSD users. While organizations such as the BSDstats Project (*http://www.bsdstats.org/*) make an effort, these projects are opt-in. They measure only folks who have installed FreeBSD and then installed the software that adds their system to the count. Most users down- load the whole of FreeBSD for free and never register, upgrade, or email a mailing list. We have no idea how many FreeBSD users are in the world.

Since FreeBSD is by far the most popular open source BSD, that's not an inconsiderable number of machines. And since one FreeBSD server can handle hundreds of thousands of internet domains, a disproportionate number of sites use FreeBSD as their supporting operating system. This means that there are hundreds of thousands, if not millions, of FreeBSD system administrators out in the world today.

## Other BSDs

FreeBSD might be the most popular BSD, but it's not the only one. BSD 4.4-Lite2 spawned several different projects, each with its own focus and purpose. Those projects in turn had their own offspring, several of which thrive today.

***NetBSD*** NetBSD is similar to FreeBSD in many ways, and NetBSD and FreeBSD share developers and code. NetBSD's main goal is to provide a secure and reliable operating system that can be ported to any

hardware platform with minimal effort. As such, NetBSD runs on Vixens, PocketPC devices, and high-end SPARC and Alpha servers. I ran NetBSD on my HP Jornada hand- held computer.[2]

***OpenBSD*** OpenBSD branched off from NetBSD in 1996 with the goal of becoming the most secure BSD. OpenBSD was the first to support hardware-accelerated cryptography, and its developers are rightfully proud of the fact that their default installation was largely immune to remote exploits for several years. The OpenBSD team has contributed several valuable pieces of software to the world, including the LibreSSL TLS library and the OpenSSH suite used by almost everyone from Linux to Microsoft.

***DragonFly BSD*** DragonFly BSD forked from FreeBSD 4 in 2003. It developed in a dif- ferent direction than FreeBSD, with a new kernel messaging system.

2. If you're ever in a position where you need to prove that you are Alpha Geek amongst the pack, running Unix on a 1998 palmtop will almost certainly do it.

DragonFly BSD has very high performance and its HAMMER filesystem supports snapshots and fine-grained history. Check out *http://www.dragonfly bsd.org/* for more information.

***macOS*** Apple's macOS? That's right. Apple incorporates large chunks of FreeBSD into its macOS on an ongoing basis. If you're looking for a stable operating system with a friendly face and a powerful core,

macOS is unquestionably for you. While FreeBSD makes an excellent desktop for a computer profes- sional, I wouldn't put it in front of a random user. I would put macOS in front of that same random user without a second thought, however, and I'd even feel that I was doing the right thing. But macOS includes many things that aren't at all necessary for an internet server, and it runs only on Apple hardware, so I don't recommend it as an inexpensive general-purpose server.

***FreeBSD's Children*** Several projects have taken FreeBSD and built other projects or products on top of it. The award-winning FreeNAS transforms a commodity system into a network fileserver. The pfSense project transforms your system into a firewall with a nice web management interface. TrueOS gives FreeBSD a friendly face while supporting resource-intensive advanced features, like ZFS, while GhostBSD puts a friendly face on equipment with less comput- ing oomph. Other projects like this appear from time to time; while not all are successful, I'm sure by the time this book comes out, we'll have one or two more solid members of this group.

## Other Unixes

Several other operating systems derive from or emulate primordial Unix in one way or another. This list is by no means exhaustive, but I'll touch on the high points.

***Solaris*** The best-known Unix might be Oracle Solaris. Solaris runs on high-end hardware that supports dozens of processors and gobs of disk. (Yes, *gobs* is a technical term, meaning *more than you could possibly ever need, and I know very well that you need more disk than I think you need*.) Solaris, especially early versions of Solaris, had strong BSD roots. Many enterprise-level applications run on Solaris. Solaris runs mainly on the SPARC hardware platform man- ufactured by Sun, which allows Sun to support interesting features, such as hot-swappable memory and mainboards.

The Oracle Corporation acquired Solaris when they bought Sun Microsystems in 2009. Oracle ceased Solaris development in 2016. While there's still an extensive installed base of Solaris systems and you can still get Solaris from Oracle, as of today, Oracle Solaris has no future.

***illumos*** Several years before Oracle purchased Sun Microsystems, Sun open sourced the majority of Solaris and sponsored the OpenSolaris project to improve that codebase. OpenSolaris ran successfully until Oracle shut down source access and reclaimed all of the OpenSolaris resources.

The OpenSolaris code was still available, though. The OpenSolaris community forked OpenSolaris into illumos (*http://illumos.org/*). If you miss Solaris, you can still use a free, modern, Solaris-like operating system. FreeBSD includes two important features from OpenSolaris, the Zetabyte Filesystem (ZFS) and DTrace, a full-system tracing system.

***AIX*** Another Unix contender is IBM's entry, AIX. AIX's main claim to fame is its journaling filesystem, which records all disk transactions as they happen and allows for fast recovery from a crash. It was also IBM's standard Unix for many years, and anything backed by Big Blue shows up all over the place. AIX started life based on BSD, but AT&T has twiddled just about everything so that you won't find much BSD today.

***Linux*** Linux is a close cousin of Unix, written from the ground up. Linux is simi- lar to FreeBSD in many ways, though FreeBSD has a much longer heritage and is friendlier to commercial use than Linux. Linux includes a require- ment that any user who distributes Linux must make his or her changes available to the

end user, while BSD has no such restriction. Of course, a Linux fan would say, “FreeBSD is more vulnerable to commercial exploita- tion than Linux.” Linux developers believe in share-and-share-alike, while BSD developers offer a no-strings-attached gift to everyone. It all depends on what’s important to you.

Many new Unix users have a perception of conflict between the BSD and Linux camps. If you dig a little deeper, however, you’ll find that most of the developers of these operating systems communicate and cooperate in a friendly and open manner. It’s just a hard fringe of users and developers that generate friction, much like different soccer teams’ hooligans or differ- ent *Star Trek* series’ fans.[3]

***Other Unixes*** Many Unixes have come and gone, while others stagger on. Past contenders include Silicon Graphics’ IRIX, Hewlett-Packard’s HP/UX, Tru64 Unix, and the suicidal SCO Group’s UnixWare. Dig further and you’ll find older castoffs, including Apple’s A/UX and Microsoft’s Xenix. (Yes, Microsoft was a licensed Unix vendor, back in that age when dinosaurs watched the skies nervously and my dad hunted mammoth for all the tribal rituals.) Many

3. Original Trek. End of discussion. Fight me.

high-end applications are designed to run best on one particular flavor of Unix. All modern Unixes have learned lessons from these older operating systems, and today's Unixes and Unix-like operating systems are remarkably similar.

## FreeBSD's Strengths

After all this, what makes FreeBSD unique?

***Portability*** The FreeBSD Project's goal is to provide a freely redistributable, stable, and secure operating system that runs on the computer hardware that people are most likely to have access to. People have ported FreeBSD to a variety of less popular platforms as well.

The best supported FreeBSD platform is the common 64-bit hardware developed by AMD, used by almost everyone, and even copied by Intel. FreeBSD also fully supports the older 32-bit computers, such as 486s and all the flavors of Pentiums. This book uses 64-bit commodity hardware, or *amd64*, as a reference platform.

FreeBSD runs well on several other hardware architectures but is not completely supported yet. These include 32-bit ARM processors and PowerPC. While these other platforms are not afterthoughts, they don't receive the same level of attention that x86 and amd64 do. The 64-bit ARM platform is expected to become Tier 1 shortly after this book comes out, however.

You can also load FreeBSD on certain older architectures, such as 64-bit SPARC. These platforms were once well supported but are on their way out.

**Why unIx-LIke?**

One thing to note is that FreeBSD, Linux, and so on are called *Unix-like* instead of *Unix*. The term *Unix* is a trademark of The Open Group. For an operating system to receive the right to call itself Unix, the vendor must prove that the OS complies with the current version of the Single Unix Specification. While FreeBSD generally meets the standard, continuous testing and recertification cost money, which the FreeBSD Project doesn't have to spare. Certification as Unix also requires that someone sign a paper stating not only that he or she is responsible for FreeBSD's conformance to the Single Unix Specification but that he or she will fix any deviations from the standard that are found in the future. FreeBSD's development model makes this even more difficult—bugs are found and deviations are fixed, but there's nobody who can sign a piece of paper that guarantees 100 percent standards compliance.

***Power*** Since FreeBSD runs adequately on 486 processors, it runs extremely well on modern computers. It's rather nice to have an operating system that doesn't demand 8 cores and 12 gigs of RAM just to run the

user interface. As a result, you can actually dedicate your hardware to accomplishing real work rather than tasks you don't care about. If you choose to run a pretty graphical interface with all sorts of spinning gewgaws and fancy whistles, FreeBSD will support you, and it won't penalize you if you choose otherwise. FreeBSD will also support you on the latest *n* -CPU hardware.

***Simplified Software Management*** FreeBSD also simplifies software management through the packaging system and the Ports Collection. Traditionally, running software on a Unix- like system required a great deal of expertise. Packages and ports simplify this considerably by automating and documenting the install, uninstall, and configuration processes for thousands of software packages.

We discuss packages in Chapter 15 and ports in Chapter 16.

***Customizable Builds*** FreeBSD provides a painless upgrade procedure, but it also lets you pre- cisely customize the operating system for your hardware. Companies like Apple do exactly this, but they control both the hardware and the software; FreeBSD pulls off the same trick on commodity hardware.

***Advanced Filesystems*** A *filesystem* is how information is stored on the physical disk—it's what maps the file *My Resume* to a series of zeros and ones on a hard drive. FreeBSD includes two well-supported filesystems, UFS (Chapter 11) and ZFS (Chapter 12). UFS has been around for multiple decades and is highly damage-resistant. ZFS is younger but includes features such as network rep- lication and self-healing.

## Who Should Use FreeBSD?

While FreeBSD can be used as a powerful desktop or development machine, its history shows a strong bias toward network services: web, mail, file, and ancillary applications. FreeBSD is most famous for its strengths as an inter- net server, and it's an excellent choice as an underlying platform for any network service. If major firms such as Netflix count on FreeBSD to provide reliable service, it will work as well for you.

If you're thinking of running FreeBSD (or any Unix) on your desktop, you'll need to understand how your computer works. FreeBSD is not your best choice if you need point-and-click simplicity. If that's your goal, get

a Mac so you can use the power of Unix when you need it and not worry about it the rest of the time. If you want to learn FreeBSD, though, running it on your desktop is the best way—as we'll discuss later.

## Who Should Run Another BSD?

NetBSD and OpenBSD are FreeBSD's closest competitors. Unlike competi- tors in the commercial world, this competition is mostly friendly. FreeBSD, NetBSD, and OpenBSD freely share code and developers; some people even maintain the same subsystems in multiple operating systems.

If you want to use old or oddball hardware, NetBSD is a good choice for you. For several years, I ran NetBSD on an ancient SGI workstation that I used as a Domain Name System (DNS) and fileserver. It did the job well until the hardware finally released a cloud of smoke and stopped working. OpenBSD has implemented an impressive variety of security features. Some of the tools are eventually integrated into FreeBSD, but that takes months or years. Some of the tools can never be duplicated in FreeBSD, however. If you have real security concerns and can use a Unix-like system without the feature set FreeBSD provides, consider OpenBSD. Take a look at my book *Absolute OpenBSD* (No Starch Press, 2013) for an introduction. If you're just experimenting to see what's out there, any BSD is good!

## Who Should Run a Proprietary Operating System?

Operating systems such as macOS, Windows, AIX, and their ilk are still quite popular, despite the open source operating systems gnawing at their market share. High-end enterprises are pretty tightly shackled to commer- cial operating systems. While this is slowly changing, you're probably stuck with commercial operating systems in such environments. But slipping in an occasional FreeBSD machine to handle basic services, such as monitor- ing and department file serving, can make your life much easier at much lower cost. Companies like Dell/EMC/Isilon have built entire businesses using FreeBSD instead of commercial operating systems.

Of course, if the software you need runs only on a proprietary operat- ing system, your choice is pretty clear. Still, always ask a vendor whether a FreeBSD version is available; you might be pleasantly surprised.

## How to Read This Book

Many computer books are thick and heavy enough to stun an ox, if you have the strength to lift them high enough. Plus, they're either encyclopedic in scope or so painfully detailed that they're difficult to actually read. Do you really need to reference a screenshot when you're told to click OK or accept the license agreement? And when was the last time you actually sat down to read the encyclopedia?

*Absolute FreeBSD* is a little different. It's designed to be read once, from front to back. You can skip around if you want to, but each chapter builds on what comes before it. While this isn't a small book, it's smaller than many popular computer books. After you've read it once, it makes a decent reference.

If you're a frequent buyer of computer books, please feel free to insert all that usual crud about "read a chapter at a time for best learning" and so on. I'm not going to coddle you—if you picked up this book, you either have two brain cells to rub together or you're visiting someone who does. (If it's the latter, hopefully your host is smart enough to take this book away from you before you learn enough to become dangerous.)

## What Must You Know?

This book is aimed at the new Unix administrator. Three decades ago, the average Unix administrator had kernel programming experience and was working on their master's degree in computer science. Even a decade ago, they were already a skilled Unix user with real programming skills and most of a bachelor's degree in comp sci. Today, Unix-like operating systems are freely available, computers are cheaper than food, and even 12-year-old children can run Unix, read the source code, and learn enough to intimi- date older

folks. As such, I don't expect you to know a huge amount about Unix before firing it up.

To use this book to its full potential, you need to have familiarity with some basic tasks, such as how to change directories, list files in a directory, and log in with a username and password. If you're not familiar with basic commands and the Unix shell, I recommend you begin with a book like *UNIX System Administration Handbook* by Evi Nemeth and friends (Prentice Hall PTR, 2017). To make things easier on newer system administrators, I include the exact commands needed to produce the desired results. If you learn best by example, you should have everything you need right here.

You'll also need to know something about computer hardware—not a huge amount, mind you, but something. It helps to know how to recog- nize a SATA cable. Your need for this knowledge depends on the hardware you're using, but if you're interested enough to pick up this book and read this far, you probably know enough.

## For the New System Administrator

If you're new to Unix, the best way to learn is to eat your own dog food. No, I'm not suggesting that you dine with Rover. If you ran a dog food company, you'd want to make a product that your own dog eats happily. If your dog turns his nose up at your latest recipe, you have a problem. The point here is that if you work with a tool or create something, you should actually use it. The same thing applies to any Unix-like operating system, including FreeBSD.

***Desktop FreeBSD*** If you're serious about learning FreeBSD, I suggest wiping out the operat- ing system on your main computer and running FreeBSD instead. No, not a desktop-oriented FreeBSD derivative like TrueOS or GhostBSD: run raw FreeBSD. Yes, I know, now that dog food doesn't sound so bad. But learn- ing an operating system is like learning a language; total immersion is the quickest and most powerful way to learn. That's what I did, and today I can make a Unix-like system do anything I want. I've written

entire books on a FreeBSD laptop, using the open source text editor XEmacs and the LibreOffice.org business suite. I've also used FreeBSD to watch movies, rip and listen to MP3s, balance my bank accounts, process my email, and surf the web. The desktop in my lab has a dozen animated BSD daemons run- ning around the window manager, and I occasionally take a break to zap them with my mouse. If this doesn't count as a Stupid Desktop Trick, I don't know what does.[4]

Many Unix system administrators these days come from a Windows background. They're beavering away in their little world when their man- ager swoops by and says, "You can handle one more system, can't you? Glad to hear it! It's a Unix box, by the way," and then vanishes into the manage- rial ether. Once the new Unix administrator decides not to quit her job and start a fresh and exciting career as a whale necropsy technician, she tentatively pokes at the system. She learns that `ls` is like `dir` and that `cd` is the same on both platforms. She can learn the commands by rote, reading, and experience. What she can't learn, coming from this background, is how a Unix machine *thinks*. Unix will not adjust to you; you must adjust to it. Windows and macOS require similar adjustments but hide them behind a glittering facade. With that in mind, let's spend a little time learning how to think about Unix.

### *How to Think About Unix*

These days, most Unix systems come with pretty GUIs out of the box, but they're just eye candy. No matter how graphically delicious the desktop looks, the real work happens on the command line. The Unix command line is actually one of Unix's strengths, and it's responsible for its unparalleled flexibility.

Unix's underlying philosophy is *many small tools, each of which does a single job well*. My mail server's local programs directory (*/usr/local/bin*) has 262 programs in it. I installed every one of them, either directly or indirectly. Most are small, simple programs that do only one task. This array of small tools makes Unix extremely flexible and adaptable. Many commercial software packages try to do everything; they wind up with all

4. In the first edition of this book, I neglected to mention exactly how to do a similar Stupid Desktop Trick, which generated more questioning email than any other topic in the whole book. In the second edition, I swore I wouldn't make that same mistake again but neglected to mention which software package provides the run-around daemons. They say the third time's the charm.

sorts of capabilities but only mediocre performance in their core functions. Remember, at one time you needed to be a programmer to use a Unix sys- tem, let alone run one. Programmers don't mind building their own tools. The Unix concept of pipes encouraged this.

#### Pipes

People used to GUI environments, such as Windows and macOS, are prob- ably unfamiliar with how Unix handles output and input. They're used to clicking something and seeing either an OK message, an error, nothing, or (all too often) a pretty blue screen with nifty high-tech letters explaining in the language called *Geek* why the system crashed. Unix does things a little differently.

Unix programs have three channels of communication, or *pipes*: stan- dard input, standard output, and standard error. Once you understand how each of these pipes works, you're a good way along to understanding the whole system.

*Standard input* is the source of information. When you're at the console typing a command, the standard input is the data coming from the key- board. If a program is listening to the network, the standard input is the network. Many programs can rearrange standard input to accept data from the network, a file, another program, the keyboard, or any other source.

The *standard output* is where the program's output is displayed. This is frequently the console (screen). Network programs usually return their out- put to the network. Programs might send their output to a file, to another

program, over the network, or anywhere else available to the computer.

Finally, *standard error* is where the program sends its error messages. Frequently, console programs return their errors to the console; others log errors in a file. If you set up a program incorrectly, it just might dis- card all error information.

These three pipes can be arbitrarily arranged, a concept that's perhaps the biggest hurdle for new Unix users and administrators. For example, if you don't like the error messages appearing on the terminal, you can redi- rect them to a file. If you don't want to repeatedly type a lot of information into a command, you can put the information into a file (so you can reuse it) and dump the file into the command's standard input. Or, better still, you can run a command to generate that information and put it in a file, or just pipe (send) the output of the first command directly to the second, without even bothering with a file.

**Small Programs, Pipes, and the Command Line** Taken to their logical extreme, these input/output pipes and the variety of tools seem overwhelming. When I saw a sysadmin type something like the following during my initial Unix training session, I gave serious consider- ation to changing careers.

```
$ tail -f /var/log/messages | grep -v popper | grep -v named &
```

Lines of incomprehensible text began spilling across the screen, and they kept coming. And worse still, my mentor kept typing as gibberish poured out! If you're from a point-and-click computing environment, a long string of commands like this is definitely intimidating. What do all those funky words mean? And an ampersand? You want me to learn *what*? Think of learning to use the command line as learning a language. When learning a language, we start with simple words. As we increase our vocabulary, we also learn how to

string the words together. We learn that placing words in a certain order makes sense, and that a different order makes no sense at all. You didn't speak that well at three years old—give yourself some slack and you'll get there.

Small, simple programs and pipes provide almost unlimited flexibil- ity. Have you ever wished you could use a function from one program in another program? By using a variety of smaller programs and arranging the inputs and outputs as you like, you can make a Unix system behave in any manner that amuses you. Eventually, you'll feel positively hogtied if you can't just run a command's output through `| sort -rnk 6 | less`.[5]

**Everything Is a File** You can't be around Unix for very long before hearing that everything is a file. Programs, account information, and system configuration are all stored in files. Unix has no Windows-style registry; if you back up the files, you have the whole system.

What's more, the system identifies system hardware as files! Your CD-ROM drive is a file, */dev/cd0*. Serial ports appear as files like */dev/cuaa0*. Even virtual devices, such as packet sniffers and partitions on hard drives, are files.

When you have a problem, keep this fact in mind. Everything is a file, or is in a file, somewhere on your system. All you have to do is find it!

## Notes on the Third Edition

*Absolute BSD* (No Starch Press, 2002) was my first technology book and was written when the various BSD operating systems had more in common than they wanted to admit. The second edition, *Absolute FreeBSD* (No Starch Press, 2007), came out after the BSDs had diverged, and detailed FreeBSD's advances in the previous five years. With another decade of growth, FreeBSD has evolved to compete with the best commercial operating systems. You'll find multiple top-tier filesystems. Disk management has changed to accom- modate new partitioning methods. Virtualization is now a thing, and FreeBSD supports it as either a client or a host.

5. This ugly thing takes the output of the last command, sorts it in reverse order by the contents of the sixth column, and presents it one screen at a time. If you have hundreds of lines of output, and you want to know which entries have the highest values in the sixth column, this is how you do it. Or, if you have lots of time, you can dump the output to a spreadsheet and fiddle with equally obscure commands for a much longer time.

This growth has driven changes in this book. We won't discuss configuring mail, DNS, or web servers. You have more software choices for these tasks than ever before. Entire books have been written about those choices and how to use them. I've written some of those books. Those topics have been dropped to make space for FreeBSD-specific material, like ZFS and jails.

Some of these new features are hugely complex. Complete coverage of ZFS would fill entire books—I know, because I've written those books, too. FreeBSD supports a whole bunch of special-purpose filesystems, each incredibly useful to the folks who need them and totally irrelevant to those who don't. Rather than write a monster tome that nobody would actually read, I've elected to cover the material that every FreeBSD sysadmin *must* know. If you're interested in deeper coverage of a particular topic, it's available.

Some subsystems are undergoing radical revision. I could wait to write this book until every FreeBSD subsystem has a stable interface, but then it would come out about . . . never. As I write this, the bhyve developers are actively rototilling their entire configuration system. Given the choice between glossing over a topic and providing flat-out wrong material, I've chosen to skip detail on bhyve. I hope to be able to delete this paragraph before this book goes to press.

I've ruthlessly excised obsolete information from this edition. For example, modern disk drives don't generally have to worry about write caching. If you discover that a piece of advice you remember using doesn't appear in this book, please check FreeBSD's information resources to see whether that advice is still applicable.

## Contents of This Book

*Absolute FreeBSD*, 3rd Edition contains the following chapters.

**Chapter 1: Getting More Help**

This chapter discusses the information resources the FreeBSD Project and its devotees provide for users. No one book can cover everything, but knowing how to use the many FreeBSD resources on the internet helps fill any gaps you find here.

**Chapter 2: Before You Install**

Getting FreeBSD installed isn't that hard. Make poor choices during the install, though, and you'll have a system that isn't suited for your needs. The best way to avoid reinstalling is to think about your require- ments and make all the decisions beforehand so that the actual install doesn't require any thought.

**Chapter 3: Installing**

This chapter gives you an overview of installing FreeBSD using differ- ent partitioning schemes and filesystems.

**Chapter 4: Start Me Up! The Boot Process**

This chapter teaches you about the FreeBSD boot process and how to make your system start, stop, and reboot in different configurations.

**Chapter 5: Read This Before You Break Something Else! (Backup and Recovery)**

Here we discuss how to back up your data on both a system-wide and a file-by-file level, and how to make your changes so that they can be eas- ily undone.

**Chapter 6: Kernel Games**

This chapter describes configuring the FreeBSD kernel. Unlike some other operating systems, you're expected to tune FreeBSD's kernel to best suit your purposes. This gives you tremendous flexibility and lets you optimize your hardware's potential.

**Chapter 7: The Network**

Here we discuss the TCP/IP protocol that underlies the modern inter- net, both version 4 and version 6.

**Chapter 8: Configuring the Network**

FreeBSD doesn't only shuffle packets crazy fast, but it also supports vir- tual LANs, link aggregation, and more. We'll configure all of that here.

**Chapter 9: Securing Your System**

This chapter teaches you how to make your computer resist attackers and intruders.

**Chapter 10: Disks, Partitioning, and GEOM**

This chapter covers some of the details of working with hard drives in FreeBSD. Working with modern hardware means understanding mul- tiple partitioning schemes, disk alignment, and FreeBSD's disk manage- ment infrastructure.

**Chapter 11: The Unix File System**

UFS has been FreeBSD's standard filesystem for decades, and the con- cepts of UFS pervade the whole operating system. Whether you intend to use UFS or not, you must understand its essentials.

**Chapter 12: The Z File System**

ZFS is a newer filesystem very popular on larger systems. If you're man- aging large amounts of data, you'll want ZFS.

**Chapter 13: Foreign Filesystems**

Every sysadmin needs to mount disks over the network or use ISOs without burning them to CD. This chapter takes you through those duties, as well as introducing FreeBSD-specific filesystems like devfs.

**Chapter 14: Exploring /etc**

This chapter describes the many configuration files in FreeBSD and how they operate.

**Chapter 15: Making Your System Useful**

Here I describe the packages system that FreeBSD uses to manage add- on software.

**Chapter 16: Customizing Software with Ports**

Sometimes the prebuilt packages won't cover everything you need. You can leverage FreeBSD's package-building system to create your own software packages, tuned to meet your exact needs.

**Chapter 17: Advanced Software Management**

This chapter discusses some of the finer points of running software on FreeBSD systems.

**Chapter 18: Upgrading FreeBSD**

This chapter teaches you how to use FreeBSD's upgrade process. The upgrade system is among the most remarkable and smooth of any oper- ating system.

**Chapter 19: Advanced Security Features**

Here we discuss some of the more interesting security features found in FreeBSD.

**Chapter 20: Small System Services**

Here we discuss some of the small programs you'll need to manage in order to use FreeBSD properly.

**Chapter 21: System Performance and Monitoring**

This chapter covers some of FreeBSD's performance-testing and trouble- shooting tools and shows you how to interpret the results. We also discuss logging and FreeBSD's SNMP implementation.

**Chapter 22: Jails**

FreeBSD has a process-isolation subsystem, much like Linux and Solaris containers, called *jails*. We'll cover the jail system and how you can leverage it for system security.

**Chapter 23: The Fringe of FreeBSD**

This chapter teaches you some of the more interesting tricks you can do with FreeBSD, such as running systems without disks and with tiny disks, as well as cloud-friendly features, like libxo.

**Chapter 24: Problem Reports and Panics**

This chapter teaches you how to deal with those rare occasions when a FreeBSD system fails, how to debug problems, and how to create a use- ful problem report.

You'll also find an annotated bibliography, an afterword, and a really spiffy professionally prepared index.

Okay, enough introductory stuff. Onward!

# 1

# GettinG More Help

As thick as this book is, it still can't possi- bly cover everything you must know about FreeBSD. After all, Unix has been kicking around for close to 50 years, BSD is pushing 40, and FreeBSD is old enough to have its doctorate. Even if you memorize this book, it won't cover every situation you might encounter.

The FreeBSD Project supports a huge variety of infor- mation resources, including numerous mailing lists and the FreeBSD web- site, not to mention the official manual and Handbook. Its users maintain even more documentation on even more sites. The flood of information can be overwhelming in itself, and it can make you want to just email the world and beg for help. But before you send a question to a mailing list or forum, confirm that the information you need isn't already available.

## Why Not Beg for Help?

FreeBSD provides two popular resources for assistance: mailing lists and forums. Many participants on both are very knowledgeable and can answer questions very quickly. But when you send a question to these community sup- port resources, you're asking tens of thousands of people all over the world to take a moment to read your message. You're also asking that one or more of them take the time to help you instead of watching a favorite movie, enjoy- ing dinner with their families, or catching up on sleep. Problems arise when these experts answer the same question 10, 50, or even hundreds of times. They become grumpy. Some get downright tetchy.

What makes matters worse is that many of these same people have spent a great deal of time and effort making the

answers to most of these ques- tions available elsewhere. If you make it clear that you've already searched the resources and your answer really doesn't appear therein, you'll probably receive a polite, helpful answer. If you ask a question that's already been asked several hundred times, however, the expert on that subject just might snap and go ballistic on you. Do your homework, and chances are you'll get an answer more quickly than a fresh call for assistance could provide.

***The FreeBSD Attitude*** "Homework? What do you mean? Am I back in school? What do you want, burnt offerings on bended knee?" Yes, you are in school. The information technology business is nothing but lifelong, self-guided learning. Get used to it or get out. Burnt offerings, on the other hand, are difficult to transmit via email and aren't quite so useful today.

Most commercial software conceals its inner workings. The only access you have to them is through the options presented by the vendor. Even if you want to learn how something works, you probably can't. When something breaks, you have no choice but to call the vendor and grovel for help. Worse, the people paid to help you frequently know little more than you do.

If you've never worked with open source software vendors, FreeBSD's support mechanism might surprise you. There is no toll-free number to call and no vendor to escalate within. No, you may not speak to a manager and for a good reason: you are the manager. Congratulations on your promotion!

***Support Options*** That being said, you're not entirely on your own. The FreeBSD community includes numerous developers, contributors, and users who care very deeply about FreeBSD's quality, and they're happy to work with users who are will- ing to do their share of the labor. FreeBSD provides everything you need: complete access to the source code used to create the system, the tools needed to turn that source code into programs, and the same debuggers used by the developers. Nothing is hidden; you can see the innards, warts and all. You can view FreeBSD's development history since the beginning,



including every change ever made and the reason for it. These tools might be beyond your abilities, but that's not the Project's problem. Various com- munity members are even happy to provide guidance as you develop your own skills so you can use those tools yourself. You'll have lots of help fulfill- ing your responsibilities.

As a grossly overgeneralized rule, people help those like themselves. If you want to use FreeBSD, you must make the jump from eating what the vendor gives you to learning how to cook. Every member of the FreeBSD user community learned how to use it, and they welcome interested new users with open arms. If you just want to know what to type without really understand- ing what's going on behind the scenes, you'll be better off reading the docu- mentation; the general FreeBSD support community simply isn't motivated to help those who won't help themselves or who can't follow instructions.

If you want to use FreeBSD but have neither the time nor the inclina- tion to learn more, invest in a commercial support contract. It might not be able to put you in touch with FreeBSD's owner, but at least you'll have some- one to yell at. You'll find several commercial support providers listed on the FreeBSD website.

It's also important to remember that the FreeBSD Project maintains only FreeBSD. If you're having trouble with some other piece of software, a FreeBSD mailing list is not the place to ask for help. FreeBSD developers are generally proficient in a variety of software, but that doesn't mean they want to help you, say, configure KDE.

The first part of your homework, then, is to learn about the resources available beyond this book. These include the integrated manual, the FreeBSD website, the mailing list archives, and other websites.

## Man Pages

*Man pages* (short for *manual pages*) are the primordial way of presenting Unix documentation. While man pages have a reputation for being obtuse, difficult, or even incomprehensible, they're actually quite friendly—for particular users. When man pages were first created, the average system administrator was a C programmer and, as a result, the pages were written by programmers, for programmers. If you can think like a programmer, man pages are perfect for you. I've tried thinking like a programmer, but I achieved real success only after remaining awake for two days straight. (Lots of caffeine and a high fever help.)

Over the last several years, the skill level required for system adminis- tration has dropped; no longer must you be a programmer. Similarly, man pages have become more and more readable. Man pages are not tutorials, however; they explain the behavior of one particular program, not how to achieve a desired effect. While they're neither friendly nor comforting, they should be your first line of defense. If you send a question to a mailing list without checking the manual, you're likely to get a terse *man whatever* in response.

***Manual Sections*** The FreeBSD manual is divided into nine sections. Roughly speaking, the sections are:

1. General user commands 2. System calls and error numbers 3. C programming libraries 4. Devices and device drivers 5. File

formats 6. Game instructions 7. Miscellaneous information 8.
System maintenance commands 9. Kernel interfaces

Each man page starts with the name of the command it documents followed by its section number in parentheses, like this: reboot(8). When you see something in this format in other documents, it's telling you to read that man page in that section of the manual. Almost every topic has a man page. For example, to see the man page for the editor vi, type this command:

```
$ man vi
```

In response, you should see the following:

```
VI(1) FreeBSD General Commands Manual VI(1)

NAME
     ex, vi, view - text editors

SYNOPSIS
     ex [-FRrSsv] [-c cmd] [-t tag] [-w size] [file ...] vi [-eFRrS] [-c cmd] [-t tag] [-w size] [file ...]
     view [-eFrS] [-c cmd] [-t tag] [-w size] [file ...]

DESCRIPTION
vi is a screen-oriented text editor. ex is a line-oriented text editor. ex and vi are different interfaces to the same program, and it is possible to switch back and forth during an edit session. view is the equivalent of using the -R (read-only) option of vi. :
```

The page starts with the title of the man page (vi) and the section num- ber (1), and then it gives the name of the page. This particular page has three names: ex, vi, and view. Typing man ex or man view would take you to this same page.

***Navigating Man Pages*** Once you're in a man page, pressing the spacebar or the PGDN key takes you forward one full screen. If you don't want to go that far, pressing ENTER or the down arrow scrolls down one line. Typing b or pressing the PGUP key takes you back one screen. To search within a man page, type / followed by the word you're searching for. You'll jump down to the first appearance of the word, which will be highlighted. Typing n subsequently takes you to the next occurrence of the word.

This assumes that you're using the default BSD pager, more(1). If you're using a different pager, use that pager's syntax. Of course, if you know so much about Unix that you've already set your preferred default pager, you've probably skipped this part of the book.

***Finding Man Pages*** New users often say that they'd be happy to read the man pages if they could find the right one. You can perform basic keyword searches on the man pages with apropos(1) and whatis(1). To search any man page name or description that includes the word you specify, use apropos(1). To match only whole words, use whatis(1). For example, if you're interested in the vi command, you might try the following:

```
$ apropos vi unvis(1) - revert a visual representation of data back to original form vidcontrol(1) - system console
control and configuration utility vis(1) - display non-printable characters in a visual format madvise, posix_madvise(2) -
give advice about use of memory posix_fadvise(2) - give advice about use of file data --snip--
```

This continues for a total of 581 entries, which is probably far more than you want to look at. Most of these have

nothing to do with vi(1), how- ever; the letters *vi* just appear in the name or description. *Device driver* is a fairly common term in the manual, so that's not surprising. On the other hand, whatis(1) gives more useful results in this case.

$ **whatis vi** vi, ex, view, nex, nvi, nview(1) - text editors $

We get only one result, clearly with relevance to vi(1). On other searches, apropos(1) gives better results than whatis(1). Experiment with both and you'll quickly learn how they fit your style.

The man -k command emulates apropos(1), while man -f emulates whatis(1).

***Section Numbers and Man*** You might find cases where a single command appears in multiple parts of the manual. For example, every man section has an introductory man page that explains the contents of the section. To specify a section to search for a man page, give the number immediately after the man command.

$ **man 3 intro**

This pulls up the introduction to section 3 of the manual. I recommend you read the intro pages to each section of the manual, if only to help you understand the breadth and depth of information available.

***Man Page Contents*** Man pages are divided into sections. While the author can put just about any heading he or she likes into a man page, several are standard. See mdoc(7) for a partial list of these headings as well as other man page standards:

• NAME gives the name(s) of a program or utility. Some programs have multiple names—for example, the vi(1) text editor is also available as ex(1) and view(1).

• SYNOPSIS lists the possible command line options and their argu- ments, or how a library call is accessed. If I'm already familiar with a program but just can't remember the option I'm looking for, I find that this header is sufficient to remind me of what I need.

• DESCRIPTION contains a brief description of the program, library, or feature. The contents of this section vary widely depending on the topic, as programs, files, and libraries all have very different documen- tation requirements.

• OPTIONS gives a program's command line options and their effects.

• BUGS describes known problems with the code and can frequently save a lot of headaches. How many times have you wrestled with a computer problem only to learn that it doesn't work the way you'd expect under those circumstances? The goal of the BUGS section is to save you time by describing known errors and other weirdnesses.[1]

• EXAMPLES gives sample uses of the program. Many programs are very complicated, and a couple samples of how they're used clarify more than any list of options possibly can.

• HISTORY shows when the command or code was added to the system and, if it is not original to FreeBSD, where it was drawn from.

• SEE ALSO is traditionally the last section of a man page. Remember that Unix is like a language and the system is an interrelated whole. Like duct tape, the SEE ALSO links hold everything together.

1. It's called *honesty*. IT professionals may find this term unfamiliar, but a dictionary can help.

If you don't have access to the manual pages at the moment, many web- sites offer them. Among them is the main FreeBSD website.

## FreeBSD.org

The FreeBSD website (*http://www.freebsd.org/*) contains a variety of informa- tion about general FreeBSD administration, installation, and management. The most useful portions are the Handbook, the FAQ, and the mailing list archives, but you'll also find a wide number of articles on dozens of topics. In addition to documents about FreeBSD, the website contains a great deal of information about the FreeBSD Project's internal management and the status of various parts of the Project.

***Web Documents*** The FreeBSD documentation is divided into articles and books. The differ- ence between the two is highly arbitrary: as a rule, books are longer than articles and cover broader topics, while articles are short and focus on a single topic. The two books that should most interest new users are the Handbook and the Frequently Asked Questions (FAQ).

The Handbook is the FreeBSD Project's tutorial-style manual. It is con- tinuously updated, describes how to perform basic system tasks, and is an excellent reference when you're first starting a project. I deliberately chose not to include some topics in this book because they have adequate cover- age in the Handbook.

The FAQ is designed to provide quick answers to the questions most fre- quently asked on the FreeBSD mailing lists. Some of the answers aren't suit- able for inclusion in the Handbook, while others just point to the proper Handbook chapter or article.

Several other books cover a variety of topics, such as The FreeBSD Developers' Handbook, The Porter's Handbook, and The FreeBSD Architecture Handbook.

Of the 50 or so articles available, some are kept only for historical rea- sons (such as the original BSD 4.4 documentation), while others discuss the subtleties of specific parts of the system, such as serial ports or building filtering bridges.

On the other hand, the official documentation is also pruned. The Handbook and FAQ cover the current FreeBSD releases, and the documen- tation team mercilessly prunes obsolete information. If you want to know exactly what works with current FreeBSD, go to the Handbook.

These documents are very formal, and they require preparation. As such, they always lag a bit behind the real world. When a new feature is first rolled out, the appropriate Handbook entry might not appear for weeks or months. If the web documentation seems out of date, your best resource for up-to-the-minute answers is the mailing list archive.

***The Mailing List Archives*** Unless you're really on the bleeding edge, someone has probably struggled with your problem before and posted a question about it to the mailing lists. After all, the archives go back to 1994 and contain millions of mes- sages. The only problem is that there are millions of pieces of email, any one of which might contain the answer you seek. While the FreeBSD.org website has its own search engine, you can also use any other search engine that indexes *https://lists.FreeBSD.org/*.

When reviewing the mailing list archives, be sure to check the date. The mailing list is forever. A discussion of hardware problems from 1995 might help you feel that you're part of a long history of sysadmins that have struggled with cruddy mainboards,[2] but it probably won't help you solve the issue with your brand new server. These ancient messages are basi- cally undead documentation, rising from the grave to give you false hope. They're part of the Project's history, though, and won't be purged.

***The Forums*** Like many other open source projects, FreeBSD has an online forum, *https://forums.FreeBSD.org/*. A forum is much like a mailing list designed for the web, except that quite a few of us old geezers don't much care for them. You can find many good discussions and instructions on the forums, how- ever, and they're a valuable information source.

Many people have also posted lengthy tutorials on the forums. Forum- based tutorials should properly go in the Handbook or an official article, but nobody's done the work to move them over yet. Read the discussion about such tutorials before following them; people will often point out errors or exceptions, or comment that the whole tutorial is obsolete with a newer version of FreeBSD. If you want to get involved in FreeBSD, convert- ing these tutorials into official documentation would be a great place to start. The forums have less of a problem with truly old information, but only because they became official in 2009. When the forums reach a quarter- century old, they'll have the same amount of undead documents. By then, though, an even more whiz-bang discussion system will have come along— or maybe, just maybe, we'll have a better way of indexing and retrieving use- ful information from online discussions.

## Other Websites

FreeBSD's users have built a plethora of websites that you might check for answers, help, education, products, and general hobnobbing. Almost every aggregation site such as *lobste.rs* and Reddit has a FreeBSD section, where you can get links to new posts and articles. Following those links takes you to a whole world of blogs. Also, many hosting companies include extensive

2. Computer hardware has gotten faster and smaller, but not particularly better.

FreeBSD tutorials. While these are meant for the company's customers, they're most often perfectly useful for everyone.

One of the most popular FreeBSD sites is FreshPorts, *https://www.Fresh Ports.org/*. FreshPorts tracks changes to FreeBSD. Originally, it tracked changes to add-on software available via the Ports system (which I'll dis- cuss in Chapter 16), but it quickly expanded to cover changes to the base system, the documentation, the website, and more. If you're looking to see how FreeBSD has changed, start with FreshPorts.

The FreeBSD Journal (*https://www.freebsdfoundation.org/journal/*) is a project of the FreeBSD Foundation. It's a commercial project, but your sub- scription fees go directly to the Foundation. Journal articles are reviewed by some of the most experienced FreeBSD developers and users, so the articles can be considered authoritative. Though as an editorial board member, I'm biased.

The FreeBSD Foundation (*https://www.freebsdfoundation.org/*) supports FreeBSD development, and I'd encourage everyone to throw a few bucks their way. I find pages like the project list useful. This lists all of the devel- opment projects that the FreeBSD Foundation has financially supported and their current state. Looking through it when writing this paragraph, I learned that FreeBSD has added wireless mesh support and multipath TCP. I don't need either of these right now, but who knows what will happen next week?

Look around, and you'll find your own favorites.

## Using FreeBSD Problem-Solving Resources

Okay, let's investigate a common question with FreeBSD resources. People have asked about FreeBSD's cryptographic support for decades, so let's fig- ure out some definitive answers about what cryptographic functions it does and does not support.

Cryptography is a complicated topic, and searching for information on it is complicated by the different ways it's referred to. It might show up as "cryptography," "cryptographic," the informal "crypto," or related words, like "encrypt." We'll try any and all of these.

***Checking the Handbook and FAQ*** Skimming the Handbook's table of contents brings up entries for "Encrypting Disk Partitions" and "Encrypting Swap," which certainly seem relevant. The FAQ points to these topics as well. That's a start. Those entries will guide you to appropriate man pages, which will lead you to more man pages.

***Checking the Man Pages*** Let's query the man pages for cryptography, using both apropos and whatis.

$ **apropos cryptography** krb5_allow_weak_crypto, krb5_cksumtype_to_enctype... crypto, cryptodev(4) - user-mode access to hardware-accelerated cryptography

This is only two entries, but the first entry is extremely long. Anything that starts with *krb5* is related to Kerberos authentication, which is a critical feature for large networks and does involve cryptography, so that's relevant. The second entry is kind of interesting, though: there's a man page for crypto(4) and cryptodev(4). (Both words point to the same man page in section 4, so the number only appears in one search result.) Let's look at the man page.

$ **man crypto** crypto(3) OpenSSL crypto(3)

NAME

crypto - OpenSSL cryptographic library

SYNOPSIS DESCRIPTION

The OpenSSL crypto library implements a wide range of cryptographic algorithms used in various Internet standards. The services provided by this library are used by the OpenSSL implementations of SSL, TLS and S/MIME, and they have also been used to implement SSH, OpenPGP, and other cryptographic standards.

OVERVIEW

libcrypto consists of a number of sub-libraries that implement the individual algorithms. --*snip*--

Hang on—OpenSSL is not a cryptographic device. Something obvi- ously isn't right. Look closely at this man page; it's from section 3 of the manual, the C Libraries section. You need to search the manual for other entries containing *crypto*. Let's try the more specific whatis(1) search.

$ **whatis crypto** crypto, cryptodev(4) - user-mode access to hardware-accelerated cryptography crypto(7) - OpenCrypto algorithms crypto, crypto_dispatch, crypto_done, crypto_freereq, crypto_freesession, crypto_get_driverid, crypto_getreq, crypto_kdispatch, crypto_kdone, crypto_kregister, crypto_newsession, crypto_register, crypto_unblock, crypto_unregister, crypto_unregister_all, crypto_find_driver(9) - API for cryptographic services in the kernel

Bingo! We have three crypto man pages: one in section 4, section 7, and section 9. This gives us information about the interface for programs accessing hardware cryptographic features, a list of supported algorithms, and a description of the kernel's cryptographic services. Reading these will give you a good grounding in FreeBSD's cryptography support. The SEE ALSO links in each will steer you to more information. You can now fill your brain with crypto.

***Mailing Lists Archives and Forums*** While the mailing lists and forums are different platforms, you search them both in similar ways. You could use the FreeBSD website search engine to search the mailing list archives, but I prefer either Google or DuckDuckGo. A search for *crypto site:lists.FreeBSD.org* spits out a whole bunch of results, as does *crypto site:forums.FreeBSD.org*.

The problem with using these sorts of discussions for general orienta- tion on a topic is that folks plunge into nitty-gritty details. You won't get an overview of cryptography, but you'll find details on *this* algorithm used with *that* hardware acceleration on *that* version of FreeBSD. You can get a very detailed answer if you craft a very detailed search.

***Using Your Answer*** Any answer you get for a question will make certain assumptions. If you're talking about cryptography, the discussion assumes you know why crypto is important, how plaintext differs from ciphertext, and what keys are. This is fairly typical of the level of expertise required for basic problems. If you get an answer that is beyond your comprehension, you need to do the research to understand it. While an experienced developer or system administrator is probably not going to be interested in explaining public key encryption, he or she might be willing to point you to a web page that explains them if you ask nicely. Always remember that people have been asking that question in relation to FreeBSD since 1994 and in relation to Unix for close to half a century.

## Asking for Help

When you finally decide to ask for help, do so in a way that allows people to actually provide the assistance you need. No matter whether you prefer email or a forum, you must include all the information you have at your disposal. There's a lot of suggested information to include, and you might think you can skip some or all of it. If you slack off and fail to provide all the necessary information, though, one of the following things will happen:

- Your question will be ignored.
- You'll receive a barrage of email asking you to gather this information.

On the other hand, if you actually want help solving your problem, include the following pieces of information in your message:

- A complete problem description. A message like *How do I make my cable modem work?* only generates a multitude of questions: What do you want your modem to do? What kind of modem is it? What are the symptoms? What happens when you try to use it? How are you trying to use it?

• The output of uname -a. This gives the operating system version and platform.

• Any error output. Be as complete as possible, and include any messages from the console or from your logs, especially */var/log/messages* and any application-specific logs. Messages about hardware problems should include a copy of */var/run/dmesg.boot*.

It's much better to start with a message like "My cable modem won't connect to my ISP. The modem is a BastardCorp v.90 model BOFH667. My OS is version 12.2 on a quad-core Opteron. Here's the contents of */var/log/ messages* and */var/run/dmesg.boot* from when I try to connect. When I manu- ally run dhclient, I get these messages." You'll skip a whole round of discus- sions with a message like this, and you'll get better results more quickly.

***Composing Your Message*** First, *be polite*. People often say things online that they wouldn't dream of saying to someone's face. These lists are staffed by volunteers who are answering your message out of sheer kindness. Before you click that Send or Submit button, ask yourself, *Would I be late for my dream date to answer this message?* The fierce attitude that is occasionally necessary when working with corporate telephone-based support only makes these knowledgeable people delete your emails unread or flat-out block your account on the forum. Their world doesn't have to include surly jerks. Screaming until someone helps you is a valuable skill when dealing with commercial soft- ware support, but it will actively hurt your ability to get support from any open source project.

No matter whether you choose the forums or email, stay on topic. If you're having a problem with *X.org*, check the *X.org* website. If your window manager isn't working, ask the people responsible for the window manager. Asking the FreeBSD folks to help you with your Java Application Server con- figuration is like complaining to industrial machinery salespeople about your fast-food lunch. They might have an extra ketchup packet, but it's not really their problem. On the other hand, if you want your FreeBSD system to no longer start the mail system at boot time, that's a FreeBSD issue.[3]

**Sending Email** FreeBSD developers tend to use mailing lists, not the forums. This means that the mailing lists can get you attention from people who know more about the system, but it also means that you need to follow the etiquette for that environment.

Send plaintext email, not HTML. Many FreeBSD developers read their email with a text-only email program, such as Mutt. Such programs are very powerful tools for handling large amounts of email, but they do not display HTML messages without contortions. To see for yourself what this is like,

3. And it's one that's in the Handbook, the FAQ, the mailing list archives, *and* the forums.

install */usr/ports/mail/mutt* and read some HTML email with it. If you're using a graphic mail client, such as Microsoft Outlook, either send your email in plaintext or make sure that your messages include both a plaintext and an HTML version. All mail clients *can* do this; it's just a question of dis- covering where your GUI hides the buttons. What's more, be sure to wrap your text at 72 characters. Sending HTML-only email or email without decent line-wrapping is an invitation to have your email discarded unread. Harsh? Not at all, once you understand whom you're writing to. Most email clients are poorly suited to handling thousands of messages a day, scat- tered across dozens of mailing lists, each containing a score of simultaneous conversations. The most popular email clients make reading email easy, but they do not make it efficient; when you get that much email, efficiency is far more important than ease. As most people on those mailing lists are in a similar situation, plaintext mail is very much the standard for them.

Top-posting replies to an email is discouraged. Make any comments inline with the discussion to retain context.

On a similar note, most email attachments are unnecessary. You do not need to use OpenPGP on messages sent to a public mailing list, and those business-card attachments just demonstrate that you aren't a system admin- istrator. Don't use a long email signature. The standard for email signatures is four lines. That's it—four lines, each no longer than 72 characters.[4] Long ASCII art signatures are definitely out.

When you've composed your nicely detailed and polite question, send it to *FreeBSD-questions@FreeBSD.org*. Yes, there are other FreeBSD mailing lists, some of which are probably dedicated to what you're having trouble with. As a new user, however, your question is almost certainly best suited to the general questions mailing list. I've lurked on many of the other mailing lists for a decade now and have yet to see a new user ask a question on any of them that wouldn't have been better served by *FreeBSD-questions*. Generally, the questioner is referred back to *FreeBSD-questions* anyway. If your question needs to be asked elsewhere, someone will tell you.

This goes back to the first point about politeness. Sending a message to the architectural mailing list asking about what architectures FreeBSD runs on is only going to annoy the people who are trying to work on architectural issues. You might get an answer, but you won't make any friends. Conversely, the people on *FreeBSD-questions* are there because they're vol- unteering to help people just like you. They want to hear your intelligent, well-researched, well-documented questions. Quite a few are FreeBSD developers, and some are even Core members. Others are slightly more experienced users who have transcended what you're going through now and actively want to give you a hand up.

4. Yes, there is a standard for email signatures and how you should behave on the internet. RFC 1855 should be enforced with a spiked club and a gel-fueled flamethrower.

**Forum Posting** The forums have a different population than the mailing lists. Some devel- opers hang out there, but not as many as on the mailing lists. The people on the forums are actively interested in helping you with your problems, though.

Forums are somewhat easier than the mailing lists. You can post only in formats the website supports. There are no concerns about top-posting versus inline posting or unreadable HTML. This ease is part of their popu- larity.

If you get deeper into FreeBSD, though, you'll eventually want to join mailing lists.

But no matter which venue you choose, politeness is vital.

***Responding to Email*** Your answer might be a brief note with a URL or even just two words: *man such-and-such*. If that's what you get, that's where you need to go. Don't ask for more details until you've actually studied that resource. If you have a question about the contents of the reference you're given, or if you're con- fused by the reference, treat it as another problem. Narrow down the source of your confusion, be specific, and ask about that. Man pages and tutorials are not perfect, and some parts appear contradictory or mutually exclusive until you understand them.

Finally, follow through. If someone asks you for more information, pro- vide it. If you don't know how to provide it, learn how. If you develop a bad reputation, nobody will want to help you.

***The Internet Is Forever*** Those of us who were on the internet back in the '80s remember when we treated it as a private playground. We could say whatever we wanted, to whomever we wanted. After all, it was purely ephemeral. Nobody was keep- ing this stuff; like CB radio, you could be a total jackass and get away with it.

All those early Usenet discussions? Yeah, Google recovered them and put them online. Our beliefs were the exact opposite of true.

Potential employers, potential dates, even family members might scan the internet for your postings to mailing lists or message boards, trying to learn what sort of person you are. I've rejected hiring more than one per- son based on their postings to mailing lists and discussion boards. I want to work with a system administrator who sends polite, professional messages to support forums, not childish and incoherent rants without sufficient detail to offer any sort of guidance. And I'd think a lot less of my in-laws if I stumbled across a message from one of them on some message board where they acted like fools. FreeBSD discussions are widely archived; choose your words well, because they will haunt you for decades.

Now that you know how to get more help when things go wrong, let's install FreeBSD.

# 2

# Before You Install

Getting FreeBSD running on your com- puter isn't enough, no matter how much that first install might satisfy you. It's just as important that your install be *successful*. A successful install is one that works for its intended pur- pose. Servers have very different requirements than desktops, and a server's intended function can completely change instal- lation requirements. Proper planning before installing FreeBSD makes installations much less painful. On the downside, you'll get much less expe- rience in reinstalling FreeBSD because you'll do each install only once. If mastering the installation program through exhaustive repeated practice is your main goal, skip this boring "thinking ahead" stuff and read the next chapter.

I'm assuming that you want to run FreeBSD in the real world, doing real work, in a real environment. This environment might be your laptop— while you might argue that your laptop isn't a production system, I chal- lenge you to erase all the data on it without backing up and then tell me

it's not a production system. If you're installing on a system intended for destructive testing, and you're truly indifferent to its fate, I still recommend following best practices so that you develop good habits.

Consider what hardware you need or have. Then decide how best to use that hardware, what filesystem you should use, and how to arrange your disks. Only then should you proceed to downloading and installing FreeBSD.

Before you even start the install, though, let's look at a couple concepts you'll hit throughout your FreeBSD experience: default files and universal configuration language (UCL).

First, however, you must understand FreeBSD's default configuration filesystem.

## Default Files

FreeBSD separates configuration files into default files and customization files. The *default files* contain variable assignments and aren't intended to be edited; instead, they're designed to be overridden by another file of the same name.

Default configurations are kept in a directory called *default*. For example, the boot loader configuration file is */boot/loader.conf*, and the default configu- ration file is */boot/defaults/loader.conf*. If you want to see a comprehensive list of loader variables, check the default configuration file.

During upgrades, the installer replaces the default configuration files but doesn't touch your local configuration files. This separation ensures that your local changes remain intact while still allowing new values to be added to the system. FreeBSD adds features with every release, and its devel- opers go to great lengths to ensure that changes to these files are backward compatible. This means that you won't have to go through the upgraded configuration and manually merge in your changes; at most, you'll have to check out the new defaults file for nifty configuration opportunities and new system features.

The loader configuration file is a good example of these files. The */boot/ defaults/loader.conf* file contains dozens of entries much like this:

```
verbose_loading="NO" # Set to YES for verbose loader output
```

The variable verbose_loading defaults to NO. To change this setting, do not edit */boot/defaults/loader.conf*—instead, add the line to */boot/loader.conf* and change it there. Your */boot/loader.conf* entries override the default set- ting, and your local configuration contains only your local changes. A sys- admin can easily see what changes have been made and how this system differs from the out-of-the-box configuration.

I encourage you to keep your configuration files in a version control system. If you have a global configuration management system like Ansible, that's grand. Without such a system, a centralized repository using svn(1) or the loved-or-loathed git(1) will do. Even local revision control systems like rcs(1) can one day save your hide.

**Don't CopY the Default Conflg!**

One common mistake is to copy the default configuration to the override file and then make changes there directly. Such copying will cause major prob- lems in certain parts of the system. You might get away with it in one or two places, but eventually it will bite you. Copying */etc/defaults/rc.conf* to */etc/rc.conf*, for example, will prevent your system from booting. You have been warned.

The default configuration mechanism appears throughout FreeBSD, especially in the core system configuration.

## Configuration with UCL

The *universal configuration language*, or *UCL*, is a common library for manag- ing Unix-style configuration files. FreeBSD uses UCL for core functions, such as the packaging system.

Any file that is in UCL can appear in one of several formats, such as the traditional variable = setting format most Unix programs use, YAML, or JSON. If you've configured any Unix software before, UCL won't be a problem.

We'll see examples of UCL-style configuration throughout this book. You don't need to know the details of UCL at this time, merely that UCL is a thing in FreeBSD.

## FreeBSD Hardware

FreeBSD supports a whole bunch of hardware, including different architec- tures and devices designed for each architecture. One of the Project's goals is to support the most widely available hardware, and that list of hardware includes far more than the "personal computer." Today's fully supported *Tier 1* hardware includes 32-bit and 64-bit versions of the Intel-style processor.

Most modern hardware uses 64-bit extensions to Intel's classic 32-bit architecture. These extensions were created by AMD, and so the platform is called *amd64*. Most hardware built in the last decade uses the amd64 stan- dard. While amd64 hardware will boot both 32-bit and 64-bit versions of FreeBSD, the 32-bit version contains a bunch of workarounds to support the hardware's features and expanded address space. Run 64-bit FreeBSD on 64-bit hardware.

The traditional 32-bit IBM-compatible PC dominated computing for decades. FreeBSD supports that hardware with the *i386* platform.[1] Use

1. The i386 platform persists despite efforts to rename it *amd32*. I mean, who bought that pricey Intel hardware anyway?

the i386 version of FreeBSD only on pure 32-bit hardware. FreeBSD offers limited support for a few other hardware platforms, calling them *Tier 2 architectures*. Some of these are increasingly popular, such as ARM. FreeBSD supports both 32-bit and 64-bit ARM CPUs with the *arm* and *arm64* plat- forms. Support for 64-bit ARM hardware is improving rapidly, and you can expect ARM64 to become a Tier 1 platform soon. Other hardware platforms are on their way out and have been demoted to Tier 2 before being removed from the

source tree. Additionally, you can run FreeBSD on PowerPC (*ppc*) and 64-bit Sparc (*sparc64*) hardware, which never made it up to Tier 1. Temporary breakage of bleeding-edge FreeBSD is accept- able on Tier 2 platforms. Tier 2 platforms might or might not have packages available.

You'll also find *Tier 3* platforms, which are highly experimental. RISCV hardware is at Tier 3.

*Tier 4* includes barely supported platforms. Some of them are long obsolete and on their way out. The code still exists and could theoretically be resurrected, but nobody cares enough to do the work. Others might be on their way in but are not yet fully developed. Every platform that reaches a higher tier passes through Tier 4 on its way up.

FreeBSD supports many network cards, hard drive controllers, and other peripherals for each architecture. As many of these architectures use similar interfaces and hardware, this isn't as much of a challenge as you might think: SATA is SATA anywhere, and an Intel Ethernet card doesn't magically trans- form when you put it in an arm64 machine.

While FreeBSD runs just fine on ancient hardware, that hardware must be in acceptable condition. If your Pentium IV crashes because it has bad RAM, installing FreeBSD won't stop the crashes.

FreeBSD supports most RAID controllers and includes software to manage most of them. However, I would encourage folks running the UFS filesystem to use FreeBSD's RAID options rather than a hardware RAID con- troller. RAID controllers were created when managing storage redundancy was so computing intensive that it monopolized the host's processor. Today's computing hardware manages RAID without breaking a sweat. Additionally, RAID controllers use custom formats on hard drives. Often, the only device that can read those disks is another RAID controller of the exact same model. The unexpected demise of a RAID controller can leave you trawl- ing dubious internet auctions in search of old controllers. And if you think those controllers are expensive new, wait until they're five years old and the only folks willing to buy them are those truly desperate for that exact model! FreeBSD has a few different options for software RAID, and those disks can be read with any similar hardware.

If you're using ZFS, the warnings against RAID controllers become "just don't." ZFS expects to have direct access to the disks. Using a RAID controller disables much of ZFS's self-healing and error-correction abilities. If you must use a RAID controller, disable RAID and have it serve as a stor- age controller. While many RAID cards claim they can act as a RAID con- troller, most actually serve up a bunch of one-drive RAID containers. Verify that your RAID controller can be shifted to just-a-bunch-of-disks (JBOD) or host-bus-adapter (HBA) mode before deploying ZFS on it.

This book uses amd64 as a reference platform. Everything should work on a 32-bit i386 host, but amd64 is the world's standard these days, so we'll use it. The test systems include a couple of iXsystems storage servers and a variety of virtual machines.[2]

***Proprietary Hardware*** Some hardware vendors believe that keeping their hardware interfaces secret prevents competitors from copying their designs and breaking into their market. This has repeatedly been demonstrated to be terrible strategy, especially as the flood of generic parts has largely drowned these secretive hardware manufacturers. A few vendors still cling to their secrecy, however. We call such devices *proprietary hardware*.

Developing device drivers for a piece of hardware without its interface specifications is quite difficult. Some hardware can be well supported without full documentation and is sufficiently common to make struggling through this lack of documentation worthwhile.

If a FreeBSD developer has a piece of hardware, documentation for that hardware, and interest in that hardware, he'll probably implement support for it. If not, that hardware won't work on FreeBSD. In most cases, unsup- ported proprietary hardware can be easily replaced with less expensive and more open options.

Some vendors provide closed-source binary drivers for their hardware in the form of kernel modules (see Chapter 6).

Remember that while FreeBSD refers to the kernel as modular, that means that you can choose which parts to load and which to leave out. Once a kernel module is loaded, that module has complete access to the entire kernel. It's entirely possible for a video driver kernel module to corrupt your filesystem. I strongly encourage you to avoid binary drivers whenever possible, and to avoid hardware that requires such drivers.

### Is MY harDware supporteD?

The easiest way to determine whether a piece of hardware is supported is to boot FreeBSD on it. If you don't have physical access to the hardware yet, check *https://www.FreeBSD.org/* for the release notes for your chosen version.

2. I no longer have customers, so, sadly, I was unable to test on their hardware.

***Hardware Requirements*** Once upon a time, a host's minimal hardware requirements were a big deal. FreeBSD 1.0 supported very specific hard drive controllers and Ethernet adapters, and needed several megabytes of RAM. Hardware that couldn't run FreeBSD was still in common use back then.

Most hardware requirements are a thing of the past. Any amd64 system ever produced can run FreeBSD. Any server-grade i386 system built this millennium can run FreeBSD. Yes, a Pentium with a meager 18GB

SCSI-2 disk and a paltry 128MB of RAM offers mediocre performance, but if you want good performance, try not using that hardware.

Just because a piece of hardware should work doesn't mean it will work. "Inexpensive" is not the same as "cheap." Supported lousy hardware is still lousy. Research your hardware before buying it.

FreeBSD runs fine on hypervisors, such as VMware, VirtualBox, Xen, and KVM. Legitimate cloud providers offer FreeBSD images and ISOs. FreeBSD runs just fine on the integrated bhyve(8) hypervisor and OpenBSD's vmm(8). You can do a base install with 128MB of RAM and 1GB of disk, although you'll probably want more than that for serious experimentation.

***BIOS versus EFI*** Back in the 1980s, IBM invented the *basic input/output system (BIOS)* to handle low-level hardware tasks, like finding the operating system. Generations of IT people have argued with the BIOS. BIOS had built-in limitations that keep it from working well on modern hardware, though. The modern BIOS-like thing is called the *Extensible Firmware Interface (EFI)*. EFI is far more flexible and powerful than the BIOS. FreeBSD boots just fine from EFI, and using EFI permits FreeBSD to do some interesting things, like full-disk encryption.

If your hardware supports EFI, use it. Only fall back to BIOS mode if FreeBSD exposes a bug in your hardware's EFI implementation, in which case I'd encourage you to file a bug (see Chapter 24). Note that the hardware setup utility might call BIOS mode "legacy boot" or "ancient crap" or some such thing.

## Disks and Filesystems

Perhaps the most critical part of installing a system is how you allocate disk space and which filesystem you use. A base install of FreeBSD fits in about half a gigabyte of disk, but the filesystem beneath those files dictates much of how the system behaves.

***FreeBSD Filesystems*** FreeBSD supports two major filesystems, UFS and ZFS. Which should you use? That depends entirely on what you want to do with your system. To make a decision before booting your install media, you'll need to under- stand the basics of each.

FreeBSD's *Unix File System (UFS)* is a direct descendant of the filesystem shipped with 4.4 BSD and has been under continuous development for decades. One of UFS's original authors still hangs around the FreeBSD community actively improving the filesystem, as well as offering support and guidance to newer generations of developers. UFS's place as the pri- mordial FreeBSD filesystem has let it extend fingers throughout the operat- ing system. Many other FreeBSD filesystems attach to the kernel's virtual memory system through infrastructure created for UFS. UFS is designed to handle the most common situations effectively while reliably supporting unusual configurations. FreeBSD ships with UFS configured to be as widely useful as possible on modern hardware, but you can choose to optimize a partition for trillions of tiny files or a handful of 1TB files if you desire. *ZFS* (not an acronym) was introduced by Solaris in 2005 and inte- grated into FreeBSD in 2007. Its youth seems to be a disadvantage, but it combines technologies and concepts that have been used for much longer. ZFS computes a checksum of every block of data or metadata and can use it for error correction. Storage is pooled, meaning that you can dynami- cally add more disks to an existing ZFS filesystem without recreating the filesystem. ZFS has a whole bunch of cool features, such as highly effective built-in replication and the ability to create and remove datasets (parti- tions) on the fly.

While ZFS was written over a decade ago, it was written for future hard- ware. All of those cool features impose a performance cost, and ZFS can use a whole bunch of memory. While 32-bit systems can use ZFS, it's not rec- ommended. I resist running ZFS on hosts with less than 4GB of RAM and refuse to run it on less than 2GB of RAM. UFS serves small and embedded systems better than ZFS can.

ZFS makes a great storage system for a virtualization server, but it isn't necessarily right for virtual machines that use disk images. Many virtual machines don't get enough memory to effectively run ZFS. Additionally, I've seen more than one KVM-based virtualization system fail to migrate ZFS-based virtual machines. If you want to use ZFS on virtualized clients, be sure your virtualization system supports restoring and migrating ZFS disk images before installing a slew of hosts.

Some people insist that ZFS requires ECC RAM. ECC RAM is good, and you should get it if you can. ZFS without ECC is no worse than UFS *with* ECC, however. ECC provides a layer of integrity checks much like ZFS. If a host's non-ECC memory gets hit by a cosmic ray, ZFS writes corrupt data to disk—just as if you used UFS.

Finally, ZFS assumes you're doing things the ZFS way. ZFS is a combina- tion filesystem and volume manager. It expects access to raw disks. Never,

never, *never* use a RAID controller with ZFS; using RAID volumes as disks interferes with ZFS's self-healing features. Many RAID controllers claim to offer raw disks, but what they really offer are one-disk RAID containers.[3]

UFS isn't perfect either. A power failure or system crash can damage a UFS filesystem. Repairing that filesystem takes time and system memory. Roughly speaking, repairing each terabyte in a UFS filesystem requires

700MB of RAM. If you create a 7TB filesystem on a system with 6GB of RAM, FreeBSD can't automatically repair it.

To boil this all down, on a modern amd64 laptop or a server, I recom- mend ZFS. Test ZFS with your virtualization system. If it works, use ZFS for 64-bit virtual machines with 4GB of RAM or greater. On i386 hard- ware or 64-bit hosts with less than 4GB of RAM, use UFS.

If you're running a high-load, high-volume application and database, experiment with both UFS and ZFS on your production hardware to see which works better in your application before proceeding. Experiment with different arrangements of disks, ZFS pool types, and GEOM RAID methods. Some applications work better with UFS than ZFS. Netflix, for example, delivers all of its content from FreeBSD hosts with massive amounts of storage formatted with UFS. Before installing your massive storage server, review Chapter 12 for additional ZFS deployment considerations.

All this advice is secondary to an iron rule: choose the filesystem that best suits your environment.

***Filesystem Encryption*** Disk encryption has become a vital feature for many environments. A user that loses his laptop doesn't want to lose his data. Certain organizations require that critical data be encrypted on resting, or inactive, disks. You can't retroactively encrypt a disk on an installed system.

FreeBSD supports two disk encryption systems: *GEOM-Based Disk Encryption (GBDE)* and *GELI*. The gbde(8) encryption system is designed for use in situations where the mere existence of encrypted data can threaten the user's life. It's designed to protect a user who has a gun to their head. Thankfully, that use case is rare; this book doesn't cover it.

The geli(8) encryption system protects against more common risks. If your laptop is stolen, GELI prevents the thief from reading the hard drive.[4] If you store your company's financial records on a GELI-encrypted parti- tion, the service tech can't read it during a service call. Chapter 23 covers GELI in more detail.

Many organizations require disks containing financial data or intellectual property to be rendered unreadable when decommissioned. You can send such disks to be shredded, but encrypting the disks at install time is equally effective. The disks become unreadable when you destroy the encryption key.

3. ZFS expert Allan Jude often declares that disks plot against us, but a disk's plot pales next to a RAID controller's perfidy.

4. Mind you, casual thieves will consider a laptop running FreeBSD effectively encrypted anyway.

I recommend encrypting either the entire system or none of the system. Partially encrypted disks leave opportunities for skilled intruders to sabo- tage your system and subvert the encryption.

Decide whether or not you need encryption before proceeding.

***Disk Partitioning Methods*** Disk *partitioning* lets you divide a disk or disk array into logical units. Even hosts with average consumer-grade operating systems, such as the Windows laptop you'll find at your local big-box store, ship with multiple partitions on the hard drive. A *partitioning scheme* is the system for organizing parti- tions on a disk.

Computing is always in transition between technologies, and right now we're amidst a particularly annoying change in disk partitioning. Older and smaller hardware uses master boot record (MBR) partitioning and is always limited to disks of 2TB or smaller. Newer and larger hardware uses the more flexible and generally better GUID Partition Tables (GPT) scheme. FreeBSD manages both types of partition with gpart(8).

Which should you use in your install? Use GPT on any system that sup- ports GPT, no matter the size of the disk. Use MBR if and *only* if the system can't support GPT. (You can use gptboot(8) and gptzfsboot(8) to bludgeon GPT support onto MBR-only disks, but save that for your second or third install.)

I've encountered more than one system that supports GPT but has a hardware limitation that prevents it from using disks larger than 2TB. While MBR might seem sensible on such a system, remember that GPT is far more flexible. Even if you're a sysadmin with decades of experience with MBR, learn and use GPT.

***Partitioning with UFS*** If you decide to use UFS for your host, you'll need to consider filesystem partitioning. Thanks to the wide variety of disk sizes FreeBSD supports, the installer doesn't attempt to predict how you'll want to partition your system. Decide how to partition the disk before installing.

At a minimum, separate your operating system from your data. If this host is for user accounts, create a separate */home* partition. If you're running a database, create a partition for the database. Web servers should have a partition for web data and probably a second one for logs.

As an old Unix hand, I usually create separate */usr*, */usr/local*, */var*, */var/ log*, and */home* partitions, as well as a partition for root (*/* ) and one for swap space, plus a separate partition for the server's application data. I'm told that I'm a fuddy-duddy, though, and that my concerns about rogue pro- cesses and users filling up the hard drive are obsolete these days.[5]

5. While I won't stand in the way of progress, I reserve the right to snicker when progress drives into the ditch.

A base install of modern FreeBSD fits in about half a gigabyte. That's trivial next to today's hard drives. On a modern disk running on real hard- ware, assigning 20GB for the operating system and related programs should be more than sufficient.

If you're running FreeBSD on modern hardware, though, you probably want to use ZFS rather than UFS.

***Multiple Operating Systems*** Back in the Stone Age (roughly 2001), being able to install four operating systems on a single 6GB hard drive thrilled me. This was the only way to run multiple operating systems on a desktop without swapping hard drives.

It's still possible to do multiboot installations, but virtualization is far better. You don't have to shut down your main operating system to access one of the other operating systems. The bhyve(8) hypervisor lets you run other operating systems, including Microsoft Windows, on top of FreeBSD. Other operating systems have hypervisors that let you run FreeBSD on top of them.

***Multiple Hard Drives*** If you have multiple hard drives in your host, you should almost certainly use them to create some sort of storage redundancy. If you're using ZFS, use a mirror or some sort of RAID-Z (see Chapter 12). If you use UFS, FreeBSD supports software RAID. When you have a whole bunch of hard drives, though, life gets a little more complicated.

The rule of thumb is still to separate your operating system from your application data. If you have 30 hard drives, mirror 2 of them for your operat- ing system install and use the others for your data. Like all rules of thumb, this is debatable. But no sysadmin will tell you that this is an actively bad idea.

With many hard drives, consider which data passes through which disk controller. If a disk controller dies, what happens to your system? If both of your operating system disks are attached to a single controller and the con- troller dies, your host goes down. Putting each drive on a different control- ler offers redundancy. Ideally, attach your mirrored operating system disks to different drive controllers.

Also, remember that SATA disk controllers split all their data through- put among all the hard drives connected to them. If you have two disks on a SATA controller, each disk works, on average, about half as fast as it would work alone on the same channel. Port multipliers add disks but slash per- disk performance.

***Swap Space*** When FreeBSD (and any other modern operating system) uses up all the physical RAM, it can move information that's been sitting idle from memory into swap. Now that even laptops ship with 32GB of RAM, it's hard

to imagine a host running out of memory, but never underestimate a pro- gram's ability to devour RAM. Virtual systems might be allocated very tiny amounts of RAM.

So, how much swap space do you need? This is a matter of long debate between sysadmins. The short answer is, "It depends." What does it depend on? *Everything*. Long-running wisdom claimed that a host should have twice as much swap as it has physical memory, but today that's not only obsolete but dangerous. When a process starts catastrophically allocating memory—say, in a bug caused by an infinite loop—the kernel kills the process once the system runs out of virtual memory. A system with 32GB of RAM and 64GB of swap has 96GB of virtual memory. The i386 platform limits memory usage to 512MB per process, which means that the kernel stops such runaway pro- cesses pretty quickly. 64-bit systems, like amd64, have vast virtual memory spaces. A system thrashing gigabytes of memory between disk and RAM will be excruciatingly slow. A modern host should have only enough swap space to perform its task.

Multiple hard drives let you increase the efficiency of swap space by splitting it between disks on different drive controllers. Remember, though, that a crash dump must fit entirely within a single swap partition. FreeBSD compresses crash dumps so that they don't take up as much room, but still, many small swap partitions can be counterproductive. If you have a large number of drives, don't use the application drives for swap; restrain swap space to the operating system drives.

The main use for swap on modern systems is to have a place to store a memory dump should the system panic and crash. FreeBSD uses kernel minidumps, so they dump only the kernel memory. A minidump is much

smaller than a full dump: a host with 8GB RAM has an average minidump size of about 250MB. Provisioning a gigabyte of swap per 10GB of RAM should be sufficient for most situations.

If you have a truly intractable problem, though, you might need to dump the entire contents of your RAM to swap. If I'm setting up an impor- tant production system, I always create an unused partition larger than the host's greatest possible virtual memory space and tell the host to dump the kernel to that partition. If my laptop has such a problem, I'll just plug in a flash drive and configure the system to dump on it instead.

## Getting FreeBSD

Now that you've made all your decisions, you need a copy of FreeBSD. If this is your first time installing FreeBSD, go to *https://www.FreeBSD.org/* and look for the Get FreeBSD section at the top. Right by that, you'll see a list of supported releases, including (probably) two releases recommended for production. Sometimes there's one. Sometimes there's three, but usu- ally two.

***FreeBSD Versions*** Two production releases? What madness is this?

FreeBSD development occurs in multiple tracks, as I will discuss in Chapter 18. A few tracks coexist, in various states of support. Each track receives bugfixes and incremental improvements. Newer tracks get new features.

As I write this, FreeBSD.org lists two production releases, numbered 11.0 and 10.4. Version 11.0 is the most recently released version, but it's also a .0 release. It's the first release on this track. It will have the newest features, but it has the greatest likelihood of including unknown bugs. Version 10.4 is slightly older and lacks some features in version 11.0, but it's the fourth release along that track. It's not guaranteed to be bug free, but many people have run it in production for months or years. Any screamingly obvious prob- lems have been fixed.

Every FreeBSD release eventually reaches *End of Life (EoL)* and loses support. The security team stops producing patches and new packages are no longer available. The older release will reach EoL before the newer version. If you install FreeBSD 10.4 today, you'll need to upgrade to 11 at some point—but by then, you'll be upgrading to something that's not a .0 release.

FreeBSD averages two production releases at a time. This isn't an invio- late rule, only observed behavior. Sometime around when the 12.0 release escapes, the 10 branch will reach EoL.

I personally will run FreeBSD .0 releases, but having been burned with other operating systems before, I sympathize with the folks who categori- cally reject .0 versions. If you've never used FreeBSD before, I recommend installing the most recent production release. It has the latest device drivers and newest features.

Follow the download link and grab your chosen version.

***Choosing Installation Images*** You can choose between several different formats of FreeBSD installation media. All installation media is available both compressed with xz(1) and uncompressed. If you can conveniently extract *.xz* files, download the com- pressed versions. This saves the donated bandwidth and reduces download time. Any modern operating system can either handle *.xz* files natively or has add-on software for the task.

FreeBSD offers two styles of installation media. The first contains only enough to boot the FreeBSD installer and bring up the network. The installer then downloads the operating system files from a FreeBSD mirror site. If you're going to do multiple installs of the same FreeBSD version, though, you're better off downloading an installer that includes the operat- ing system files.

The installer comes in both optical disk (*.iso*) and flash (*.img*) formats. Choose the format that fits your system. If you're installing a virtual machine, an ISO is probably simplest.