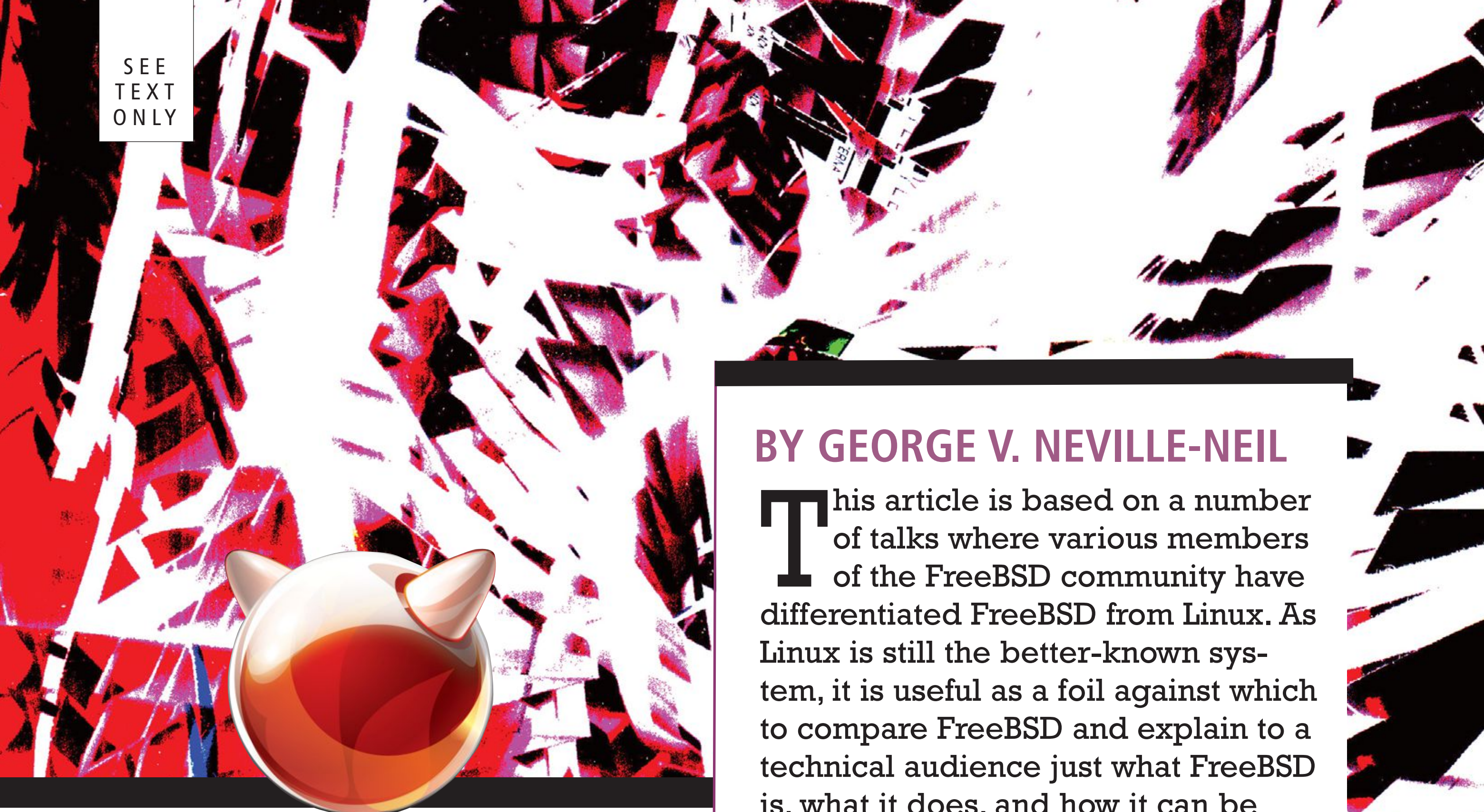

SEE TEXT ONLY

# FreeBSD is Not a LINUX DISTRO

BY GEORGE V. NEVILLE-NEIL

This article is based on a number of talks where various members of the FreeBSD community have differentiated FreeBSD from Linux. As Linux is still the better-known system, it is useful as a foil against which to compare FreeBSD and explain to a technical audience just what FreeBSD is, what it does, and how it can be applied to build a variety of systems that require an operating system. Since this article appears in the *FreeBSD Journal*, I expect readers are familiar with FreeBSD, and so I have organized the article as a set of talking points that can be referenced when you're trying to explain FreeBSD to colleagues or company management.

## FreeBSD Is a Complete System

FreeBSD is a complete, open-source operating system that includes all the sources and tools to enable the consumer of the operating system to rebuild the entire system from the supplied sources. The completeness of the system is one of the key differentiators between FreeBSD and Linux. Linux is actually just the operating system kernel—a large piece of code—but not a complete system. To build a Linux system, a number of tools, packages, and libraries need to be installed alongside the sources. With a FreeBSD system, users can rebuild their entire systems the moment after FreeBSD is installed onto their computers. The \emph{self hosting} aspect of the FreeBSD system should not be overlooked because it is a core part of the philosophy of FreeBSD, the empowerment of the user. A

complete FreeBSD system comes not only with the source code, but also with extensive documentation on all aspects of the system, including the built-in commands, library, and kernel APIs. The goal is to make users of FreeBSD as productive as possible once the system has been installed.

The operating system is only a small component of most modern systems. The majority of the software involved in building and deploying any significant system is not a component of the operating system, but instead is part of the user space software that runs on top of the operating system. FreeBSD supports third-party software via the ports and packages system. Most users will interact only with the packages system, which contains pre-packaged versions of software such as web servers and browsers, high-performance math libraries, language compilers and interpreters, and nearly any other open-source tool that is actively being maintained. Packages are built from the ports system, which is a large, hierarchically organized collection of Makefiles that knows from whence to retrieve software and how to build that software so that it runs on FreeBSD. The current set of ports encompasses more than 24,000 software packages. Consumers of FreeBSD can add their own software packages to the ports system as a way of building their own customized, installable version of FreeBSD.

## Who Is the End User?

A Linux distro is a way of packaging an operating system kernel along with a set of applications into a system that can be installed by an end user. For many years Linux distros have been targeted for either desktop or server installations. A common example is the Ubuntu distro which comes in two configurations, one for servers and the other for desktops.

From FreeBSD's point of view, the end user is most often an engineer or software developer who will use FreeBSD to create something new, based around the FreeBSD system. Consumers of FreeBSD come in all shapes and sizes, and over the years, other open-source systems have been developed around FreeBSD, which are then consumed for particular purposes, such as pfSense for firewalls, FreeNAS for storage appliances, and TrueOS (formerly PC-BSD) for desktops and laptops. It is these systems that appear closer to a "distro" than FreeBSD itself.

Alongside these open-source systems, many companies have taken all, or part, of FreeBSD and used it as the basis of their own products. Apple, NetApp, Isilon, Juniper, and Sony, to name only a few, have used FreeBSD as the basis of successful products in desktop, mobile, storage, networking, and gaming. Each of these engineering consumers has been able to treat FreeBSD as a collection of libraries that can be combined to create an operating system to support their own products, and in this case, the end user isn't a consumer per se, but rather an engineer who is building a product for an end user.

## Why Do People Use FreeBSD?

What motivates someone to use FreeBSD? There are five key reasons that people decide to use FreeBSD, and these are: our history of innovation, great tools, mature release model, documentation, and business-friendly license, all of which differentiate FreeBSD from a Linux distro.

FreeBSD is a child of the original Berkeley Software Distribution, the Unix variant that was developed at the University of California at Berkeley during the 1970s and 1980s. Virtual Memory, the Fast File System, Sockets API, and TCP/IP are all innovations that occurred in the original BSD system. FreeBSD has continued that tradition of innovation, producing high-performance software that was an early adopter of multicore systems, 10G, 40G, and 100G Ethernet, and the LLVM compiler suite.

LLVM is an example of how FreeBSD works with and integrates great tools. The LLVM compiler suite has led to an order of magnitude increase in interesting compiler technologies, technologies that just were not possible with the GNU compilers. LLVM has been particularly strong in the use of new compiler techniques to enhance security, leading to both research and deployment of more secure systems. With LLVM as our default compiler, all of these innovations have appeared in FreeBSD before appearing in other operating systems.

Anyone who has tried to produce a series of products on top of a Linux distro has run up against the inconsistencies in the Linux release model. Between one minor release and another, the Linux kernel APIs are not stable; in fact, Linux is committed to not having stable kernel APIs. That may be fine for those who want to only run the tip of the tree, but it is a disaster if you are a company trying to field products and upgrade them over time.

The FreeBSD release model requires that long-term branches, those numbered 9.x, 10.x, 11.x, maintain kernel API stability throughout their lifetimes. This commitment to stability means that products built using FreeBSD 10 can continue to be upgraded until the end of life of the 10 branch. New features and bug fixes will appear throughout the lifetime of the branch, but they will only appear if they do not change the way in which already established APIs work. These concepts are often wrapped up in the idea of the principle of least astonishment (POLA), whereby we try to not surprise our consumers, and where we clearly delineate the circumstances under which breaking changes are allowed to occur, such as across major release branches.

FreeBSD's documentation is second to none, and is, in fact, often referenced by those who are looking for generic information about other Unix systems and how they work. The manual pages not only cover the traditional areas of the system, such as commands (Section 1), system calls (Section 2), and libraries (Section 3), but also the kernel APIs themselves are documented in Section 9 of the manual pages. *The FreeBSD Handbook* acts as an overarching reference document to the system as a whole and covers higher-level topics relating to systems administration and software development.

Open-source systems are not just software; they also encompass a philosophy, most often described in the license they choose to use. The two-clause BSD license used by FreeBSD embodies the FreeBSD philosophy, which might be summed up as: use our code and don't sue us. The license is short, a mere 200 words, and is easy to understand. The GPLv2 used in Linux is over 2,900 words, and requires a lawyer to understand all of the ramifications of working with it.

## Community Organization

Each open-source project has its own unique form of organization. Linux adheres to the "big high stomper" model, whereby Linus and his various lieutenants control access to the source tree. Linus is the leader because he started the project, and his lieutenants maintain their position in the project because they were chosen by Linus.

The FreeBSD Project is organized by and for those who commit to the source tree, and has no single owner or dictator. A committer is simply a person with commit access to the central source code server. Three types of commit bits exist, one each for documentation, port, and the source tree. There is no wall between these three types of access; they exist simply to differentiate the areas in which people might be doing most of their work. A source bit is granted to those who are working on the built source of the system either in the kernel or its associated tools. Documentation bits are held by those who are working on manual pages as well as projects such as the *FreeBSD Handbook*, the living document that describes FreeBSD as a whole and which goes far beyond what is contained in the manual pages. The ports system is maintained by ports committers, who have ports bits. Any one person may have any or all of these bits, and, indeed, those who have worked on FreeBSD for a number of years often wind up with all three of these commit bits.

The process of being granted a commit bit is relatively straightforward:

**1** Alice interacts with the project, often by submitting patches to the system sources, ports, or documentation.

**2** Until Alice has her own commit bit, her changes must be committed by a current committer, Bob.

**3** After some time, Bob realizes that Alice could be committing her changes on her own, and he proposes Alice for a commit bit.

**4** Bob emails the appropriate team to propose Alice for a commit bit. If Alice is working on the source, then Bob will contact the core team, but if she is working on documentation, Bob will contact the docs team, or, if she is working on one or more ports, he'd contact the ports team.

**5** The core, documentation, and ports teams can then vote on the proposed commit bit, and if the bit is granted, Bob would become Alice's mentor.

**6** For some period of time Bob will have to approve all of Alice's commits to the tree, a process that requires Alice to check her changes with Bob, via the code review system, https://reviews.freebsd.org.

**7** Once Bob is happy that Alice can get by on her own, he releases her from mentorship. Alice can now go on to mentor new committers on her own.

These seven steps are how the FreeBSD Project continues to bring new blood into the project.

The core team is one of the components of the mentorship process. The core team, consisting of nine members, is elected every two years to help run the project, but the core team does not dictate what the committers work on. Core exists to facilitate the project and occasionally steps in to mediate disagreements that may arise between committers. Core also sponsors other teams, such as the ports, documentation, security, and release engineering by granting hats. Within the project, a particular responsibility is referred to as a hat, and the head of the team is thought of as wearing that hat. A more complete description of the governance of the project is given in this issue by McKusick and Rice.

## Modern Features

While philosophy, community organization, and licenses are important differentiators between FreeBSD and Linux, there are also excellent technological reasons for working with FreeBSD. FreeBSD has many modern features that are not present in any Linux distro, including the UFS and ZFS filesystems, DTrace, and Capsicum.

FreeBSD has always had a modern implementation of the Fast File System, originally designed and built for BSD, and continuously upgraded over the last 30 years to keep pace with changing storage technology. The latest version, UFS2, includes support snapshots, as well as journaled soft updates. Journaled soft updates make recovering from a system failure—one where the integrity of the filesystem must be verified—quick and painless. Prior to the inclusion of soft updates, the process of checking a filesystem, carried out by the fsck program, might take hours on a large disk, or even longer with a multi-terabyte disk. Journaled soft updates remove the need to check over the entire disk, meaning that even an 8T drive can be made ready within seconds after a system reboot.

Large storage deployments, those that contain tens of disks and petabytes of data, are addressed using the Zetabyte Filesystem (ZFS). First designed and implemented as part of Solaris, ZFS was imported into FreeBSD in the early 2000s and quickly gained popularity in systems that required a fully functional volume manager backed by the latest RAID techniques. ZFS in FreeBSD remains fully up-to-date with the code maintained in the OpenZFS project, and several of the ZFS developers now work directly on the code in FreeBSD

Another excellent technology to come out of Sun's Solaris group is DTrace, which was ported to FreeBSD in 2008, and which is now being maintained and updated actively within the FreeBSD source tree. DTrace gives programmers and systems administrators complete system transparency, allowing users to see inside any function call within the kernel or within a program running on FreeBSD, without the need for the original source code.

FreeBSD has always had a pragmatic approach to security, bringing in features that had broad applicability in securing the system. One recent addition, Capsicum, is a good example of this pragmatism. Capsicum is a modern capability system that was built with and for FreeBSD. Capabilities come out of research that was done in the 1970s, but there has never been a practical, widely fielded, capability system in a general-purpose operating system until Capsicum was integrated into FreeBSD. Capsicum helps developers build more-secure software by providing a lightweight framework for the compartmentalization of software. Robert Watson, who built Capsicum, points out, "Without compartmentalization, one vulnerability means that the entire application—and all the data it has access to—are available to the attacker. With compartmentalization, software is still vulnerable, but attackers must work harder to find and exploit many more vulnerabilities before they gain the full rights of the user."

## Conclusion

The goal of this article was to give you a way of explaining to others how FreeBSD is not a Linux distro. We presented differentiators in five key areas, including: technical innovation, tooling, release model, documentation, and the business-friendly BSD license. Whether you're reading this as a consumer of FreeBSD or as someone who uses FreeBSD to produce some other, technical artifact, we're sure that you also have ways in which you see FreeBSD as being distinctly different from a Linux distro. We encourage readers to write the *Journal* to share their ideas. If we have a good response, we'll publish a brief, follow-up article so that your ideas can help others in the FreeBSD community. ●

**GEORGE V. NEVILLE-NEIL** works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, networking and time protocols. He is the coauthor with Marshall Kirk McKusick and Robert N. M. Watson of *The Design and Implementation of the FreeBSD Operating System*. For over 10 years he has been the columnist better known as Kode Vicious. He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler and currently lives in New York City.