FreeBSD from a NetBSD user’s perspective

Table of Contents

I’ve been a NetBSD developer for three years and it’s been my primary operating system for a long time too - on everything: routers, laptops, Raspberry Pis, PowerPC mac minis, Vortex86 embedded boards, and servers.

I’ve recently been using FreeBSD a lot at work. We have a lot of servers and embedded boards running it, and I was given the option of installing anything I wanted on my workstation. I chose FreeBSD to maintain a separation of BSDs between my work and home life ;)

I thought I’d write a little bit about some differences that stand out to me. Since everyone that knows me well knows that typical use cases like web hosting aren’t *really* my jam, and I’m more of an embedded, audio, and graphics person, maybe I can offer a more uncommon perspective.

Packages

Packaging for FreeBSD Ports is very straightforward coming from a NetBSD/pkgsrc background, and I imagine it’s the same in the other direction. pkgsrc was forked from FreeBSD Ports a few decades ago, and runs on many more operating systems - so the developers have to be much more careful about portability.

The most obvious difference when you install them right away is that NetBSD defaults to installing third-party software to `/usr/pkg` and FreeBSD defaults to `/usr/local`.

On NetBSD, I use `/usr/local` for files that I install system-wide by means other than the packaging system. On servers, routers, and so on, I keep documentation on what the machine does in in `/usr/local/man`.

I suppose I could use `/opt` for this on FreeBSD, but most installation scripts default to `/usr/local`.

I'm not sure how ingrained the idea that FreeBSD packages are always installed to `/usr/local` is - sometimes it's nice to have other directory layouts, for example, if you want older and newer package trees to co-exist. I suppose this problem is more commonly solved using jails on FreeBSD.

For managing binary packages, FreeBSD seems to always give you the high-level `pkg` tool by default, which is comparable to `pkgin` for pkgsrc. However, NetBSD by default comes with the much more rustic `pkg_add`, and pkgin is optional for those who want it. pkgin only has one man page compared to pkg's several, but it also isn't an all-in-one-replacement for the lower-level packaging tools, just a user-friendly frontend - with pkgsrc you need the `pkg_` tools for some tasks.

For managing source packages, poudriere seems pretty similar to pbulk, except it uses jails instead of chroots. You can do limited and customized builds and overall the feature sets seem very similar.

FreeBSD Ports sometimes shows me a text menu for configuring package options, while pkgsrc just uses text configuration and tries to make it so you don't need to reconfigure packages. I'm the kind of person who prefers text configuration, but I can see how others wouldn't.

Init system

FreeBSD's init system is a port of NetBSD's init system, so it's interesting to see what's different.

On NetBSD, init files are loaded from a single directory - `/etc/rc.d`. Since pkgsrc is not allowed to touch files outside its installation prefix, if you want to use init files from packages you have to either copy them from `$PREFIX/share/examples/rc.d` (making site-specific customizations if necessary), or make a symlink.

On FreeBSD, `/usr/local/etc/rc.d` is also a search path that the init system uses, so you can immediately run init scripts installed by a package (presumably only if you didn't change the default installation prefix). This is probably more natural for users coming from Linux.

While writing some init scripts of my own, I noticed a few things missing from the FreeBSD rc.subr man page, maybe there's some synchronizations to be made there.

Hardware and drivers

I’m not going to whine about missing drivers, etc - a serious BSD user knows to check the hardware before buying it and keeps a stack of USB network adapters around in case of emergencies. Seriously though, I’m more interested in the differences in how hardware support is *handled*.

Webcams and webcamd

> Note
> Internal laptop webcams are nearly always USB.

Something that quite shocked me was how differently webcam support is handled. In NetBSD, there is a generic driver for all USB video devices built in to the default kernel. This implements an API that’s somewhat compatible with Video4Linux. When the driver detects your webcam, it will automatically create `/dev/video0` which can be recorded with ffmpeg, and streamed from Firefox.

The only meeting software that I’ve tested extensively on NetBSD is Jitsi Meet - it works perfectly in Firefox. Other video conferencing web tools often require you to use Chrome, which hasn’t been ported to NetBSD (the eternally-not-upstreamed full BSD patch set is huge). I’m told that Zoom works if you configure Firefox with a fake user agent.

In FreeBSD, it seems you first have to install webcamd, "an application which is a port of Linux USB device drivers into userspace on FreeBSD." You configure it to attach to a specific generic USB device which is exposed by the kernel to userspace, e.g. `ugen0.6`, after loading the `cuse` module. I’m not sure if there are specific advantages to doing it this way.

Right now, I just don’t use my webcam during meetings - I don’t think anyone minds.

GPU drivers and modules

On FreeBSD, you have to explicitly install and load graphics drivers, while on NetBSD they’re baked in to the default kernel and will automatically be loaded. FreeBSD has a proprietary driver for Nvidia GPUs, which takes some extra care to configure, while on NetBSD the open source driver nouveau is used instead.

I often wish the ported-from-Linux DRM/KMS GPU drivers were absent from the default kernel on NetBSD, mostly because they inflate the size significantly and trip up the Kernel Undefined Behavior Sanitizer often, but I also appreciate them being automatically loaded and installed by default - surely it’s possible to strike a nice balance here by having them as modules that automatically load.

Kernel modules seem to be necessary more often on FreeBSD in general.

Jails, VMs, and chroots

FreeBSD Jails are very nice. NetBSD doesn't really have a comparable feature, although it does have hardened chroots which are commonly used as sandboxes.

On NetBSD, there is a surprising amount of tooling for working with chroot sandboxes - my favourite is sandboxctl. It is really quite amazing for what it is, with a few commands you have a NetBSD/i386 8.0 shell on a NetBSD/amd64 9.2 host machine. It even automatically handles downloading the operating system. With FreeBSD it seems recommended to use the `bsdinstall` tool (this is just the normal FreeBSD installer program) to set up jails, which is quite surprising.

It is possible to apply some extra security restrictions to jails, for example, disallowing raw sockets. The restrictions seem quite similar to what NetBSD's kauth framework was designed to handle, maybe we could build something similar on that (wink, nudge).

For VMs, I haven't used FreeBSD's bhyve enough to compare it to NetBSD's NVMM/HAXM/Xen. NVMM and HAXM use QEMU, so you probably get access to more peripherals and running older operating systems might be easier due to the many decades of emulated hardware baked into QEMU. On the other hand, QEMU is this huge piece of software with documentation scattered all over the web and no man pages.

Multicast DNS

I use Multicast DNS on my home network - mostly this is just for convenience, and so I don't have to remember lots of different IP addresses.

NetBSD includes multicast DNS support by default through a port of macOS's mDNSResponder called mdnsd. It seems it's available for FreeBSD but not a core part of the OS. I'm also under the impression that it's more common to use Avahi (the Linux community's answer to mDNSResponder) on FreeBSD, but I don't know for certain.

There are other bits of available-by-default-in-NetBSD software that needs to be installed from packages in FreeBSD, tmux, and bozohttpd for quickly sharing files over a network.

Some of the other default tooling is quite different. I'm very used to using progress in my shell scripts on NetBSD, and the pw man page is much scarier than NetBSD's useradd. Instead of using ftp to download files from HTTP, you use `fetch`… which, fair enough.

Storage

I use ZFS on both and haven’t had a problem with it on either.

The NetBSD bootloader doesn’t handle ZFS natively, so I have a small FFS root partition and then layer ZFS on top of it. Having a small FFS root for recovery is probably a good thing in my case, because I keep building new kernels, forgetting to build the ZFS module (because modules aren’t used much on NetBSD!), then wondering why `/home` is missing on next boot.

Last updated 2021-06-06 18:32:24 CEST