!!! EARLY DRAFT !!!

The Penguin’s Guide to Daemonland

An Introduction to FreeBSD for Linux Users

2nd January 2021

Contents

Legal

Placeholder for legal

Contents

Chapter Overview

This book is divided into six parts (I to VI). They are generally meant to be read in order except for part VI which contains optional background information. You are of course free to skip around but might miss some things if you start with later chapters.

Part I

Part I is named “FreeBSD Quickstart” and consists of five chapters providing exactly that. It’s important for people who are making their first contact with FreeBSD. This part starts with the *Pitfalls* chapter, meant to highlight some important differences between a Linux and a FreeBSD environment.

Chapter two, *Differences*, explains some important concepts and deepens your understanding of what you read in the chapter before. Pay attention to this if you’re new to FreeBSD – it could save you quite a bit of confusion later!

The next one is the “Toy VM” chapter guiding you to get quick access to a FreeBSD Virtual Machine. This is great for people who want to dive right in instead of reading about system installation first. That chapter is completely optional of course. If you have access to a test machine that runs FreeBSD or if you decide to install a system yourself, that’s fine, too. In the latter case you may want read the *installation* chapter from the next part instead. Which ever way you choose: It makes a lot of sense to have access to a FreeBSD system for chapter four and later. While just reading everything may do the trick for a few special people, it won’t stick for most of us if we never actually did it ourselves.

Chapter four gives you the *Administration Basics* – this is the chapter you would want to read if you’ve got a job interview tomorrow and you know that it’ll involve some FreeBSD! It will introduce you to doing some of the most important administration tasks like basic package and service management.

Finally chapter five explains what so-called "Linuxisms" are and how to overcome common problems which they pose.

Placeholder for Parts II to V

Part VI

All of the chapters in this part are optional as you can definitely use FreeBSD without knowing a lot of background information. However it's there for a good reason: The knowledge presented here can be anything from interesting to valuable depending on what your relationship and use case for FreeBSD is.

This is the *History* chapter that covers what computing was like at the time when Unix was born, why Unix was made in the first place and what downright incredible things happened with it. It can be useful for people who have always wondered why the things we have today are the way they are (and not totally different).

The next chapter, *Family Matters*, goes over what Unix-likes are, how FreeBSD's relation to other such Unix-likes are and what the other BSDs are like. If you want to know more of the broader Unix family in general or the BSD family in particular, this chapter should help you pick which other operating systems might be for you to explore, too.

Finally there is this book's last chapter, *Culture and Community*. It covers how the FreeBSD project is organized and how the community works. If you'd like to get involved with FreeBSD, this should provide you with some information to make things as easy as possible for you.

Types of Readers / How to Read

In general I am taking three groups of readers into account, giving some suggestions on how to read this book:

1. People who want to get up to speed with FreeBSD gently but as quick as possible and have no time or interest in looking left and right

2. People who prefer to take their time with the matter and enjoy additional information that helps to understand the whole topic quite a bit better
3. People who have used FreeBSD ages ago and are returning to it after not having worked with it for years

If you fall into **group one**, just start with part I, then read parts II and III. When you are done with that, it's time to actually get your hands dirty. Have a look at part IV and pick a couple of the example projects that are of interest to you and get e.g. a webserver or a monitoring system up and running. Read or skip the suggested alternatives as you please.

Then proceed to part V if you are interested in how using FreeBSD as a desktop works. Even if you plan to only use FreeBSD for servers it can be a good opportunity to learn some more things about the OS. The choice is yours of course. Part VI is there in case you later become interested in the history of Unix, how some of the other BSDs compare to FreeBSD and such.

Should you belong to **group two**, I recommend to actually start reading the fist two chapters of part VI first. You will find quite a bit of information on how it all began and why what we have today is the way it is. In addition to that you will get a good first understanding of the broader *nix landscape and learn where FreeBSD and Linux fit into it. Then read parts I, II and III.

After finishing those, take a look at what part IV has to offer, pick a couple of projects and do useful things with FreeBSD. If you like what you see you might want to read part V, too, and maybe give desktop FreeBSD a try. Even if you only follow along with an old laptop, there's plenty to be learned about the OS that can be helpful in server-only environments as well. Maybe while working with this book you found that you enjoy working with FreeBSD? Then read also the final chapter of part VI to learn more about FreeBSD's community and how to interact with it.

If you are a person that fits into **group three**, you can really read whatever you wish. The longer you've been away the more will have changed, though and it might make sense to start at the beginning. If you departed after FreeBSD 7.x, you will find that among other things the old pkg_* tools have been replaced. Should you have left after 4.x you are in for quite a few great improvements: Forget disklabel(8) and friends and say hello to GEOM. Meet the rc.d init system that replaced the old BSD rc. Oh, and CVS is dead, too!

Chances ae that you can skip part I entirely. You may want to still read the first chapter (Penguin Pitfalls), though, because it'll remind you of some things to watch out for (and probably amuses you, too). Parts II and III should get you familiar with what FreeBSD looks like today. Part IV could give you a couple of ideas for some small (or not *that* small) projects. And maybe you've always wanted to try out FreeBSD on the desktop? There really has been no better time than now. A lot of the pains of the old days are gone – and while FreeBSD *is* admittedly less polished on the desktop than Linux, it works well enough to be a sensible option.

When it comes to part VI, it completely depends on what you already know and what's of interest to you. Take a look at the TOC and read what suits you. Besides… welcome back!

Preface

About This Book

The *Penguin's Guide* is an Open Source book on learning the FreeBSD® operating system written for Linux® users. Therefore you should already have an idea of what *Open Source* means as well as what *Linux* is as some prior knowledge is assumed (see next section). However if you don't really know what *FreeBSD* is just yet, that's fine. You'll be a confident FreeBSD user before too long if you decide to follow along.

You should have received your copy of this book for free. It was not written for profit but in the hope that it would be useful to Linux users interested in getting to know FreeBSD and adding some new skills to their tool belt. You are allowed – and in fact encouraged – to give it to other people who might be interested in it **[once it's more than a draft that is likely to contain too many errors and would lead people astray – don't give it to non-experienced FreeBSD users, yet!]**.

As an Open Source project you can help to improve it: Send in corrections, point out things that you didn't understand or provide additional material / suggestions that could be valuable to future readers (and get credit if your contribution gets included in the book). Contact info: **freebsdbook@elderlinux.org**

While you might think that you surely cannot contribute anything as a newcomer, the BSD community has a different view on this. You being a newcomer is in fact an interesting attribute: There are things that a veteran *will not see* because of experience. An unprejudiced opinion on something can be valuable input or feedback.

So if you take notes during your journey to learn FreeBSD, we'd be interested in e.g. what was hard for you to understand or get used to, what still makes little sense even after dealing with it for a while, what could be made more beginner-friendly (and probably how), etc. We would seriously appreciate if you were to let us know! Your feedback can actually make a difference.

Audience

As the subtitle states, this book is written as an introduction to FreeBSD for Linux users. Which means the reader is obviously assumed to posses some prior knowledge regarding Linux. But what exactly does "some knowledge" mean?

In general this book was written with "the average Linux user" in mind (as the author pictures one). That means if you are a complete novice who has only been running Ubuntu for a couple of weeks, you might not be able to follow easily all the time and without doing some additional research. If however you've used more than two or three distributions, are not afraid of the command line and know how the system works in general you almost certainly qualify.

There are parts where some more in-depth knowledge is helpful, but the author tries to make this optional information. You also don't need programming skills, are not expected to understand *awk* magick or have mastered *vim* wizardry nor are you required to know every single package available for your preferred flavour of Linux.

Speaking of distributions: There are cases with references to particular distros. This is usually done to give you a comparison with something you might already know. If you don't know Gentoo for example and have no idea what *portage* is, such a comparison might not help you – but it will help others. If you care enough you could do some reading on that topic, too, and thus explore some more corners of the Linux ecosystem alongside your journey to learn FreeBSD. As always this is your decision.

There is no single distribution which this book is targeted at. If you have a Debian background, that's perfectly ok. Fedora for you? Fine, too. The author has been an Arch Linux user for years before venturing off into Daemonland "for a few weeks" mostly out of curiosity (and then liked it there enough to make this place his home permanently). So while there might be a slight prejudice towards the Arch Linux state of affairs, he is working with Debian-based distributions and CentOS, too, and has also used Alpine Linux, Gentoo, Slackware and SuSE in the past as well.

I assume there are few people out there who are into speciality Linux distributions only (e.g. embedded Linux) and have no idea of common distros at all. If that is you – perhaps you'll need to take a look at the manpage of some utility you're not familiar with. On the other hand maybe you consider yourself an expert Linux user. In this case some parts may be somewhat boring for you. Feel free to just skip them.

So who might want to read this book and learn FreeBSD after already being capable of using Linux?

- Sysadmins who want to (or have to) take care of some FreeBSD systems in addition to their Linux machines
- People looking for something to put into their resume that sets them apart from common Linux admins that are ten a penny
- Employees wondering if using FreeBSD for some tasks might benefit their organization
- Programmers who are to evaluate basing their product on FreeBSD (e.g. for licensing reasons)
- Interested users who'd like to take a look beyond the Linux teacup
- Users who like permissive licenses, democratic project leadership or who dislike monopolies and want to diversify their infrastructure
- People unhappy with the direction that Linux is currently taking and who want to check out alternatives
- Any Linux user who has heard about FreeBSD a couple of times and finally wants to know more

Do you fall into one of these groups or do you have another reason to learn FreeBSD? Even if you are not entirely sure you might decide to simply read on for a couple of chapters and see if you get the feeling of learning something worthwhile.

Why Even Bother?

Chances are that everything you know about FreeBSD, yet, is information you read on the Internet. With this comes the usual problem of unreliable sources. There are a lot of people out there who will doubt that you are serious if you say that you work with FreeBSD. Must be a joke, right? There also is this somewhat popular claim that "BSD is dying". And since "nobody uses FreeBSD anymore" (another such claim), why should you even bother to learn it?

The "BSD is dying" story can however be compared to the "year of the Linux desktop" prediction: It has been made time and time again but never actually happened. In fact FreeBSD is very much alive, it just doesn't get as much attention in the media, silently doing its work. There are various big companies who use and support FreeBSD for one reason or another, though. You will probably be surprised that almost every person that you know benefits from FreeBSD at least indirectly!

You doubt that this is true? Let's take a look at a few examples as it's interesting to see why some of those chose FreeBSD. Traditionally you'd have named companies like Yahoo! and Hotmail (yes *that* Hotmail that Microsoft®bought and had trouble to migrate to Windows™servers, thus continuing to run FreeBSD in-house for years). It's not too hard to figure why they chose FreeBSD over Linux: While the latter did exist at that time, it was in its early infancy and generally regarded a toy OS. FreeBSD on the other hand was already a production-ready and battle-tested operating system that had earned an excellent reputation.

Today you'd probably name well-known companies like Netflix and WhatsApp first. Both make use of FreeBSD, e.g. because of its world-class networking capabilities. It's only natural that a company which alone produces roughly 1/3 of all the US-based Internet traffic (Netflix) is really interested in a high-performance networking stack. Both companies also chose FreeBSD because they had staff already familiar with it.

There are others like Juniper whose JunOS for their firewall appliances is based on FreeBSD and Sony used it as the building block for their Playstation 4's Orbis OS (and likely for the PS5, too). Both probably chose it for the simple reason that they liked the license as it allowed them to close the source for their operating system.

Xinuos who acquired the rights to UnixWare®as well as to OpenServer®chose to base their new product on FreeBSD. Providing real long-term support, they outclass even Red Hat®, the recognized leader in LTS for Linux – by e.g. still supporting a 25+ year old product! Thinking in such long terms they probably wouldn't have settled FreeBSD if they didn't believe in its maintainability and technical merits.

The LPI (Linux Professional Institute) has teamed up with the formerly independent BSD Certification group to offer a new certificate: The *LPI BSD Specialist*. It's the same organization that issues the well-known LPIC-1 to LPIC-3 certificates.

Even the server of distrowatch.org, a site you surely know if you've been using Linux for a longer time, runs FreeBSD. It started on FreeBSD, switched to Debian when the hardware broke and the system had to be replaced in a rush. But with the next hardware change, they switched back to FreeBSD. And probably nobody is going to argue that these people who try out and review Linux distributions all the time did it because they don't know Linux!

Also FreeBSD is pretty strong when it comes to storage. Thanks to its *GEOM storage framework* it is extremely flexible when it comes to truly fit even exotic needs. The excellent ZFS integration is another big plus for people who value their data. It is no surprise that iXsystems®with their TrueNAS™platform is traditionally FreeBSD-based (there's a Linux-based version now, too, because their customers kept asking for support of Docker containers).

Speaking of containers: FreeBSD came up with containerization back in the last millennium. Its legendary *jails* have inspired Solaris' zones and are somewhat similar to the containers that are popular on Linux today – except they were conceived with security in mind right from the start.

Then there's the *ports framework* used for package building that allows for easy customization of build-time options for various software. Gentoo's portage system heavily draws from that concept (hence the name). Customizing and rolling your own packages is very easy. Building and updating FreeBSD from source is even easier.

FreeBSD offers three firewalls to choose from, among them Pf which is by many considered the most advanced Open Source firewall in existence. Fed up with iptables? Try Pf for a change – and experience actually readable rule sets!

In many regards Linux and FreeBSD complement each other pretty well. Ready to get the best of both worlds? Learn more about FreeBSD to find out how and where it might fit into your environment.

FreeBSD for Linux Users

Dear Linux user! Do you still remember the first steps you did with Linux? If you were coming from an operating system like Windows, you were likely in for quite some hours of reading, exploring and trying to figure things out. Obviously you didn't give up even if there were frustrating moments – because in the end being able to use an Open Source operating system that doesn't spy on you and puts *you* in control was well worth the effort.

Here's the good news for you: Since Linux and the FreeBSD both belong to the family of so-called *Unix-likes*, they bear a lot of similarities. You can rest assured that the filesystem tree starts with the root directory ("/") and you will find a familiar structure with "/etc", "/dev", "/usr/bin", "/var/log", and so on. The command "ls -lah ~" will let you know which files, dot-files and subdirectories are in your home directory along with information about permissions, human-readable sizes and so on.

Looks like you can feel at home right away, right? For a good part of affairs: Yes. So do you just have to figure out how the package manager works and be good to go? Not entirely. While many things work the same there are also quite a few differences. Many of those are obvious when you encounter them but some are more subtle and bear the risk for you to trip over. If it wasn't for those differences which include some of the strong points of FreeBSD, it wouldn't make too much sense to dig into said operating system. You could just try out some different Linux distribution. Also you certainly wouldn't need a book like this.

Getting back to the examples above, you will find some "odd" things if you look at the filesystem hierarchy a bit closer. E.g. there's no "/proc" on FreeBSD (at least not by default that is). While you can of course retrieve information that you'd expect from the process filesystem, you do that (as well as setting values) via the *sysctl*(8) command.

You can configure the SSH daemon in "/etc/ssh/sshd_config", but even if you've installed *sudo*, there won't be a "/etc/sudoers" file – because it is in another place instead, "/usr/local/etc/sudoers"! Pretty strange, isn't it? No, on the contrary, and here is why: FreeBSD ships with a lot of basic programs that are considered part of the actual operating system (often called *base system* or sometimes just as *base*). Any third party package is kept separate from the base system. Those packages live in "/usr/local" and its subdirectories. While at first it's a new concept to grasp, a lot of people come to appreciate that clean separation before too long (especially when they realize that there are real benefits to it).

Should you for example be used to working with "ls -v" on Linux you'll find that this flag is not available in FreeBSD's version of *ls*. On the other hand, there's functionality there which is not present in GNU coreutils's *ls*. As you can see, FreeBSD – while not wildly different – *is* different enough from Linux to warrant your attention when dealing with it. You will also have to re-learn a couple of things.

The best advice I can give you at this point is to look at differences with an open mind. Remember just how different Linux is to Windows and how strange things probably were when you first encountered them. Judging Linux from a point of view where the way Windows handles things is "normal" will lead to looking down upon Linux for being different. It takes an open mind to accept that differences can be – and in this case most often (filesystem organization, package management, modularity, ...) actually *are* – for the better.

The same thing is true if you compare FreeBSD to Linux. Try not to fall into the trap of considering things you are more familiar with as "superior" automatically. If you're new to something you cannot really judge it as you are missing information on why it is done that way. Later you might find out what the reason is and what looked just plain weird before might turn out to be a perfectly sensible solution – or even an unsuspectedly brilliant approach to certain things.

Part I.

FreeBSD Quickstart

1. Popular Penguin Pitfalls!

Placeholder

2. (Some) Important Differences to be Aware of

3. Your FreeBSD Toy VM

4. Administration Basics for the Impatient

5. Identifying “Linuxisms” and Living Without them

Part II.

Managing FreeBSD

6. Installation

7. Disk Partitioning and Filesystems

8. System Boot & Service Management

9. Users and Permissions

10. Networking

11. Updating the OS

12. Timekeeping

13. Package Management

14. Logging

15. Firewalling

16. System Mail

17. Foreign Filesystems & FUSE

Part III.

(Slightly) Advanced Topics

18. Breaking and Repairing the System

19. Using ZFS

20. Tuning FreeBSD

21. Secure Levels

22. Updating from Source

23. Using Ports

24. Jails

25. Bhyve

26. mfsBSD

27. Linux Emulation

Part IV.

FreeBSD by Example

28. Rolling Customized Packages

29. NFS Server

30. ZFS Replication

31. Simple Web Stack

32. DNS Server with BIND

33. VPN with OpenVPN

34. Jailing Web, DB, BIND and OpenVPN

35. Managing TLS Certificates with LE

36. Mailserver with Postfix

37. Doing Backups with Bareos

38. Monitoring with NRPE & Icinga

39. PXE Booting Multiple Operating Systems

40. Configuration Management and Automation with SaltStack

Part V.

FreeBSD Workstation & Laptop

41. Graphical FreeBSD with X11

42. Window Managers

43. Desktop Environments

44. Display Managers

45. Making Your Desktop More Comfortable

Part VI.

Background Information

46. A (very) Brief History of Unix

Reading this chapter is not in any way a hard requirement. Both Linux and FreeBSD are very much usable without knowing a lot about where they come from. Should you have absolutely no interest in history whatsoever, you'll probably want to just skip this background chapter.

If however you have a sense for history you will most likely enjoy the following pages. Knowing a little about computer history can help you a great deal in understanding where technology as we know it today came from – and thus why it is like it is. That in turn helps you to make the most out of it.

Claiming that Unix has an interesting history is an understatement. Sure, there are parts that are only of interest to a few people. But believe it or not, there actually are *Unix historians* and even organisations like tuhs.org (the *Unix Heritage Society*) trying to preserve as much of its history as possible! Other parts of the Unix story however could well have originated from an agent thriller (e.g. conspirative phone calls asking a person to come to a certain place where then a tape with the newest version of Unix was found...).

There are a whole lot of hilarious to downright unbelievable stories surrounding Unix like e.g. AT&T declaring Unix as *industrial waste* (sic!) for tax reasons when they licensed it to others! I'm not going into a lot of detail here, though, merely aiming to give you a useful overview. Anybody choosing to dig deeper is guaranteed more than one good laugh, though. The history of Unix is full of irony, random but important events and such. See *literature* in the appendix for some recommendations.

Unix – believe it or not – is already half a century old! The world was quite different back in the day and it's common knowledge that in the extremely fast-paced IT business *one* decade worth of progress can already alter things beyond recognition. Therefore if your hair isn't white (or at least grey) already, a little history may be in order if you as a person living in today's world want to grasp just what made Unix such a special thing.

46.1. Mainframes and Multics

It all begun in the age of very big, very slow and extremely expensive mainframe computers. Those were designed according to a "Batch-Processing" model: Programs were written offline and stored e.g. on punch cards; those were then queued as jobs and run one after the other in order to avoid wasting precious calculation time waiting for slow typing humans to finish inputting their program. Eventually the calculations were done and the output was handed back to the programmer (maybe hours, maybe days later!). Letting a user program directly at the computer and having the machine be idle for that time was unthinkable in most environments.

The severe downsides of this practice are obvious. So a new idea arose in the mid 1950's: The *Time-Sharing* concept. If one person occupying the computer was a guarantee for "dead times" while entering the program interactively – couldn't this problem be overcome if multiple users worked on the same machine simultaneously?

As new computers became available, fast enough to allow for the constant switching required to do this, the influential *Compatible Time-Sharing System* (CTSS) was created in the early 1960's. "Compatible" in this case meant that it was a system that did the classical Batch-Processing of the "main" programs while at the same time distributing some processing time to allow for interactive use by programmers at a terminal.

In the mid 1960's MIT, General Electric and AT&T's Bell Labs begun to work on a Time-Sharing operating system called *Multiplexed Information and Computing Service* (or Multics). It was extremely ambitious and brought with it a plethora of new ideas. One of those was the so-called "single-level store", a concept that basically did away with the distinction between data in RAM and on disk. A file and process memory were the same thing (called a *segment*) from the perspective of the application. This by the way is where the familiar *ls* command in Unix really comes from: The abbreviation actually once meant "list segments"!

Multics pioneered a hierarchical filesystem (i. e. it supported subdirectories) and already offered symbolic links. The OS featured background processes (system services) which were already called *daemons*. It also introduced *dynamic linking*, allowing programs to use external libraries linked in at runtime as well as many other novel ideas.

The development of the system had already taken a long time and burnt a lot of money when in 1969 Bell Labs finally lost faith in it and pulled out of the project. Multics was eventually completed by the remaining team and marketed but is regarded as a commercial failure by most people. It was criticized as overly complex, (initially) too slow and big as well as too expensive.

46.2. Ancient Unix

What was to become Unix started with the dissatisfaction of several Bell Labs employees over the loss of what they found had been a convenient platform for programming: Multics. They made scratches (on paper and boards) for one of the things that had fascinated them most with it – a hierarchical filesystem.

Ken Thompson decided that he would write his own operating system. As perceived complexity had been one of the major reasons for abandoning Multics he wanted to go the opposite way: A radically simple operating system for programmers. He created some prototypes of a kernel to run on the GE-645 mainframe that they had used so far. But his effort suffered a severe blow quite soon: With Multics cancelled within Bell Labs, it was decided to dispose of the expensive GE machine.

Some of the staff still wanted to continue. There was a little problem, however: The official statement of Bell Labs was that after the Multics debacle they *would not* participate in any work related to operating systems anymore! A small team comprising of Thompson, Dennis Ritchie and a few others decided to ignore this and to continue with OS programming nevertheless. They tried to come up with good arguments to buy a computer for them to use but failed to convince their superiors.

Thompson went looking for a machine they could use for the their work (and also to run the game on that he had written: *Space Travel*!). He found a spare machine in another department, an old DEC PDP-7 – not quite what they wanted but better than nothing. The machine didn't provide a programming environment, so they initially had to use the GE-645 to generate the code for the PDP-7 and then transfer it over... The machine also proved to be too slow for the game, but it was at least sufficient for a very simple OS.

Initially this unnamed project was a primitive single-user and single-tasking operating system. The team continued to improve their OS and (due to the very limited memory available) soon wished they had more capable hardware to develop on – which was completely out of question without the blessing of their bosses.

When the patent department expressed the need for a text processor, Thompson and Ritchie jumped the opportunity and offered to provide one. Sure, on closer inspection it would require an operating system to run on, too – but of course there was no need to talk too much about that aspect specifically, was there? So the reference to the OS was hidden in a footnote – and this time the superiors took the bait.

With it being an official project now, a newer PDP-11 could finally be bought and Unix was quickly ported to the new machine. After completion of a text editor and a text-formatting utility which received praise from the patent department, the team had succeeded in making their formerly secret OS project not just an official but also a successful one.

Peter Neumann proposed the Name *Uniplexed Information and Computing Service* as a pun on Multics. Following a joke by Brian Kernighan ("Emasculated Multics is Unics"), the system that now supported *two* users concurrently, was pronounced "eunuchs" initially. It is unknown how the name morphed to *Unix* later.

Other departments within Bell Labs also acquired PDP-11s and chose to run Unix on them instead of the official DEC operating system. A little later *man pages* were invented to help users new to the system.

Various versions of the so-called *Research Unix* existed at first within Bell Labs and later outside as software licensed to others, too. Since the actual system was constantly under development, these versions refer to the editions of the manual that came with the system. Here's just a couple of facts around these:

- V1 (Nov. 1971): Unix on the PDP-11, for the first time with a manual
- V2 (Jun. 1972): C compiler included
- V3 (Feb. 1973): Invention of pipes, filesystem split between /bin and /usr/bin (``usr" actually stood for ``unix system resources")
- V4 (Nov. 1973): Operating System re-written in C, the programming language created by Ritchie; first version to be presented to the outside world
- V5 (Jun. 1974): Widely licensed primarily to universities; ported to various machines similar to the original PDP-11/20

- V6 (May 1975): First port to a very different platform; licenses available for commercial users
- V7 (Jan. 1979): Introduction of the Bourne shell and many classic Unix tools like *awk*, *make* and *tar*

There were versions V8 to V10 between 1985 and 1989, too. Those late *Research* versions were mostly used internally before the team moved on to creating a successor to Unix (named *Plan 9*). However they did not have as much of as an impact on the main Unix development anymore, since a lot had happened in between. V8 was also based on a BSD release (4.1c). More about that in a minute.

46.3. Law, Money – and Users

There are two more topics to know about in order to understand the truly unique situation which made the Unix success story possible at all. One is legal requirements for and monetary interests of AT&T, the other is Unix user's efforts of organizing.

As stated above, Unix was licensed to various educational and commercial organizations. This did not happen deliberately because managers understood what they had with Unix, though. On the contrary! During that time, AT&T still maintained its telephone monopoly and after antitrust cases it was bound by special regulations. One of these forbid it to engage in any fields outside of communication (so the computer business was clearly off limits) and another required them to license their patents to others upon request. The company didn't care for software.

After Unix was presented on a conference and papers about it were published, a lot of academic interest in the operating system quickly built up. In the beginning the license fees were pretty low ($99), but before too long, AT&T was nearly blown away by the number of licensing requests and eventually more people (staff in the patent department, lawyers, etc.) did Unix-related work than there were programmers improving the code! The company begun to realize just how valuable Unix actually was. Licensing fees kept increasing and increasing with each new edition (to a towering $250,000 in the end).

Ironically this early success of Unix threatened its further development. Thanks to their experiences around Multics, Thompson and Ritchie had realized early on how important the *community aspect* would be for such a project if it was to thrive. However the company's lawyers were busy fighting the forming user communities – the more successful Unix became the harder they tried!

An example of this is the story of the legendary *Lions' Commentary on UNIX 6th Edition*. The book was written by Australian professor John Lions for his computer science courses and included the source code for the kernel. It was available for Unix licensees and spread like wildfire. This was possible because the license of V6 Unix allowed classroom use of the code. When V7 was released, AT&T specifically disallowed this use, so there never was a new edition of the book and many universities just stopped teaching practical Unix, again concentrating on theory only. The *Lions book* however became the most photocopied printed work in computer science.

It's also very interesting why Unix users had such a huge drive to get organized in the first place. The terms under which you could get Unix were a little ... well, let's say: special. They are commonly summed up like this:

- **No advertising**
- **No support**
- **No bug fixes**
- **Payment in advance**

Sounds extremely tempting, doesn't it? Since users were officially left alone with the "as-is" software, they had to help each other. Forming user groups was simply a logical step to take and people soon did that. As the code for the system was available, users made fixes themselves and shared them. Often they were applied to the official Unix as well and thus included in the next edition. Many new features also started as such outside contributions.

Eventually AT&T was subject to a forced break-up. The new situation enabled Unix to become a very profitable commercial product after all – which in turn led to more trouble with the community that had formed around it, considering it *their* system! AT&T had already taken many actions that made the "we against them" thinking grow.

46.4. Original BSD: The Berkeley Software Distribution

The University of California, Berkeley has been an important hot-spot in Unix development. Bill Joy who later co-founded Sun Microsystems, was very active in the CSRG (Computer Systems Research Group). He was responsible for BSD, the *Berkeley Software Distribution*. 1BSD and then 2BSD consisted of additional programs for V6 Unix (the latter including e.g. Joy's *c shell* and his new visual editor: *vi* – which is still popular today at least in form of its offspring *vim* and *neovim*).

When the CSRG got VAX minicomputers they begun working with UNIX/32V, the VAX port of V7 Unix. A large portion of the kernel was rewritten to support virtual memory and many other improvements were made. Since the changes were so invasive, Joy chose to distribute the entire operating system as 3BSD. Those releases were sold to holders of valid Unix licenses – and soon people commonly purchased Unix from AT&T for the license only but actually installed and ran what was commonly called *BSD Unix* instead.

Eric Allman (known e.g. for *sendmail*) said about early Unix: "[...] 4th Edition was pretty flaky. It was a system that only a researcher could love. It was slow. It didn't have a lot of tools." While the situation had somewhat improved with later editions, it was BSD where things really took off: A new filesystem, the *Fast Filesystem* was introduced, new technology like Unix domain sockets included and over time practically no part of the OS remained untouched.

This led to increasing complexity and some Unix purists therefore declared V7 Unix the pinnacle of its evolution. However it was BSD that made Unix useful and usable for real-world tasks. As the word spread, CSRG received funding e.g. by DARPA (Defense Advanced Research Projects Agency of the US Department of Defense) after they had chosen to use it for their ARPANET project. Part of the contract was to put an early TCP/IP stack into BSD. Joy refused to use the stack provided by an outside company because he felt the implementation was simply not good enough. Instead he chose to write his own much faster implementation and DARPA eventually accepted that it was superior.

CSRG decided to release the TCP/IP code under a very permissive license to the world as *Networking Release 1* (or Net/1). Since it was all their code, no blessing from AT&T was required to do this. The novel thing here was that it was released to the general public without requiring any proof of a valid Unix license as had been required for 3BSD and 4BSD so far.

Soon the idea came about to make another such freely available release with all the Berkeley bits and not just the network stack. While working on that, people realized that they had replaced such a huge part of the operating system that it was in fact feasible to actually replace the remaining tools, too. The project was completed in one and a half years. A couple of files for the kernel were all the AT&T code that remained. They were deleted and NET/2 (which was as such not a complete working OS but very close to that) released.

When the funding ended and after legal trouble (more on that in a minute) the CSRG was dissolved and development of BSD at Berkeley ceased after the final gift to the world in form of *4.4BSD-Lite Release 2*.

46.5. Commercial Unix and Standardization Efforts

After AT&T was legally able to compete in the software market, they didn't hesitate for long: Various Unix-related projects as well as the newest evolution of V7 were combined to form what was called *System III*. This was the first commercial release Unix and the end user would no longer get the source code with the product. One important innovation was the introduction of *named pipes*.

AT&T licensed System III to various companies which then sold Unix derivatives. Products like *HP-UX* (HP), *Irix* (SGI), *Sinix* (Siemens Nixdorf), *Ultrix* (DEC) and *Xenix* (Microsoft) were initially based on System III.

Unix followed a versioning scheme that can still cause confusion today; don't start looking for either System I, II or IV – there was no such thing. System III was quickly succeeded by System V, often abbreviated SysV. Strictly speaking it was the name for AT&T Unix, but today it's most often used for the family of operating systems derived from it (as opposed to the BSD-based ones).

In 1984 a European consortium initially known as "BISON" (for Bull, ICL, Siemens, Olivetti and Nixdorf) was formed and renamed to *X/Open* when Philips and Ericsson joined. The common goal of the founders was to promote open standards in the IT field. X/Open started creating a specification for Unix-derived operating systems with the aim of increasing interoperability of applications and lowering the cost for porting them between the various Unix derivatives. The consortium published 4 issues of its specifications as *X/Open Portability Guide* (XPG1 to 4) over the course of the next eight years.

Others also noticed that the various Unix-derived systems had diverged quite a bit – most notably between BSD-based and SysV-based ones but also among different systems of the same family. To lessen the problems of portability issues with software that were growing bigger and bigger, in 1985 an effort was made to come up with a well-defined standard that all Unix vendors should follow. This lead to POSIX, the *Portable Operating System Interface*. Compared to XPG, POSIX has a narrower scope, concentrating on the direct OS interfaces.

46.6. The "Unix Wars"

With their new System V, AT&T tried to reclaim full control over Unix again and thus to battle BSD. Promotional material like buttons and posters were made that said: "System V: Consider it standard". The BSD users responded the same way, but theirs read: "4.2 > V" (4.2BSD was the latest release at the time).

In general the camps and situation were summed up by Eric S. Raymond as "Short-hair programmers vs. long-haired ones": More business-oriented people (and companies) sided with AT&T, the more technical users and admins tended to be in favour of BSD. The rivalry between these two camps and the race for more features dominated much of the 80's. It's known as the "Unix Wars".

Initially BSD had a head start especially due to the availability of TCP/IP but newer System V releases also added interesting features of their own. When AT&T realized that they couldn't beat BSD that way a new strategy was needed. They landed a real coup when in January 1988 they announced they'd buy stakes of Sun Microsystems (Sun was the most important BSD-based vendor of the time). But not only that, they would also team up with Sun for a "Unix unification" project!

The software outcome of this was *System V Release 4* (SVR4) which integrated many BSD parts into the system. That made it possible to run BSD programs on SysV Unix by just using different paths. Equally important however were the actions of the other Unix vendors who believed that this new situation would severely threaten their business.

Seven companies (Apollo, Bull, DEC, HP, IBM, Nixdorf and Siemens) went on to form the *Open Software Foundation* (OSF) with more joining quickly until the body had more than a hundred members. OSF's goals included gaining more influence on the POSIX standardization as well as creation of *OSF/1*, a Unix system meant to compete with AT&T's and Sun's work on SVR4. Another noteworthy technology that came from OSF was the generally well-received Motif GUI specification and widget toolkit.

AT&T and Sun (supported by a couple of other companies) founded *Unix International* (UI) in response. Officially to also promote open standards but in fact for reasons of a counterbalance to the OSF. So for the second phase of the Unix Wars, the former rivalry between SysV and BSD had shifted to an inter-SysV rivalry between two influential groups.

The "Wars" finally came to an end as the participants eventually realized that the biggest threat was not the other side but in fact an outside actor: Microsoft. Unix fragmentation and infighting had allowed the Redmond-based software company to rise in influence and Windows NT was perceived a possible game-changer. Willing to work together now, in 1993 the big Unix vendors formed the *Common Open Software Environment* initiative (COSE). AT&T pulled out of the Unix business the same year, selling the Unix System Laboratories (USL) including trademarks to Novell. The latter sold the exclusive right of utilisation to X/Open.

COSE set new goals: Real interoperability and standardization of what already existed without a strategic agenda for any product group. The standardization efforts led to what later became known as the *Single Unix Specification* (SUS). A notable effort was the introduction of the *Common Desktop Environment* (CDE) that used the Motif widget set and was available for many Unix derivatives.

The success of the new initiative led to the actual merger of the former competitors OSF and UI in 1994 under the name OSF. Another merger, the "new OSF" with X/Open, eventually formed *The Open Group* which is the holder of the UNIX trademark and maintainer of the SUS to this day.

46.7. BSD Outside UCB

Commercial Unix existed for the PC platform in form of Microsoft's Xenix, IBM's PC/IX or Sun's BSD-based SunOS for the short-lived Sun386i. But when *386BSD* (also called

Jolix) was released in 1992 it meant a revolution because it was publicly available as free software under a BSD license! William and Lynne Jolitz had ported 4.3BSD Net/2 to the PC and created the missing parts (left out from Net/2 because they were AT&T-owned code).

After initially being a huge success, project development slowed down considerably and eventually came to a halt. Users of 386BSD collected bug fixes and began releasing unofficial patchkits. As disagreements between the Jolitzes and the patchkit maintainers arose, one group started work on NetBSD in march 1993. Their goal was a more open development model for a multi-platform BSD-based system they wanted to create. Independently another group of dissatisfied users founded the FreeBSD project about three months later.

Some of the people who had been deeply involved with BSD founded *Berkeley Software Design Inc.* (BSDi) in 1991. BSDi sold their 4.3BSD-based 386/BSD operating system that was partly proprietary. A license for 386/BSD (including the source) was just below $1,000 whereas an AT&T source license at that time billed at about $20,000. This caused huge irritation on the side of AT&T and in 1992 the now legendary *USL v. BSDi* lawsuit was filed.

BSDi accepted liability only for the missing files they had created and argued that the rest of their product was built from the freely distributed BSD sources. The judge agreed. Not willing to accept this, USL chose to widen the case and sue the University of California, too. They claimed that with Net/2 the university had breached the license contract, diluted USL's trademark and infringed on their copyright as well as misappropriated Unix trade secret. They asked the court for a preliminary injunction to prohibit BSDi to distribute their software until the case was decided.

In 1993 the judge denied the preliminary injunction after recognizing that USL had no legally valid copyright over 32V Unix (that BSD was derived from) and had failed to provide evidence for any obvious trade secret. Only days later the university filed its countersuit.

With SVR4 AT&T had included the TCP/IP stack that had originated at Berkeley like a duck takes to water. But when they did that, they simply stripped the UC copyright notices so it appeared as if it was their code. The original BSD license requires that credit be given to the authors and USL had carelessly ignored that. UCB demanded that USL needed to reprint all manuals with the due credit among other things.

After USL was bought by Novell, the new decision makers favoured a settlement out of court that was eventually reached in early 1994. It meant that only very few of the files needed to be omitted from the upcoming 4.4BSD-lite release and several modified to show proper copyright notices. USL on the other hand agreed to not file future suits against users of the 4.4BSD-lite release.

46.8. GNU, MINIX and Linux

Irritated about the continuing proprietarization of Unix, in 1983 Richard Matthew Stallman (RMS) had started the GNU project ("GNU's not Unix"). The goal was to create an operating system that was portable so it would not die when an ageing hardware platform was retired. It also had to be freely distributable for everyone – and the source should always be available to study and modify.

The project chose to be Unix-compatible because that OS was already quite popular and RMS believed that it would be very beneficial to the project if a lot of software was already available for it. Another plus was Unix' modularity: As the system consisted of many small tools, those could be replaced one by one rather than requiring one huge effort.

RMS started work on the GNU Emacs editor right away (initially based on the James Gosling version but effectively replacing most of the code) and later did the first version of the extremely popular *GNU Compiler Collection* (GCC, which originally stood for *GNU C Compiler*). Other critical components like the GNU C Library (glibc) and the BASH shell were created in addition to adoptions of many (often enhanced) Unix tools.

In 1985 Stallman also founded the *Free Software Foundation* (FSF), a non-profit corporation. It was formed to help the GNU project, to provide stewardship of the GPL family of licenses and to promote free software in general.

GNU is supposed to be a complete operating system. But there was – and in fact still is – a problem: While the various tools and libraries made good progress pretty soon, the project's kernel didn't. They had chosen a microkernel design and while at the time it looked like that would be the future of operating system kernels, it also brought with it lots of complexity. GNU's HURD is still not considered production-ready today!

Very disappointed that Unix V7 could no longer be used at his university due to the changes in the license, Andrew Tanenbaum decided that he would write a small Unix-like operating system specifically for teaching. In 1987 Tanenbaum published his textbook *Operating Systems: Design and Implementation* and released *MINIX 1.0*.

The first version of MINIX was system-call compatible with Unix V7. One major point where Tanenbaum chose to go a different way was implementing a microkernel architecture (which would later lead to the so-called *Tanenbaum–Torvalds debate* over Linux's monolithic kernel approach). Version 2 of MINIX added POSIX.1 compliance and a TCP/IP stack among other things.

Initially not free software (Tanenbaum's publisher did not agree with publishing code that could be copied freely), in 2000 MINIX 2 would be re-licensed under a BSD license. The first version of MINIX 3 was released in 2005 and while it's still meant to be useful in education, a new goal of providing a fault-tolerant system meant for high availability environments was set.

In 1991 a Finnish Computer Science student posted a now legendary announcement to the MINIX mailing list: He told the world about a free OS for i386 that he was creating – "just a hobby, won't be big and professional like gnu". Linus Torvalds was not only aware of GNU, he also mentioned running GNU tools on his new kernel.

As he continued to improve his kernel, others started playing with it. Since there was a kernel now both free *and* working, it was a logical thing to combine it with GNU – and that was what people did. Over the course of 1993 several early *Linux distributions* came into existence, bundling the kernel with GNU software to create a usable operating system from these components.

At this time the USL v. BSDi case was still ongoing and the legal uncertainty of BSD massively helped increase the popularity of GNU/Linux. As a system written from scratch it was deemed free from potential legal trouble and as such pretty appealing for personal use as well as for business.

To not keep quiet about one more oddity, the so-called *SCO-Linux controversies* need to be mentioned. *Caldera*, a company that was one of the early Linux distributors, renamed itself *The SCO Group* after acquiring the Unix part of SCO (which in turn had acquired UnixWare from Novell). They then started attacking Linux with various claims about Unix code in Linux and sued several companies like IBM, Red Hat and Novell.

This amusing / sad (depending entirely on your sense of humour) story is still ongoing. The SCO Group has failed to provide any evidence of copyright infringement and went bankrupt over the process. After selling their active business they were renamed once more and now bear the name *The TSG Group*, their only remaining "property" being the ongoing lawsuits...

46.9. Open Source vs. Closed Source Unix-likes

During the time when commercial Unix was booming, few people would have even deemed it possible that the leading Unix systems could ever be dethroned by one of those Open Source upstarts. Exactly this is what did happen however.

The once pretty important IRIX barely made it into our millennium: In 2001, SGI announced to shut in down and end support in 2013. HP-UX is not entirely dead, yet, but HP bet on the wrong horse. Their Unix variant is available for their old PA-RISC architecture as well as the Intel Itanium that succeeded it. The latter platform has commercially failed however and while that has been an open secret for a while, Intel announced the formal discontinuation in 2019.

Former top dog Sun felt the pressure mostly from Linux for a while before adopting a strategy of Open Sourcing their operating system as OpenSolaris from 2005 onward. Sun decided to completely open everything up as far as possible, including their perceived crown jewels like ZFS and DTrace. In 2010 then Oracle bought Sun and closed down Solaris again – but did not show much effort to keep the system from dying. In late 2016 Oracle cancelled the upcoming Solaris 12 and put Solaris 11 in maintenance mode. Massive layoffs in the following years crippled the team so much that no sane person believes in a real future for Solaris anymore.

AT&T Unix, Xenix, UnixWare and SCO Unix / SCO OpenServer ended up in Xinuos OpenServer. The latest release is mixed source (partly open and partly closed) and based on FreeBSD. OpenServer is still alive in the enterprise sector but its market share is probably not big enough to be visible easily.

The only vendor whose proprietary Unix seems to still have a somewhat bright future is IBM. AIX is alive and officially still a somewhat popular option but not even IBM denies that many of their customers prefer Linux over it. And market analysts note that while in October 2018 IBM announced to take over enterprise Linux distributor Red Hat for 34 billion (!) dollars US, they surely don't spend such an amount of money on AIX development. So it's not hard to read the signs on the wall for another commercial Unix, either.

Today Linux is everywhere. In about 2004 it took the lead on the Top-500 supercomputers – and in 2018 it reached a share of 100% (!) in supercomputing. In the mobile world, Linux-derived Android has a market share of above 87% in 2020, leaving about 13% for iOS, the only remaining serious contender. It's used in space, too: Since 2013 it powers the computers on the ISS.

And when it comes to servers, the only commercial operating system that still competes is Windows server. Since Linux is free, precise figures are not known. However even on Microsoft's own Azure Cloud the count of Linux installations has surpassed that of Windows instances – and nobody (including Microsoft) is anticipating that this will change again.

Speaking of the big Redmond-based company that declared Linux "cancer" and fought it hard a decade ago... Today Microsoft is deeply involved with the further development of Linux, giving large sums of money to be a member in the important organizations, paying developers pushing Linux forward and even including more and more of it in their "Windows subsystem for Linux".

In all fields except for the PC desktop we've reached or passed the point where there's a new de facto monopoly. And while it's certainly a good thing that this new monopolist is an Open Source one, a monopoly is never a good thing, not even when it's GNU/Linux.

Despite this massive onslaught of Linux draining basically from all of the commercial vendors, FreeBSD has not seen a lot of growth but maintained a pretty stable number of developers. With certain events in Linux land there has been renewed interest in FreeBSD and it is starting to become more visible again.

For some years now the TOR project for example has been asking people to setup more BSD-based exit nodes to diversify their network for a healthier infrastructure that does not rely mostly on one operating system.

What will be the next section in a book like this when we look back at our time in a decade or two? Will renewed enthusiasm for FreeBSD and other Open Source operating systems make a noticeable impact worth of being mentioned in history? Only time will tell.

47. Open Source OS Family Matters

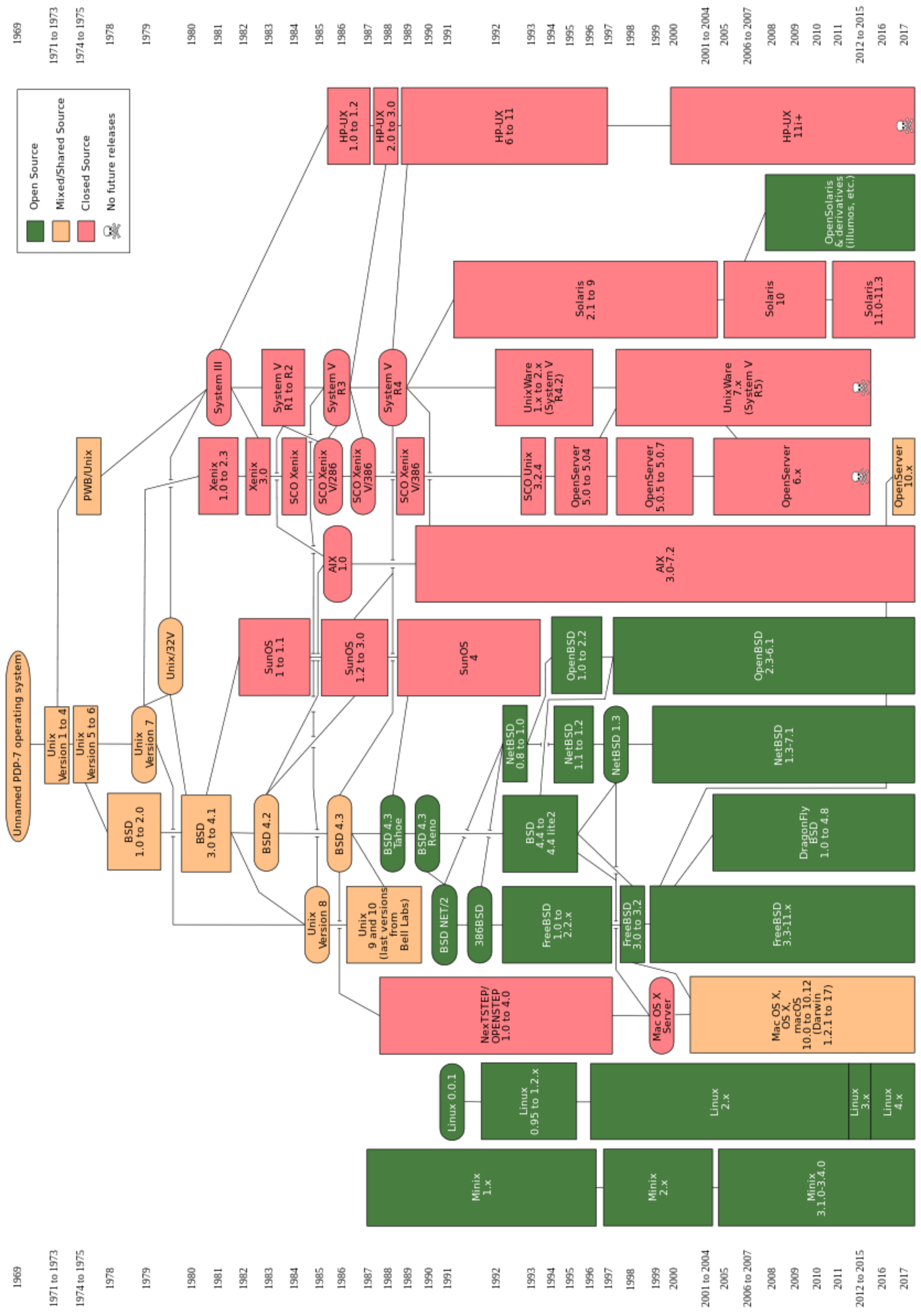
This chapter is meant to help newcomers to BSD land get oriented. It's another background chapter; if you've already settled on FreeBSD and know (or don't care) who your neighbours are, you can of course skip it.

FooBSD, BarBSD, BazBSD? What's the thing with the various BSDs? Are these distributions? Don't worry, you survived the Linux distro jungle, you'll be fine over here, too. But while BSD land is quite a bit smaller and definitely not as crowded, things are working a little differently. There are multiple BSDs – but they are *not* "BSD distributions"! You'll soon know why.

Whenever there is more than a one of a kind operating system that's truly unlike any other, the similar ones are grouped into the same *operating system family*. For example there is the once popular *DOS family*; some of its members being MS-DOS, DR-DOS, ROM-DOS and FreeDOS.

Whether there actually is a *Windows family* or an *NT family* is somewhat controversial. Some people say "Windows family" and refer to the various versions and releases of Windows. In the author's opinion that's rather the Microsoft product line. Others argue that there is only Microsoft Windows and as such there is no family. Taking ReactOS as an Open Source Windows NT clone into consideration however, one could speak of a Windows OS family (consisting of two systems).

Either way our topic is Unix – and that has a huge family tree. It's in fact big enough that it comprises of two subordinate "families": The *System V* line and the *BSD* line (the background of both was covered in some more depth in Chapter 1). See the next page for a simplified diagram of the broader Unix family to get an idea (the complete tree including lesser known systems would be much larger).



[Source: Created by Eraserhead1, Infinity0 (Own work: CC-BY-SA-3.0, GFDL) - Image:Unix history-simple.svg, Levenez Unix History Diagram, Information on the history of IBM's AIX on ibm.com, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=7091465]

47.1. What are UNIX, Unix, *nix and Unix-likes?

Both FreeBSD and Linux are Unix-like operating systems. Linux was designed to behave like Unix but written from scratch. With GNU even the name clearly states that it's not Unix – and so it's not surprising that a GNU/Linux system is defined as a Unix-like. FreeBSD on the other hand is derived directly from AT&T Unix (see chapter 1). But why is it also Unix-like and not simply Unix? Weren't the original BSD releases often referred to as "BSD Unix"? And why do you sometimes read UNIX in all caps?

What sounds a bit confusing is actually pretty simple: Today UNIX does not mean a specific operating system but rather a trademark of The Open Group. Operating systems that adhere to the SUS (Single Unix Specification) can get certified. Systems that are certified, *are* "Unix" with no further requirements to be met. Apple got their macOS certified for example. So that operating system is Unix even though it is technically and historically an offspring of the BSD line of systems.

FreeBSD on the other hand is *not* certified. It might be possible that it passes the certification process if put to the test. To get certified involves quite a bit of money handed over to The Open Group, however. And as FreeBSD is, well, a *free* operating system, the project never had the spare money for a certification effort (there are always more important things to work on). For that reason FreeBSD is not (a) Unix as far as the legal status is concerned. To underline its heritage some people do point out that it is a "genetic Unix", though!

47.2. Major Differences Between Linux and *BSD

47.2.1. Complete OS vs. Distribution

Operating systems comprise of two distinct parts: The kernel (which is responsible for managing the hardware and has full privilege) and what is called *userland*, the various programs that come with the OS and use the kernel.

The first and foremost difference between Linux and any BSD is that the latter is a *complete operating system* whereas Linux is, strictly speaking, just a kernel. Bundled with the necessary userland parts – e.g. as GNU/Linux which is commonly just called "Linux" as well – it works like a complete operating system, too. So what's the matter really?

For one thing Linux requires a distributor to combine the kernel with some userland tools to form an operating system useful for the average user. You disagree with goals of all the existing Linux distributions and want something that fits your special needs? Given you have enough motivation, technical knowledge and resources, you are free to create another one, adding one more Linux flavour to the list. Whether you want to use a different package manager, ship some special tools by default or really just want to modify the wallpapers - if you give it a name and decide to share, there's one more distribution. Some people might argue that with minor changes it's only a remix or something, but a remixed distributed is distribution.

In BSD land it's very much different. You can do the same things, of course: Use a different means of package management, replace some system tools and so on. But that wouldn't be a distribution but rather a new operating system of the BSD family and e.g. a FreeBSD-derivative. This is due to the *whole system* approach that the BSDs follow. There is no single "BSD kernel" as there is the Linux kernel. OpenBSD's kernel is *vastly* different from FreeBSD's which is very much different from DragonFly's – even though DragonFly BSD began as a fork of FreeBSD!

To really understand why BSD people emphasise this so much let's resort to a somewhat brutal comparison: The BSDs are whole operating systems whereas a Linux distribution is "a kernel plus a bunch of packages". This may sound disrespectful towards Linux and even like a superficial argument to set BSD apart but it isn't. The difference between the two models is actually pretty much user-visible and you will often hear people say that a BSD system "feels much more integrated" due to the holistic approach.

Here's an example of one of the benefits of doing things like Linux does it: Many of the individual tools are developed by separate groups and projects. As such a developer you are relatively free to play with adding new features and changing things in your program. Being a responsible programmer you are not going to break things for others deliberately, but it's not your duty to ensure that the new version works perfectly fine on say, Gentoo RISC-V. If you break that platform somebody is probably going to let you know or to send in a patch but it really is your decision how many extra miles you want to go. It's the downstream consumer's (i.e. distributor's) responsibility to notice the breakage and deal with it. This allows for much faster innovation among other things.

Let's assume that Linux distributors do a great job: They take the kernel, carefully select the other software they use and probably patch that to integrate well into the system. There still is a lot going on beyond their reach (if you're not Red Hat that is)! Maybe there was this nifty new feature added to the kernel but you cannot use it, yet – you probably have to wait until it's supported by glibc. And then you might need to wait until even the more conservative (but influential) distributions finally ship a version of glibc that would support it. Before that happens many projects will not bother to add support for the new feature to their programs. If you do not have the resources to maintain a fork or heavily patched (and tested!) downstream package, you're out of luck. You're out of luck, too, if one of the projects that your use case depends on simply does not want to support that feature. In the latter case an interesting feature might actually never land and will eventually be thrown out of the kernel again.

With the BSDs it works the opposite way. A new feature was added to the kernel in the development branch? If it's in the next release you can be quite sure that a good part of the userland will use the new feature – and more importantly: It will do so in a consistent way. At the same time you as the developer making the required changes will want to make sure to *not* break RISC-V, because otherwise CI will fail and a light will turn red. This means that the BSDs have much more direct control over their complete system allowing e.g. for a change to *ls* after only an internal discussion and without having to go through a long debate on the mailing list where you discuss issues related to coreutils over at the GNU project. Just like the kernel, FreeBSD's ls is not the same as OpenBSD's and each project can make adjustments that makes it fit perfectly into their system but might have little chance of getting accepted if discussed in a cross-OS discussion.

Another effect of this is that building e.g. FreeBSD is extremely easy to do. You probably know the great *Linux From Scratch* book. There is no such equivalent for FreeBSD – and if there were it would be a couple of pages at most. You can literally build the full operating system (including the kernel, system compiler, all libraries and tools) by issuing "make -C /usr/src buildworld buildkernel". Yes, that's all there is to it! There's no need to build any packages at all, the *Makefile* in the source tree does it all for you.

47.2.2. Updating

Updating packages on any Linux distribution is nothing that people are generally afraid of. Yes, I broke my main workstation like four times while I was using Arch Linux. Three of the four times it was my fault as I just went ahead and did the update without reading about the respective "manual intervention" requirement that Arch informs about on its web page. The fourth' time it was actually a problem with one of the packages that could be fixed by downgrading and later attempting the update again. But that's not a bad record for a bleeding-edge rolling-release distribution in several years.

What people experienced with Linux do have a bit more respect for however is distribution upgrades. I've had upgrades ruin a couple of Ubuntu installations beyond repair (i.e. beyond the point where I deemed a fresh re-install to be more practical). With CentOS for example you're expected to do a fresh install instead of a major upgrade in the first place – there's in fact no supported way to turn a CentOS 7.x box into CentOS 8.x for example. There is one for RHEL, but I've never done that. With other distributions that I've used it's also a mixed bag, sometimes closer to Russian Roulette, sometimes to "it will probably be fine".

FreeBSD plays in an entirely different league in this regard. I manage servers that have been installed as FreeBSD 5.x, were using customized kernels and software with manually picked compile-time options for years and happily continue to serve their purpose as 12.2 with a GENERIC kernel and stock packages today. They usually survived several hardware upgrades, too. This is not a story that will astonish any fellow FreeBSD user, it's more the common case than the exception. This is another benefit of the holistic system approach: While a Linux distribution upgrade gets fairly complicated because of the nearly endless possible combination of packages that might be installed, the FreeBSD base system for each installation of the same version and revision is the same (assuming the same architecture).

When it comes to the actual operating system on BSD it almost doesn't matter what third party packages are installed or not. No rule without exception: There's some special cases like e.g. additional kernel modules installed via packages and included in the user-defined system configuration. This could break your graphical desktop for example until you update the packages, too. Also some applications might have been linked against older versions of system libraries and won't work after the update until you reinstall the packages, too. What about packages that cannot be replaced because you specifically require an old version (think of that customer who's throwing money at you for keeping his darn old PHP 5.6 alive even though everybody knows you should really get rid of that)? FreeBSD provides *compat* packages down to FreeBSD 4, providing system libraries from those releases and enabling you to continue running old software if you really must (even if you only have the binary).

Changes in configuration do happen. However FreeBSD prefers to do the more invasive changes incrementally and not all at once. After performing an update to a new major release chances are that you will receive warnings during system startup, notifying you of deprecated options. If you read them and take action, you'll be fine when those options transition from deprecated to removed in a future release.

Unlike with Linux it's also not uncommon to update *from source* rather than *in binary*. If you customized your kernel, deliberately left out parts of the userland (e.g. because they didn't meet your licensing requirements) or something like that you can still upgrade the system pretty easy by building it from source. For DragonFly for example upgrading from source is the only supported way to get to a newer release without re-installing.

47.2.3. Documentation and Manpages

Manpages have been part of Unix for a long time and so both FreeBSD and Linux inherited them. The GNU project came up with the more advanced *info pages* which support hyperlinks to other pages for example. In practise, info pages failed to succeed manpages – they are heavily used within the GNU project but have not seen overwhelming adoption outside of it.

Since with most Linux distributions GNU tools are a central part of the operating system, you will have to use info in many cases when you look for documentation. Usually there's a manpage for the same tool, too – and it's not too uncommon that the information you seek is in one of them but not the other. If you're really lucky, the two documentation sources even disagree as they are out of sync, written by different authors, etc...

Also the manpages for various programs are somewhat of a mixed bag: Some of them pretty good and useful, some severely lacking and others next to or entirely non-existent. As a result of that you will find that especially the younger generation of Linux users does know about manpages and info pages but will generally prefer to search on the net if they need to figure out something.

All of the BSDs generally value good and accurate documentation very highly. A mistake on a manpage is a very serious bug as they are meant to be the authoritative documentation for a program, kernel interface, configuration file (yes, really!) and so on. A person who maintains documentation is not “merely” a *docs committer* but as such has exactly the same rights and esteem as a source or ports committer.

Of course documentation is never perfect and there are dark and dusty corners of FreeBSD as well. But in general if you compare the manpages for one tool on Linux to that of FreeBSD’s equivalent, you’ll see a huge difference. If you are new to FreeBSD, by all means do give manpages another chance should you be one of the people who mostly disregarded them on Linux! There are developers who have even switched to FreeBSD for the reason of the much better documentation available there.

The same thing is true for other sources of official documentation. The FreeBSD project prides itself in its *handbook* for example. It is a great source of information for the newcomer and the long-time user alike. There are efforts to provide similar material for various Linux distributions but most of these simply fall short of reaching the same height of quality. One exception here is Gentoo which provides an excellent handbook (but then many fundamental design decisions of Gentoo were influenced by FreeBSD).

Do not fall into the trap of heading to the FreeBSD wiki when looking for information, though! While e.g. the Arch Linux wiki is a great source of information for Arch users, FreeBSD’s wiki is actually meant for the developers. A lot of the pages are horribly out of date and when you look at them you are supposed to notice this. It’s more of a place where developers put some notes for themselves or fellow developers without a lot of explanation for the common user. You might find what you are looking for in the wiki, but be prepared for material in pretty raw shape.

47.2.4. Licensing

While Linux has embraced the GPL license, FreeBSD is committed to the BSD license. The former is a popular option for Open Source that belongs in the so-called *copyleft* family of licenses, the latter is a premier member of the family of permissive or “copycenter” licenses.

There have been fierce wars going on between supporters of both camps about what is "the best" license. When looking at the matter calmly as in *sine ira et studio*, both families have different goals while both are supporting Open Source.

The idea of copyleft is that code should remain *free*. So the licenses allow free use but explicitly force any vendor using it to publish changed code under the same license again. Supporters of this model deem this necessary. You will hear the familiar stories of greedy corporations taking free code and making a lot of money from it while making it proprietary and thus essentially ripping of the Open Source community.

While individuals in favour of permissive licenses don't like the closing down Open Source code either, they argue that the possibility of such an event is the price for true freedom ("no strings attached") and they are willing to pay it. Also they insist on a small detail that copyleft advocates do not like to stress too much: With a permissive license there's no closing down of the *original* licensed code. It's only possible to a closed-sourcederivative thereof! So it's more of a theoretical loss to the world than an actual one.

There's a point to both positions and "who's right" depends on the particular case. However there are concrete consequences of license choices. For example Red Hat is paying specialists to maintain old Linux kernel versions for their enterprise distribution. They are forced to keep the source open, allowing other actors like Oracle or the forming community of Rocky Linux to use their work free of charge. If Linux was available under a permissive license they could just keep the source of their work closed.

On the other hand Linux is having a really hard time with ZFS (involving lots of drama). The code for ZFS is Open Source and in fact even under a *copyleft* license – but under one that most people deem to be *incompatible* with Linux' copyleft GPL and thus falling on their own sword!

Another thing to consider is license complexity. If you feel like reading pages and pages of legalese, go ahead an read the GPLv3 if you've never done so. Even if you don't feel like it you probably should do this at least once. It's less important when just using Open Source software yourself but still pretty enlightening. Should you consider to base some project on such software you no longer should but actually *have to* read licenses – and to make really sure you understand the implications. You probably need to pay a lawyer, too, just to be save and have explained to you what the various points actually mean.

The GPL license has grown with every revision and is quite far away from being simple. The BSD license on the other hand has been cut down with every revision: The 4-clause *original BSD license* included the requirement to mention the organization that the software originated with in advertisements. This was dropped for the 3-clause *Revised BSD license*. The version preferred by FreeBSD – the 2-clause *Simplified BSD license* – also dropped the prohibition of using the names of the organization and contributors e.g. to promote products derived from code under that license.

FreeBSD being permissively licensed means that you can use the code and rest assured that there are no complicated legal tangles at all. The license basically means: Honour the authors by keeping the copyright notice when you distribute in either source or binary and claim full liability in case something doesn't work as you expect it. Plain, simple and to the point where you can spend your time on development rather than on law studies.

47.3. FreeBSD and the BSD Family as a Whole

FreeBSD is the biggest and most popular of the BSD projects. Being developed as a general purpose OS (with a traditionally strong focus on servers however), it has the broadest target audience, largest user base, highest count of applications available for it and so on. It is also the most visible of the BSDs both on the net and on conferences and the best funded one, too.

But that doesn't mean that it's the best fit for each and every use case. There are quite a few people who have a different favourite BSD and usually for good technical reasons (not that "I like it best!" wasn't a valid reason, too).

The relationship between the various BSD projects is somewhat different from that between common Linux distros. Since the BSD world is a lot smaller, all of the projects have their niche and certain areas where they excel. Therefore there is not so much actual competition at all. On the contrary: BSD projects tend to share code pretty often and it's not uncommon that people are quite familiar with more than one OS.

There is disagreement about fundamental things of course and since the BSD community is human beings there are always some folks with a somewhat sharp tongue around. You will hear things like FreeBSD being "the Ubuntu of *BSD", usually in a disrespectful tone. Somebody else probably states something like "I really don't see the point in NetBSD". But all things taken into account, people usually behave very decently.

In fact what may look like a hostile comment to the outsider might actually be just some humour that escapes you. Keep in mind that many of the BSD developers know each other even across the "borders" of the various projects. There's for example a conference talk given some time ago which goes by the title: "My BSD sucks less than yours". An OpenBSD developer and a FreeBSD developer discuss the strong points and weaknesses of their OS in a very entertaining way.

In general however there is this statement that a lot of people hold up: **All BSDs were created equal**. I'm a very firm believer in that stance. There's no single "best" food, book, weather. No "one size fit's all" clothes and certainly no such operating system, either. In theory it would be best if you tried out all of them and then made an educated choice. In practise however there are always limiting factors like time, enthusiasm, abilities. To narrow down the ones that might be of interest to you, I'll give a little overview of the other BSDs next.

47.4. Close relatives: Other BSDs

While FreeBSD as an operating system is a good all-rounder, nobody is in doubt that it's surpassed by other BSDs in certain areas that those focus and put their main attention to.

47.4.1. OpenBSD

In a 2016 blog post for BSD newcomers I introduced OpenBSD like this:

"If you choose OpenBSD, there's a lot of really cool and modern features... *that you will have to do without*! Oh, don't get me wrong: That can be a *great* thing!"

I still think that for understanding OpenBSD it's important to really get right from the start that it is a very special OS that happily goes a way you as an outsider wouldn't expect (or might even be shocked about). First and foremost OpenBSD is about security and correctness. Also it's built to work for its developers, meeting their requirements. If it meets yours, too, you're invited to use it as you see fit. If it doesn't… Well, too bad, use another one! OpenBSD developers won't add something to the system just because you think it would be nice.

Sounds unusual? It is. OpenBSD people live up to their principles and will not sacrifice them for a possibly slightly larger user base. Here's some examples that should convince you that my claims of these guys really meaning it are true:

- OpenBSD comes with HyperThreading support disabled by default
- The OS used to support loadable kernel modules – that feature is gone entirely
- The "year 2038 issue" (32-bit Unix time rollover) was solved as early as 2014 in all supported platforms
- The team maintains their own fork of Xorg (named Xenocara)
- For years and years the team has maintained Apache 1.3 because they did not like the more restrictive license for 2.x. After a brief interlude with Nginx they scrapped well-known web servers and wrote their own from scratch

As you can see, OpenBSD chooses to "play save" even when it hurts performance (HyperThreading) or makes administrating the system less comfortable. And they are not afraid to do things themselves if there's nothing out there that meets their expectations.

One such expectation is that something can be made to adhere one of the important principles of the project that goes: "Secure by default". With other operating systems you have to tune them for security – OpenBSD is meant to be fine by default. You cannot forget to enable the firewall on a new installation like on FreeBSD or Linux. Why? Because it's enabled in the default configuration! In contrast to FreeBSD, it does not like to provide as many knobs as possible and won't let you do some unsecure things easily (e.g. running the base system web server not chrooted is not supported).

OpenBSD takes a very clear and strict policy on licenses. While there is some GPL2-licensed software in the OS, the license deemed barely acceptable but unavoidable for the time being. GPL3 is rejected completely (of course you can install software covered by that license from packages, though!). The preferred license is ISC which is functionally equivalent to Simplified BSD but put differently. The team will not accept NDAs under any circumstance. If a vendor chooses not to provide documentation without signing an NDA, OpenBSD chooses to not support that hardware. Period.

In general it's "Do it right or simply don't do it". A lot of effort is put into code auditing and improvement. Excellent documentation is considered crucial. To make the audits easier to do, developers try hard to maintain a tidy system. They will rip out code (removing unused functionality) as happily as developers of other projects add new things. They do not oppose new features if they make the system more useful but think about the implications first and usually draw a line, declaring what possible features are in fact unwanted (because they'd add more complexity than good).

OpenBSD runs on a considerable number of architectures and the team deliberately keeps some old ones going, too. This has helped many times to catch edge-cases early that would have resulted in unnoticed bugs if tested on x86 hardware only. For that reason the project builds the OS on real hardware even for the more exotic platforms and does not go the easy route using emulators.

While the OpenBSD project is quite a bit smaller than FreeBSD, it has sparked an impressive amount of technology. Tools like *OpenSSH*, *sudo* or *tmux* which are daily-drivers for many of us have their roots in OpenBSD. *OpenNTPD*, *OpenSMPD*, *relayd*, *spamd* and many more come from it as well. The same is true for the awesome Pf firewall that has been ported to FreeBSD and NetBSD as well (but both ports have diverged significantly and the actual upstream development is on OpenBSD).

A notable field where OpenBSD also really shines is exploit mitigations. The developers don't buy the illusion of a bug-free system even though they try hard to get as close to that ideal as possible. As a consequence they come up with new ways to mitigate common (and ideally even future) exploit techniques. OpenBSD pioneered technology like system-wide stack protection, ASLR, W^X (policy to disallow memory being writeable and executable at the same time), KARL (randomized kernel re-linking during startup so that each time the system boots the kernel addresses are different) and so on.

While all of this may sound somewhat "extremist" to many people it's simply a very commendable and appealing pure form of a *no compromise* stance to others. Only you know if that's for you or not.

47.4.2. NetBSD

47.4.3. DragonFly BSD

47.5. Distant Relatives: Other Unix-likes

47.5.1. OpenSolaris / illumos

47.5.2. Minix

47.5.3. Linux

47.6. FreeBSD-derived Systems (general)

47.6.1. HardenedBSD

47.6.2. MidnightBSD

47.6.3. mfsBSD

47.7. FreeBSD-derived Systems (desktop-oriented)

47.7.1. GhostBSD

47.7.2. NomadBSD

47.7.3. helloSystem

47.7.4. (Some) Historic Derivatives

Desktop BSD

PC-BSD / TrueOS

Project Trident

FuryBSD

48. FreeBSD Culture and Community

48.1. An Overview of the FreeBSD Project

48.1.1. Architectures and Tiers

48.1.2. FreeBSD Branches and Support

48.1.3. System Components

48.1.4. The mascot: Beastie

48.2. Values of FreeBSD

48.2.1. Code of Conduct

48.2.2. Permissive Licenses

48.2.3. Cathedral, not Bazaar!

48.3. Project Structure

48.3.1. Governance

48.3.2. The Foundation's Role

48.4. The Community

48.5. Known Deficiencies – where FreeBSD Comes up Short

Afterword

Placeholder for afterword

Appendix

Literature

Placeholder for literature

List of Contributors

The following people (in alphabetic order) helped hammer this book into shape:

P. C. sent a mail with lots of small corrections for the history chapter and the first half of the family matters chapter already written at that point in time. He also made some suggestions that helped to make the book more useful.

F. F. mailed in with additional info for the family matters chapter and suggestions regarding desktop FreeBSD.

Nathanael A. Hoyle pointed me at multiple spelling mistakes and made suggestions regarding the structure of the history chapter.

O. K. kindly provided a thorough correction of the history chapter including a lot of helpful hints. Special thanks to O. for patiently giving explanations when my English failed me time and time again!

About the Author

Placeholder for author