FreeBSD® JOURNAL

July/August 2021

A Straight Path to the FreeBSD Desktop

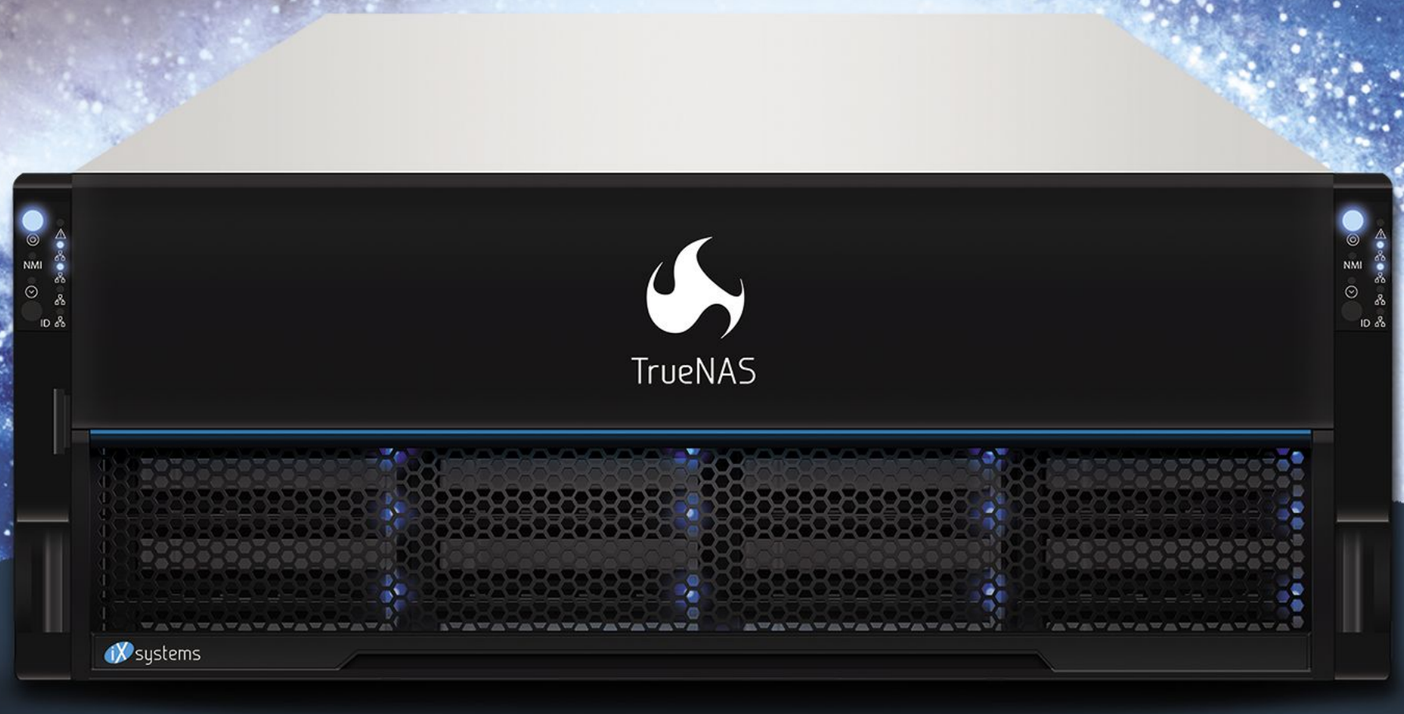
Human Interface Device (HID) Support in FreeBSD 13

The Panfrost Driver

Updating FreeBSD from Git

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS



UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY

IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

- PERFORMANCE AND SCALE WITHOUT COMPROMISE
- INTELLIGENT STORAGE OPTIMIZATION
- SELF-HEALING DATA PROTECTION
- UNLIMITED SNAPSHOTS AND REPLICATION

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

iXsystems.com/TrueNAS | (855) GREP-4-iX

iXsystems™

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Foundation

One of the myths surrounding FreeBSD is that it is only useful in server environments or as the foundation for appliances. The truth is FreeBSD is also a desktop operating system. FreeBSD's base system and packages include device drivers for modern graphics adapters and input devices. Consistent with FreeBSD's role as a toolkit, FreeBSD supports a variety of graphical interfaces ranging from minimalist window managers to full-featured desktop environments. The first article in this issue walks through several of these options explaining how users can tailor their desktop to their needs. It also provides pointers to downstream projects which build an integrated desktop system on top of FreeBSD. The next two articles dig into the details behind some of the newer desktop-supporting device drivers. Finally, our fourth article in this issue explains how to stay current with FreeBSD's development after the project's recent migration from Subversion to Git.

In June the editorial board along with developers from the FreeBSD Project hashed out themes for the next six issues of the *FreeBSD Journal*. We are very excited about these topics. As always, we love to hear from readers. If you have feedback on articles, suggestions for topics for a future article, or would like to write an article, please email us at info@freebsdjournal.com.

- FreeBSD Development (Sep/Oct 2021)
- Storage (Nov/Dec 2021)
- Ports and Packaged Base (Jan/Feb 2022)
- ARM64 is Tier 1 (Mar/Apr 2022)
- Disaster Recovery (May/Jun 2022)
- Science, Systems, and FreeBSD (Jul/Aug 2022)

Finally, we would like to thank George Neville-Neil and Kristof Provost for their service on the editorial board. Both individuals have stepped down for a well-earned break. George has served on the editorial board since the Journal's founding and was the first chair of the editorial board. Kristof has served on the editorial board for the past two years. Even though they are no longer on the editorial board, they will still be around the *Journal*'s pages. One of them (or at least his alter ego) may even make an appearance in this issue…

On behalf of the FreeBSD Foundation,
John Baldwin
Chair of the *FreeBSD Journal* Editorial Board

FreeBSD® JOURNAL

July/August 2021

Desktop/Wireless

Plus

WeGet letters

by Michael W Lucas

Dear Michael,

As you know, KV is a fan of FreeBSD and of travel. Over the years, I've been through Lenovos, Dells, System 76, and some wacky one offs, and it seems that every couple of years, the things that work or don't work when I first load FreeBSD, change, and then I get to track down all the device IDs and try to update drivers so that FreeBSD can be my Daily Driver on my laptop. Although there is a pretty detailed Wiki page for FreeBSD on Laptops (https://wiki.freebsd.org/Laptops), what I've always wanted to see is a "FreeBSD Laptop Shootout," though I admit that some days I just want to shoot my laptop. Surely you have some advice for me and for your readers on how to pick a laptop and make it work as your primary interface to all things FreeBSD. What brands consistently work? Does X11 and the like work well? Will FreeBSD eat my battery, as it's already eaten my brain? How about networking and the Wi-Fis? There must be a way to cut this cake so that I and others can have it and eat it too!

—KV

KV? As in Kode Vicious?

As in the person who suggested George V Neville-Neil chain this albatross of a letter column around my neck? (https://freebsdfoundation.org/past-issues/big-data/) That Kode Vicious? Surely not! No, really, it can't be. He'd know better than to show himself around these parts--especially as, over three years into this travesty of a column, George still hasn't paid me. Never mind that mailing gelato presents challenges and a pandemic has scrambled conferences, travel, and life for two of those years. I refuse to permit my indignation to be undone by any puny "reality."

But this is my only letter this month, and I refuse to disappoint either of my devoted fans. Yes, yes, I had three, but one was unable to withstand any further exposure to the Truth and had to

bow out. I don't blame him for that. I blame him for thinking he was strong enough in the first place, but that's completely separate. I suppose I must allow my righteous distress to further ferment, at least until I have opportunity to properly express my displeasure. Tickets to the event will be $20 each, and all proceeds will go to the Vexed Columnist Legal Defense Fund. Bring a raincoat and eye protection, some of the flying bits might be sharp.

So: Laptops.

Laptops are the worst possible hardware for running an open-source operating system, with the possible exception of a deceased badger. Laptop manufacturers pull every scam they can imagine to reduce weight and power consumption. When it comes to cutting costs, they solicit scams from their suppliers. If a laptop manufacturer needs everything on a printed circuit board shifted a quarter millimeter to the left so they can accommodate the power cable for the wallet vacuum, the component manufacturer will merrily create a new part number without changing any of the hardware's design or firmware—unless, of course, they want to slip a minor change into this new model, nothing to worry about, we're just rearranging the ABI to put the commands in base twelve order as per ancient Sumerians, those ancient priests had the right idea and it's so much simpler than the finger method our executives rely on today. The device model numbers are irrelevant except when they're vital, and the manufacturer's contribution to the development process is a proprietary screen color tuning button that's hooked into a legacy PS2 connector because they got the case cheap on Overstock.

And you want to run FreeBSD on one of these?

The easiest way is to make someone else do the work. The FreeBSD laptop wiki is sadly under advertised, but it's a great place to check before buying a laptop. Many of the laptops listed there are older, but any laptop built in the last decade probably has sufficient computing power for your trivial workloads if you replace the spinning rust with a flash drive. Perusal of the wiki would lead me to believe that ThinkPads are not a bad choice, as much as any laptop can be "not bad."

But suppose you want a brand-new model. The odds of a brand-new laptop being fully supported are finite, but negligible. You might get pretty close, though, especially if you don't choose hardware released last week.

The easiest way to figure out a decent model is to go to a store that has an intriguing model and booting off a USB drive. Unfortunately, wannabe hackers asked store clerks if they could reboot from a USB to load lame malware onto aforementioned floor models, so you're stuck making friends with corporate IT and testing new models before they get the mandated image.

Without such access, you're stuck going to the manufacturer's web site and checking the laptop's technical specifications. That's great, except for the specifications not being specific. With any luck, you can get information on the video, sound, and network.

Graphics vendors delight in keeping details of their hardware private. Presumably they'd rather pay for people to write device drivers than have bored developers deliver said drivers for free. The FreeBSD wiki has a page on graphics (https://wiki.freebsd.org/Graphics), including compatibility matrixes for X.org, so you have a small hope of not wasting that beautiful retina display.

While you can get perfectly adequate USB sound and network devices that will certainly work with your laptop, many people have this narrow-minded insistence that what comes inside laptop should just work. But remember that part where device manufacturers might—or might not—change part numbers when they make trivial changes? Yeah. That. Fortunately, sound is mostly supported. Network cards get new models with the change of the moon, but you might get lucky.

Suppose you took the plunge, bought a laptop, and found that one or two little things don't work. What can you possibly do? You fix it, of course. Failing that, you make it super easy for someone else to fix it. Most FreeBSD developers are running -CURRENT, so you'll want a way to boot -CURRENT on your laptop. One of those flash drive installations is enough. Then run FreeBSD's "spill your hardware's guts" command, `pciconf -lv`. This dumps complete information about all the devices controlled and managed through the system's PCI bus, which is basically everything except the padded box it arrived in.

Then compose a message for hackers@freebsd.org. I've written elsewhere about that message, so I'm not going to belabor the topic any further unless the one person who makes it this far in the article sends a note asking me to tell them how to be something other than a complete jerk in email. The weirdly generous folks there get a thrill out of making new hardware work but be sure to treat them well and thank them kindly. Some of them can hold a grudge for decades against an innocent letters columnist who was having a spectacularly bad day.

It's not that the output will let the hacker support your device. But it provides vital clues. FreeBSD developers are well accustomed to transforming breadcrumbs into miracles, but if you lure one into helping you, expect multiple rounds of back-and-forth debugging and testing. You'll be building and trying new kernels at the very least.

Once you understand exactly how poorly you chose a laptop, the least you can do is update the wiki.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS is the author of *Absolute FreeBSD*, *TLS Mastery*, *git sync murder*, and the forthcoming *Domesticate Your Badgers* and *DNSSEC Mastery*. He is fully convinced that laptops serve best as blunt instruments..

new faces of FreeBSD

BY DRU LAVIGNE

This column aims to shine a spotlight on contributors who recently received their commit bit and to introduce them to the FreeBSD community.

In this installment, the spotlight is on Charlie Li, who received his ports bit in April.

Tell us a bit about yourself, your background, and your interests.

• Li: I was born in China but came to the United States in time for my fifth birthday with, among other things, a now-textbook-thick Toshiba Satellite with Windows 95 that at the time to me was mostly a toy but started my journey in breaking software. I pretty much grew up contributing to open source in one way or another between the end of elementary school and beginning of middle school, initially with reporting bugs whilst running Firefox (the included updater didn't work for an entire release cycle) and soon after contributing to Wikimedia projects including some years as a Wikimedia Commons administrator.

I now bounce between Philadelphia and northeast Pennsylvania, balancing my technical consulting business with a second round of undergraduate education in Electrical Engineering Technology (ABET-accredited) at Penn State University. Interests in the computing space chiefly include making sure our desktop side works well enough that those curious can use it, and embedded applications where much of my business has headed. Outside of computing, I play amateur radio (callsign: K3CL) and race road and track bicycles.

How did you first learn about FreeBSD and what about FreeBSD interested you?

• Li: I had known that FreeBSD and the rest of the BSD family existed since the early 2010s but not much more than a cursory glance at Wikimedia content about them, not least because I was still wrangling with getting Linux to work smoothly with my desktops and laptops. During the FreeBSD 10.2-RELEASE cycle, I noticed my then-VPS provider started offering images of FreeBSD, so curiosity got the best of me, and I created a VM. It quickly became my primary web and mail server for my personal stuff, and later, during the 11.1-RELEASE cycle, I spun up another VM for my business side as well.

It was a "slow" learning process since I was very much buried in the Linux mindset at the time, but one of the aspects that got me hooked quickly was the separation between base system and ports. From there, I started dabbling with non-default ports options, which eventually led to replacing Linux with FreeBSD on my laptop for the "full experience." This was the early stages of graphics/drm-kmod that would panic my system periodically, but I kept at it and tried to be as helpful as possible to those working on this code. For my desktop, I had to resort to using MATE, which I had used for a bit on Linux, as it was up-to-date in ports at the time,

but deep down I wanted to continue using Cinnamon, which was effectively unmaintained in our tree after 2.4. This and more outdated ports gave me a crash course on ports maintenance and I'm still at it.

Seeing how others have used FreeBSD as a basis for their own systems, products and businesses, along with a supportive community to this end, cemented my interest.

How did you end up becoming a committer?

• Li: The desktop@ folks noticed that I not only effectively maintain the Cinnamon desktop environment and some other (mostly desktop-related) ports, but care about the overall health of the desktop "subsystem" or ecosystem. They may have been trying to punish me for a while, but life gets in the way and the asks get forgotten and remembered. During one of the "remembered" cycles we got it done.

How has your experience been since joining the FreeBSD Project? Do you have any advice for readers who may be interested in also becoming a FreeBSD committer?

• Li: Since joining the Project, overall, the experience has been positive. I still have a lot of learning and relationship building to do with existing Project members, but it's a process. Otherwise, it has been exciting to help out in a different role in this community and representing us in the greater open source world.

For those interested in thinking about being a committer, don't. I know it sounds harsh, but this is not something that one can really ask for, but rather, as previously stated, one can only get punished with the bit. There is a lot of responsibility that comes with the commit bit and much of it does not even involve code or documentation, but rather taking that systems thinking approach to everything we do. Relationships in and outside this project are vital to getting stuff done.

Something that I emphasize to any contributor, particularly in ports and especially the desktop ones, dogfood everything before thinking about submitting a patch. As a user, nothing angers me more than software that was not at the very least runtime tested, even in a cursory manner, before committed into the tree, even if the port follows everything in the Porter's Handbook to the letter and thus should theoretically work. No, testport doesn't count as dogfooding. An honest maintainer/committer who is up-front either in advance or on request about known or unknown issues gleaned from dogfooding is received better than one who submits/commits based on passing testport, but otherwise doesn't mind the runtime nuances. We don't need to start additional fires apart from the technical ones.

In essence, submit useful quality contributions, mind the overall system, build relationships, those will go a long way. Never forget that technology is a tool and is nothing without people.

DRU LAVIGNE is the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

A Straight Path to the FreeBSD Desktop

BY VERMADEN

So, you want to try a FreeBSD desktop? Well, maybe you should think again because even people who write and modify FreeBSD daily — FreeBSD developers — used Macbooks for years rather than trying to run a FreeBSD desktop. If that's the case, then why should you try it?

After many years of transition from 4.4BSD and 386BSD to FreeBSD, it remains UNIX at heart and does things a UNIX way. Simple things remain simple and complicated things remain as simple as possible. It's still X11-based and uses traditional plain text files for configuration. Some people actually prefer that — especially after `systemd(1)` from the Linux world. But FreeBSD is a lot simpler and more coherent and it also has several subsystems or features that you will not find in the Linux world. For example, ZFS Boot Environments or GEOM storage framework. There are also Jails, the Base System concept, a great audio subsystem mixed directly in kernel, and many more.

In this article I will try to straighten your path to a FreeBSD desktop.

Hardware

Hardware is the most important part. Really. I have made this mistake more than once. I got hardware that was not supported by FreeBSD. I really wanted the 'most powerful' Intel X3000 graphics card that was packed with Intel G965 based motherboards. I wanted to pair that with the very powerful (back then) Intel Core 2 Quad Q6600 CPU. Unfortunately, FreeBSD did not support that GPU. It supported ALL OTHER Intel GPUs like GMA 950 or GMA 3000, but not that one. That one decision forced me to the Ubuntu Linux world for about a year, and that was not a very happy year as you can probably imagine.

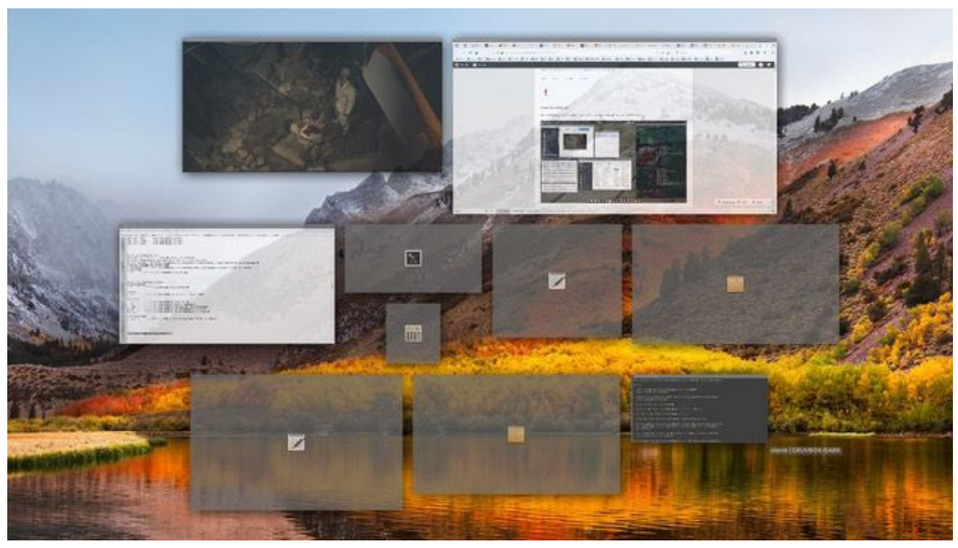
The first rule of thumb with FreeBSD hardware is to NOT get the latest things. To play it safe, look for things that have been on the market for at least two years. That way, you will minimize the chance of ending up with something that's not supported. You should also refer to Hardware Notes for the latest editions of FreeBSD. At the time of this writing, that would include 12.2-RELEASE and 13.0-RELEASE. Here they are:

- https://www.freebsd.org/releases/13.0R/hardware/
- https://www.freebsd.org/releases/12.2R/hardware/

Keep in mind that they are also not perfect. For example, they list many AMD CPUs as supported while they fail to mention AMD EPYC, server CPUs, and AMD Ryzen desktop/mobile CPUs when FreeBSD works on them flawlessly.

Laptops

A lot of people prefer 'mobile computing' instead of classic PC. Generally, IBM and Lenovo ThinkPad laptops are quite well supported by FreeBSD. I prefer the classic 7-row IBM keyboard, so I use a 2011 (yes, a decade old computer) ThinkPad W520 on which everything works beautifully. Others with that classic keyboard are X220/T420/T420s/T520. But you do not need to go back in time that far. For example, ThinkPad X1 Carbon Generation 5th and 6th work reliably. ThinkPad X460/X470/T460/T470 are also ok. There is whole list of laptops tested under FreeBSD with helpful notes, and there is also a BSD Hardware Database. I encourage you to check both before spending your money.



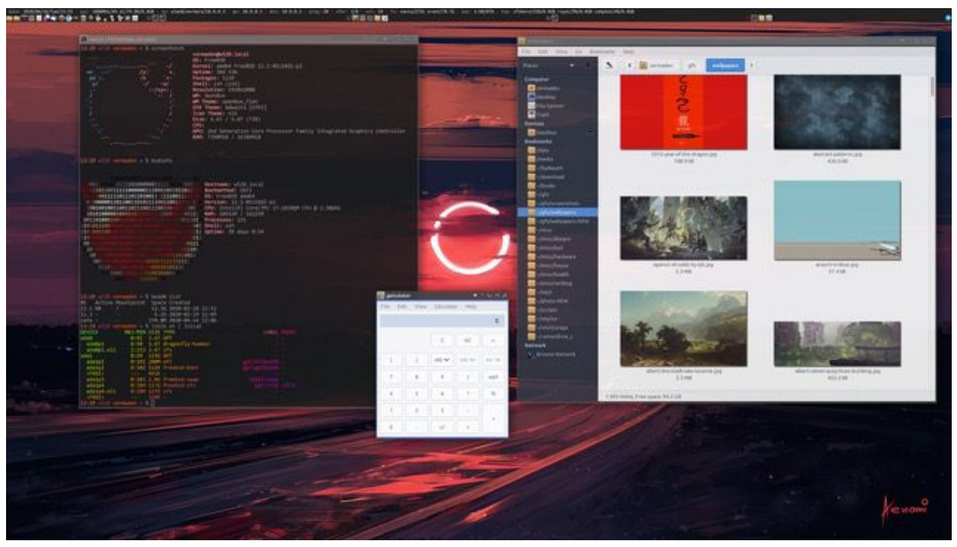
- https://wiki.freebsd.org/Laptops
- http://bsd-hardware.info/

WiFi

This is the part that requires the most rework and improvement in FreeBSD land, but on many fronts, it is already happening. The topic was discussed at the latest FreeBSD Developer Summit, and the FreeBSD Foundation had already noticed that problem and started an initiative to address it. Currently, it's safe to bet on 802.11n chips--as most work, but some only in 802.11g mode (slower). One such chip is Realtek RTL8188CUS, which is really a 'temporary workaround' when you find that the WiFi chip in your laptop is not supported. It's a tiny USB 802.11n WiFi adapter that sticks out several millimeters from the USB-A port. The downside is that FreeBSD only supports it in 802.11g mode, but it still does the job.

GPU

The GPUs that work well under FreeBSD fall into three categories: Intel or AMD GPUs that are supported by open-source drivers; Nvidia GPUs that are well supported by an Nvidia binary driver without any open-source code; ultra-old or too-new GPUs that are just not supported at all. Personally, I use low-power integrated Intel cards and they work well for me, but if you seek more power, then you will have to look at AMD or Nvidia products. I would prefer AMD solutions, as their drivers and designs are open-source, but a Nvidia closed binary driver also works well for many.



Bluetooth

Personally, I see Bluetooth more as a mobile phone thing then a laptop thing, but Bluetooth is present in all laptops and even in many SBCs like Raspberry Pi. Bluetooth on FreeBSD for a mouse or keyboard should work, but it can also 'kill' your suspend/resume cycle on some laptops.

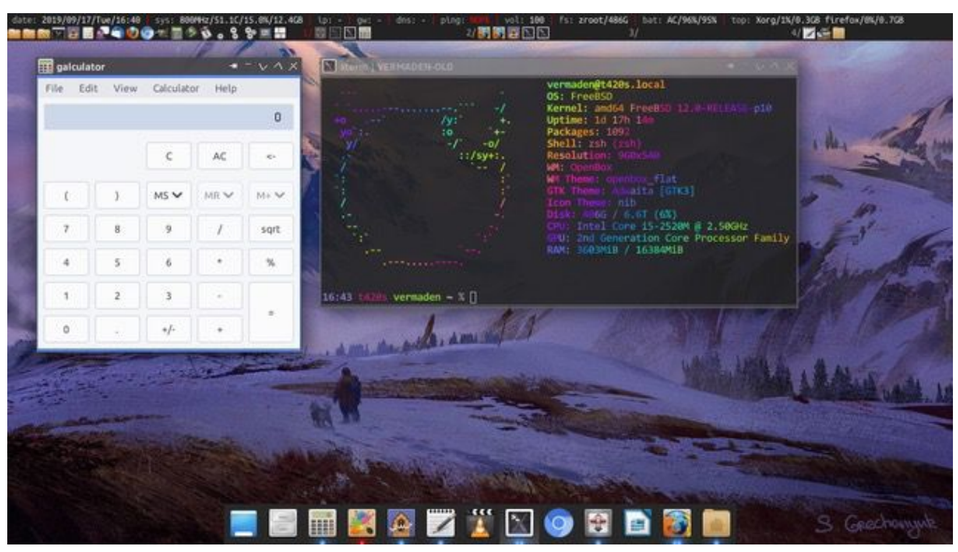
BIOS Settings

Sometimes you need to change several BIOS settings to make your laptop properly suspend/resume. The things that sometimes need to be disabled are:

- Bluetooth
- Trusted Platform Module (TPM)

X11

It does not matter if you select FreeBSD 12.2 or 13.0 — both do good job on the desktop. For Intel and AMD GPUs, you will need `graphics/drm-fbsd12.0-kmod` or `graphics/drm-fbsd13-kmod` respectively. For the Nvidia GPU, you will use the appropriate x11/nvidia-driver version instead, as there are several so check which one is best for your GPU in the Nvidia Release Notes for FreeBSD. As for X11 itself, one can use either `x11/xorg` or `x11/xorg-minimal`. Use the latter one if you wany the smallest possible number of installed packages. In a way, that is pointless, because when you start to install GTK/QT desktop software, you will end up with about 1,000 packages anyway and about 10 GB of used space.

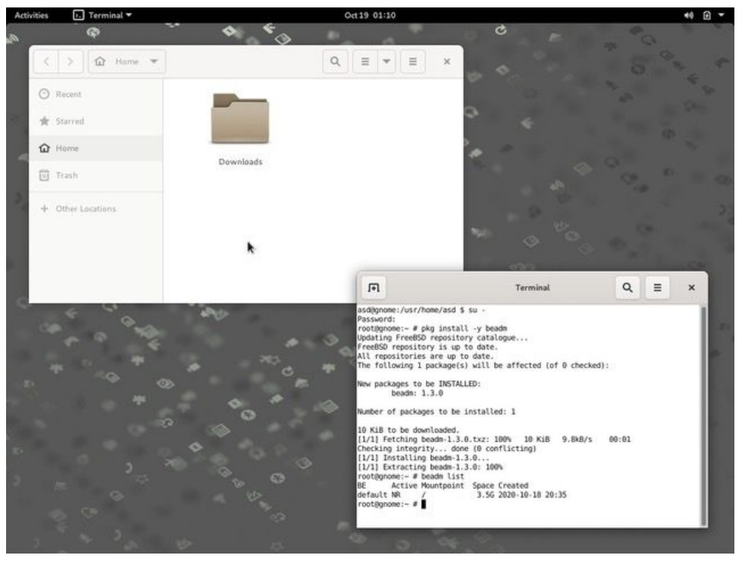


Do not forget that FreeBSD has lots of documentation on X11 — in the Handbook and FAQs:

- https://freebsd.org/handbook/x11
- https://freebsd.org/faq/#X11

Graphical Environment

Choosing a graphical environment is up to you and and there are many to choose from. Some people prefer minimalistic Openbox or the FVWM stacking window manager. Others prefer tiling window managers such as i3 or Awesome. Some choose light environments like MATE or XFCE. Others find fully fledged desktop environments like GNOME or KDE the most comfortable. If you stick to a more minimalistic window manager approach, you will need to create your own environment with your own status bar and information bar. Decide if you want a notification daemon or not and whether to use your system tray and clipboard manager or not at all — with desktop environments these decisions are made for you. You do not have to reinvent the wheel again :)



Ready Solutions

You may install FreeBSD and set it up to create your own desktop your own way, but if you just want to see how it feels to use a FreeBSD desktop, then you may try one of the several ready to use FreeBSD desktop-oriented solutions. I will add a short summary of each of them here - more insight is available at the FreeBSD Foundation page — https://freebsdfoundation.org/freebsd-project/resources/guide-to-freebsd-desktop-distributions/.

GhostBSD

It's probably one of the older and more mature ones. It uses MATE as its graphical desktop. It also uses the OpenRC init system instead of the default rc(8) FreeBSD one. That may be attractive to some Linux users who already know the OpenRC system. They also have an XFCE variant if you find that attractive. Check https://www.ghostbsd.org/.

NomadBSD

This one focuses on Openbox with small additions for very light desktop. It also uses Tint2 and Plank and it is similar to a MacOS layout. It has several interesting DSB tools for auto-mounting or sound volume control. Their page is https://nomadbsd.org/ address.

helloSystem

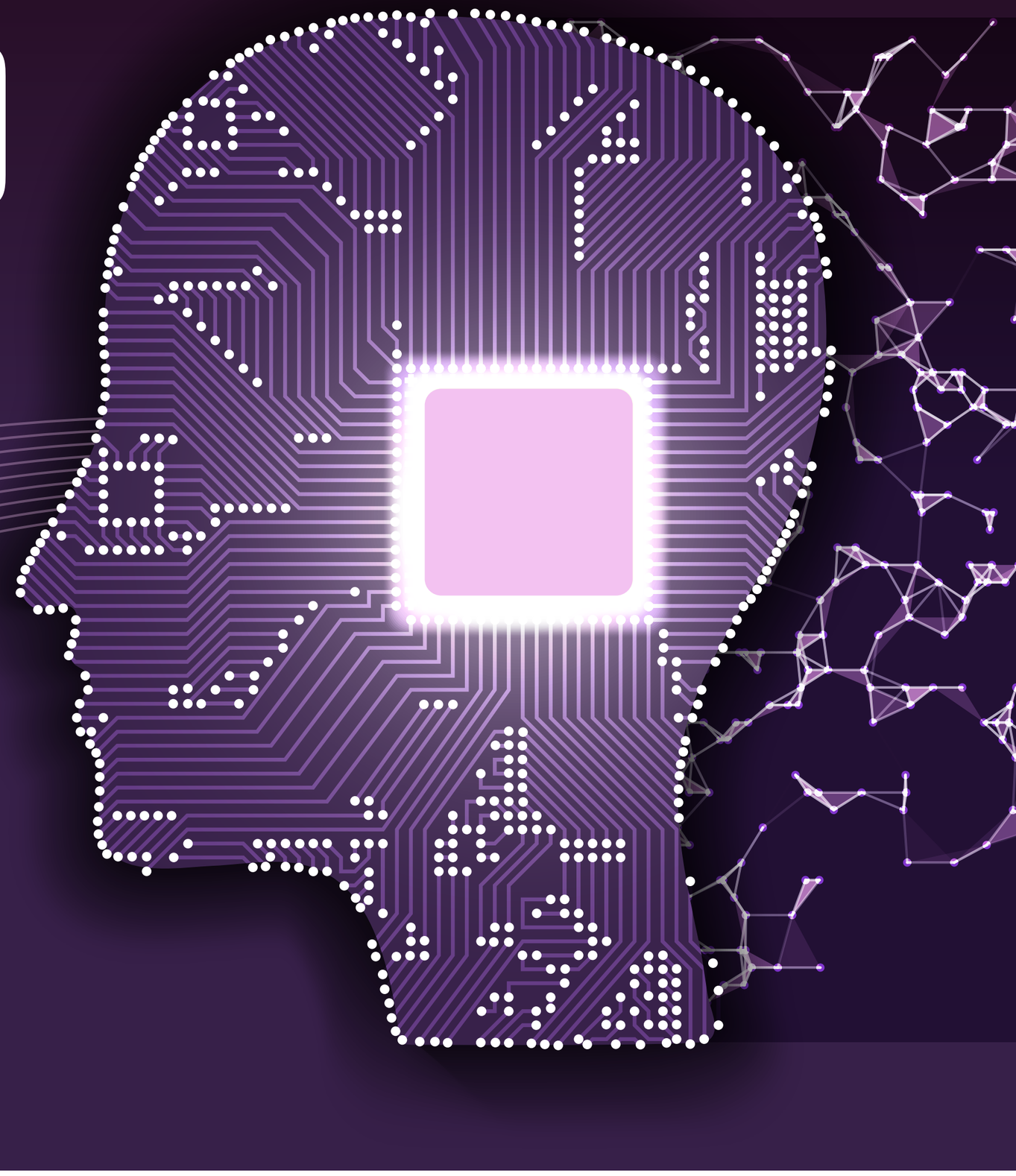
The helloSystem is still in very early development but has some unique features not even available on Linux like a global menu search/filtering system. Its graphical part is written mostly in QT. Check the details at https://hellosystem.github.io/docs/.

Closing Thoughts

I chose the 'create-my-own-desktop' path and built my graphical environment with Openbox as the window manager — described in 26 parts in my FreeBSD Desktop series — https://vermaden.wordpress.com/freebsd-desktop/. You should also check the FreeBSD Foundation summary on the X11 setup — https://freebsdfoundation.org/freebsd-project/resources/installing-a-desktop-environment-on-freebsd/.You may also use the desktop-installer to do lot of that work for you — https://www.grayhatfreelancing.com/freebsd-desktop-workstation-quick-build/.

VERMADEN is another ${RANDOM} sysadmin sharing his work experiences in the IT industry.

Human Interface Device (HID) Support in FreeBSD 13



BY VLADIMIR KONDRATYEV

The HID class consists primarily of devices that are used by humans to control the operationof computer systems. Typical examples of HID class devices include keyboards, pointing devices like standard mouse devices, trackballs, and joysticks. It was adopted primarily to simplify the installation of such devices.

Prior to the appearance of the HID, devices usually conformed to strictly defined protocols. All hardware design improvements resulted in either overloading the use of data in an existing protocol or the creation of custom device drivers and the propagation of a new protocol to developers. By contrast, all HID-defined devices deliver self-describing packages that may contain any number of data types and formats. The key idea is that information about a HID device is stored in segments of its ROM (read-only memory). These segments are called descriptors. A single HID driver on a computer parses descriptors and enables dynamic association of data with application functionality, which has enabled rapid innovation and development, and prolific diversification of new human-interface devices.

Among descriptors of different types, there is one which any HID device must have independently of physical transport. It is called a HID report descriptor. The report descriptor prescribes each piece of data that the device generates and what the data is measuring. The sample of a report descriptor for a 3-button mouse with wheel and tilt follows:

```
0x05, 0x01,         // Usage Page (Generic Desktop)
0x09, 0x02,         // Usage (Mouse)
0xa1, 0x01,         // Collection (Application)
0x09, 0x01,         //  Usage (Pointer)
0xa1, 0x00,         //  Collection (Physical)
0x05, 0x09,         //   Usage Page (Button)
0x19, 0x01,         //   Usage Minimum (1)
0x29, 0x03,         //   Usage Maximum (3)
0x15, 0x00,         //   Logical Minimum (0)
0x25, 0x01,         //   Logical Maximum (1)
0x95, 0x03,         //   Report Count (3)
0x75, 0x01,         //   Report Size (1)
0x81, 0x02,         //   Input (Data,Var,Abs)
0x95, 0x05,         //   Report Count (5)
0x81, 0x03,         //   Input (Const,Var,Abs)
0x05, 0x01,         //   Usage Page (Generic Desktop)
0x09, 0x30,         //   Usage (X)
0x09, 0x31,         //   Usage (Y)
0x09, 0x38,         //   Usage (Wheel)
0x15, 0x81,         //   Logical Minimum (-127)
0x25, 0x7f,         //   Logical Maximum (127)
0x75, 0x08,         //   Report Size (8)
0x95, 0x03,         //   Report Count (3)
0x81, 0x06,         //   Input (Data,Var,Rel)
0x05, 0x0c,         //   Usage Page (Consumer Devices)
0x0a, 0x38, 0x02,   //   Usage (AC Pan)
0x95, 0x01,         //   Report Count (1)
0x81, 0x06,         //    Input (Data,Var,Rel)
0xc0,               //  End Collection
0xc0,               // End Collection
```

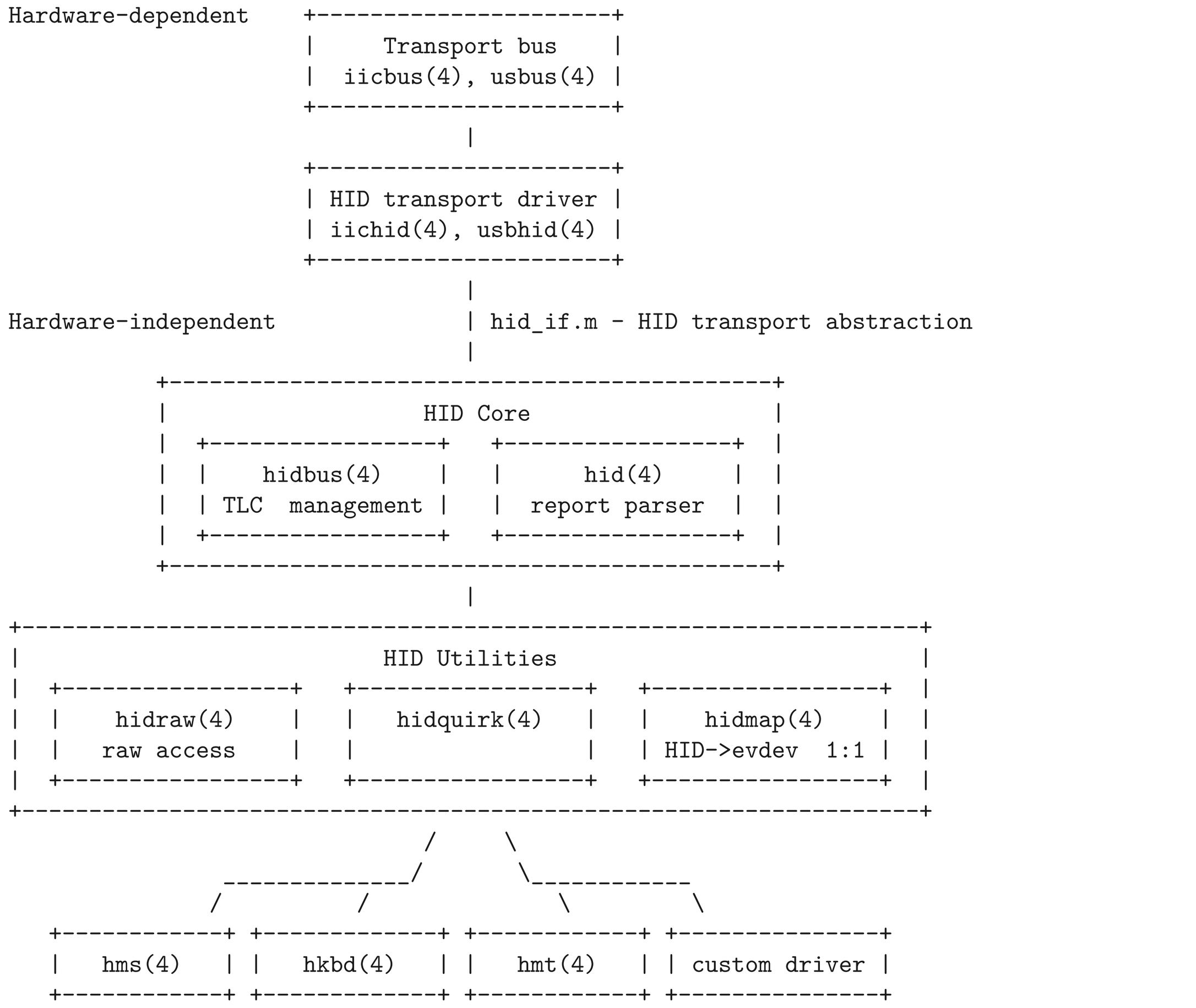
Listing 1. Decoded HID report descriptor for a 3-button mouse with wheel and tilt axes

The initial import of HID support from NetBSD was done in 1998 along with the USB stack. It consisted of a report descriptor parser and 3 drivers based on it: ukbd(4), ums(4) and uhid(4). A userspace version of a report descriptor parser known as libusbhid(3) was imported later in 2000. It became a base of HID support in the bluetooth stack, bthidd(8), committed in 2004. While such a combination of drivers mostly suits desktop needs, it does not cope well with laptops and hand-held devices. Microsoft published HID-over-I2C specifications which were adopted by almost all (with Apple being the main exception) laptop producers. Touch devices got a large market share and a lot of complex devices that combine the functionality of two or more simpler devices appeared e.g., a keyboard with a mouse, or touch screen with a pen tablet and so on. All this necessitated revision of the HID subsystem.

HID Subsystem Architecture

The new HID subsystem is designed as a bus. Any transport bus may provide HID devices and register them with the HID core. The HID bus then loads generic device drivers on top of it. The transport drivers are responsible for raw data transport and device setup/management. The HID core contains helper routines for report-parsing and is responsible for auto discovery of a child device for each Top Level Collection described by the report descriptor. Generic device drivers are responsible for report interpretation and the user-space API. Device specifics

and quirks are handled by hidquirk and raw access is handled by the hidraw driver. hidmap is the HID item to the evdev event converter. Generic HID functionality is moved out of USB-HID into a new subsystem under a dev/hid directory and became a separate kernel module. That is something Open/NetBSD did 5 years ago.



HID subsystem architecture.

HID Transport Drivers

A transport bus normally provides hotplug detection or device enumeration KPIs to the transport drivers. Transport drivers use this to find any suitable HID device. They allocate HID device resources and attach hidbus. Hidbus is never aware of which transport drivers are available and is not interested in it. It is only interested in child devices.

Transport drivers implement an abstract HID transport interface that provides independent access to HID capabilities and functions through the device tree. A kobj-based HID interface can be found in sys/dev/hid/hid_if.m. Once hidbus child is attached, the bus methods are used by HID core to communicate with the device. Currently, the FreeBSD kernel includes support for USB, and I2C drivers.

hidbus

hidbus is a driver that provides support for multiple HID driver attachments to a single HID transport back-end. This ability existed in Net/OpenBSD from the day 0 (uhidev and ihidev drivers) but has never been ported to FreeBSD. Unlike Net/OpenBSD we do not use report number alone to a distinct report source, but we follow the MS way and use a Top Level Collection (TLC) usage to which the report belongs.

A TLC is a grouping of functionality that targets a particular software consumer (or type of consumer) of the functionality. An operating system uses the Usage associated with this collection to link the device to its controlling application or driver. Common examples are a keyboard or mouse. A keyboard with an integrated pointing device could contain two different application collections. The HID device describes the purpose of each TLC to allow the consumers of HID functionality to identify the TLC in which they might be interested. Hidbus generates a child device for each TLC described by the Report Descriptor and adds PnP strings to allow devd/devmatch to detect the proper driver. While running, hidbus broadcasts data generated by transport drivers to all the children.

```
0x05, 0x01,         // Usage Page (Generic Desktop Ctrls)
0x09, 0x06,         // Usage (Keyboard)
0xA1, 0x01,         // Collection (Application)
0x05, 0x07,         //   Usage Page (Kbrd/Keypad)
0x85, 0x01,         //   Report ID (1)
0x19, 0xE0,         //   Usage Minimum (0xE0)
0x29, 0xE7,         //   Usage Maximum (0xE7)
0x15, 0x00,         //   Logical Minimum (0)
0x25, 0x01,         //   Logical Maximum (1)
0x75, 0x01,         //   Report Size (1)
0x95, 0x08,         //   Report Count (8)
0x81, 0x02,         //   Input (Data,Var,Abs)
0x95, 0x01,         //   Report Count (1)
0x75, 0x08,         //   Report Size (8)
0x81, 0x01,         //   Input (Const,Array,Abs)
0x95, 0x06,         //   Report Count (6)
0x75, 0x08,         //   Report Size (8)
0x15, 0x00,         //   Logical Minimum (0)
0x26, 0xA4, 0x00,   //   Logical Maximum (164)
0x05, 0x07,         //   Usage Page (Kbrd/Keypad)
0x19, 0x00,         //   Usage Minimum (0x00)
0x29, 0xA4,         //   Usage Maximum (0xA4)
0x81, 0x00,         //   Input (Data,Array,Abs)
0xC0,               // End Collection
0x05, 0x01,         // Usage Page (Generic Desktop)
0x09, 0x02,         // Usage (Mouse)
0xa1, 0x01,         // Collection (Application)
0x09, 0x01,         //  Usage (Pointer)
0xa1, 0x00,         //  Collection (Physical)
0x85, 0x02,         //   Report ID (2)
0x05, 0x09,         //   Usage Page (Button)
0x19, 0x01,         //   Usage Minimum (1)
0x29, 0x03,         //   Usage Maximum (3)
0x15, 0x00,         //   Logical Minimum (0)
0x25, 0x01,         //   Logical Maximum (1)
```

```
0x95, 0x03,         //   Report Count (3)
0x75, 0x01,         //   Report Size (1)
0x81, 0x02,         //   Input (Data,Var,Abs)
0x95, 0x05,         //   Report Count (5)
0x81, 0x03,         //   Input (Const,Var,Abs)
0x05, 0x01,         //   Usage Page (Generic Desktop)
0x09, 0x30,         //   Usage (X)
0x09, 0x31,         //   Usage (Y)
0x15, 0x81,         //   Logical Minimum (-127)
0x25, 0x7f,         //   Logical Maximum (127)
0x75, 0x08,         //   Report Size (8)
0x95, 0x02,         //   Report Count (2)
0x81, 0x06,         //   Input (Data,Var,Rel)
0xc0,               //  End Collection
0xc0,               // End Collection
```

Listing 2. HID report descriptor for keyboard with integrated mouse consisting of 2 TLCs.

hidmap

hidmap is a generic HID item value to an evdev event conversion engine that makes it possible to write HID drivers in a mostly declarative way by defining translation tables. The motivation behind its creation was that existing USB-HID drivers are overgrown in size due to following factors:

- USB transfer handling
- Character device support code
- Protocol conversion routines e.g. HID to sysmouse or HID to AT kbd set 1
- Long hid_locate() and hid_get_data() chains in report parsers

p.1 is eliminated with help of the transport abstraction layer.

To solve p.2 support for legacy, the mouse interface was dropped. We use character device handlers built in evdev.

To reduce the amount of code required by p.3 and p.4, hidmap was created. It is based on the fact that HID and evdev are close relatives and we can directly map many HID usages to evdev events. Listing 3 illustrates a sample mapping of HID usages to evdev events for the mouse report from Listing 1.

```
                        HID Usage                       mapped evdev event
                        ---------                       ------------------
0x05, 0x09,         //   Usage Page (Button)
0x19, 0x01,         //   Usage Minimum (1)              BTN_LEFT   (BTN_MOUSE+0)
0x29, 0x08,         //   Usage Maximum (3)              BTN_RIGHT  (BTN_MOUSE+1)
0x95, 0x08,         //   Report Count (3)               BTN_MIDDLE (BTN_MOUSE+2)
0x81, 0x02,         //   Input (Data,Var,Abs)
0x05, 0x01,         //   Usage Page (Generic Desktop)
0x09, 0x30,         //   Usage (X)                      REL_X
0x09, 0x31,         //   Usage (Y)                      REL_Y
0x09, 0x38,         //   Usage (Wheel)                  REL_WHEEL
0x95, 0x03,         //   Report Count (3)
0x81, 0x06,         //   Input (Data,Var,Rel)
0x05, 0x0c,         //   Usage Page (Consumer Devices)
0x0a, 0x38, 0x02,   //   Usage (AC Pan)                 REL_HWHEEL
```

```
0x95, 0x01,         //   Report Count (1)
0x81, 0x06,         //   Input (Data,Var,Rel)
```

Listing 3. Mapping of HID usages to evdev events for mouse report from Listing 1.

With help of hidmap, the mouse driver for such a device fits in a couple dozen lines of code. See Listing 4.

```
/* my_mouse's HID usage to evdev event mappings */
static const struct hidmap_item my_mouse_map[] = {
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_X,    REL_X) },
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_Y,    REL_Y) },
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_WHEEL,  REL_WHEEL) },
  { HIDMAP_REL(HUP_CONSUMER,        HUC_AC_PAN, REL_HWHEEL) },
  { HIDMAP_KEY_RANGE(HUP_BUTTON,    1, 3,     BTN_MOUSE) },
};

/* A match on these entries will load my_mouse */
static const struct hid_device_id my_mouse_devs[] = {
  { HID_TLC(HUP_GENERIC_DESKTOP, HUG_MOUSE) },
};

static int
my_mouse_probe(device_t dev)
{
  return (HIDMAP_PROBE(device_get_softc(dev), dev,
      my_mouse_devs, my_mouse_map, "My mouse"));
}

static int
my_mouse_attach(device_t dev)
{
  return (hidmap_attach(device_get_softc(dev)));
}

static int
my_mouse_detach(device_t dev)
{
  return (hidmap_detach(device_get_softc(dev)));
}
```

Listing 4. Sample of hidmap-based driver for mouse report from Listing 1.

For example, the real FreeBSD mouse driver has shrunk from ~1200LOC in traditional ums(4) to ~330 LOC in hms(4) based on new HID KPI. Also, it got I2C and absolute coordinate support missing in ums(4) and drift suppression code required to work around issues connected to missing GPIO interrupt support in FreeBSD on x86. This simplicity allowed the author and Greg V to create a bunch of hidmap-based drivers that are bundled with FreeBSD 13+, namely:

- hms - HID mouse driver.
- hcons - Consumer page AKA Multimedia keys driver.
- hsctrl - System Controls page (Power/Sleep keys) driver.
- hpen - Generic / MS Windows compatible HID pen tablet driver.
- hgame - Game controller and joystick driver.

•xb360gp - driver for Xbox360-compatible game controllers.
•ps4dshock - Sony DualShock 4 gamepad driver.

There are also some drivers which are not based on hidmap like hkbd (4)and hmt(4). They are ports of existing USB-HID drivers, namely ukbd(4) and wmt(4) to the new infrastructure. They add support for I2C keyboards and I2C MT touchpads/touchscreens.

Other Modules

The HID subsystem contains two other modules that can be loaded optionally:

hidraw(4) - driver for raw access to HID devices resembling uhid(4). Unlike uhid(4), it allows access to devices that were claimed by other drivers and supports both uhid and Linux hidraw interfaces.

Hidquirk(4) - quirk module which was mostly copied from existing USB-HID.

Conclusion and Further Work

The FreeBSD HID subsystem is still under development but is already used by many. Recent work added awaited support for widely used hardware like I2C touchpads and touchscreens, multimedia keys on a USB keyboard, game controllers, absolute mice found on many VMs and so on. This has led to improved user experience in areas where we lagged behind other OS-es including other BSDs. But there are lot of things still left in TODO list e.g:

- Implement usrhid, a user-space transport driver that can create kernel hid-devices for each device connected to the user-space controlled bus. Existing Linux uhid protocol can be used as a starting point. It defines an API for providing I/O events from the kernel to user-space and vice versa.
- Convert bthidd(8) to use usrhid to consolidate HID support between the kernel and user spaces.
- Complete evdev-awared WIP moused https://github.com/wulf7/moused and replace our in-base moused(8) with it. This is required as new hms and hmt drivers do not support our legacy sys/mouse.h interface.
- Enable usbhid(4) by default and start deprecation of ums(4), ukbd(4) and other legacy USB-HID drivers along with deprecation of all mouse(4)/sysmouse(4) stuff in the kernel and base system

---

VLADIMIR KONDRATYEV is an ex-FreeBSD system administrator and currently a core banking system specialist. He has been a FreeBSD committer since 2017 and a FreeBSD desktop user for the last ~20 years. In his spare time, he tries to improve the desktop experience, contributing mostly to input device drivers.

The Panfrost Driver

BY RUSLAN BUKIN

The mature operating system and the new graphics driver for the arm64 platform gets us ready to run FreeBSD on high-end server/desktop systems, research-platforms, embedded environments, or even current smartphones/personal devices.

Every tech giant is considering moving—or is already moving—their hardware to arm64 architecture. Google just announced the Tensor chip for their new Pixel 6 devices; Amazon built AWS Graviton processors; Facebook is testing massive arm64 server deployment and powers Oculus AR/VR glasses with an arm64 SoC. The Apple M1 SoC debuted in Macbooks. Many Android devices and Google Chromebooks are now based on arm64.

At the same time, the arm64 port for FreeBSD has been promoted to a Tier 1 platform and a few arm64 server-solutions are now available from different vendors. Some people in the FreeBSD community are already using arm64 machines as FreeBSD desktops.

The largest ARM market is still smartphones (the ARM share is nearly 100%) and personal devices. These are based on a variety of ARM-based SoCs with a choice of peripherals and network connectivity options included (5G, etc.). But the selection of Mobile GPUs the SoCs are based on is limited: Vivante, Broadcom VideoCore, Nvidia Tegra, Qualcomm Adreno, Imagination PowerVR, ARM Mali and the graphics included for the Apple M1 SoC.

FreeBSD does not support any of these GPUs, but work is in progress on one of them, so, let's talk about the ARM Mali GPU.

The Mali GPU

The Mali GPU was first launched in 2005 by Falanx Microsystems A/S (a Norwegian company)—a spin-off from a research project of the Norwegian University of Science and Technology, and then acquired by ARM in 2006. Since 2007, ARM has developed several generations/micro-architectures of Mali: Utgard, Midgard, Bifrost and a very new Valhall. The Utgard GPU (Mali 4xx) was a GPU of the armv7 era, but it is still available on a newer low-end arm64 systems. ARM claims it is the world's most shipped mobile GPU ever. Midgard (Mali Txxx) and Bifrost (Mali Gxx) are newer GPUs that are often included with an arm64-based smartphone or tablet.

The Mali GPU has several threads intended for different operations. For example, the Midgard GPU uses complex tiling architecture. It has three threads responsible for vertex shaders; a tiler that sorts triangles into tilers and passes down to a fragment shader; and the fragment shader that processes final rasterization over passed-in tiles (i.e. writes to the framebuffer).

The Mali GPU supports Vulcan 1.2, OpenGL ES 3.2, OpenCL 1.2 (Midgard architecture), OpenCL 2.0 (Bifrost architecture), and OpenCL 2.1 (Valhall architecture). SPIR-V is also supported by Mali GPUs.

Software

ARM writes its own proprietary software drivers in-house for these GPUs. These are binary blobs for Linux. Binary blobs are not available for FreeBSD and using them on Linux ties you to a specific version of the Linux distribution/kernel and display subsystem. That restricts your ability to use your machine as you see fit and undermines the philosophy of free and open-source software. It also provides no way to debug the code and adds security issues. It disallows kernel hackers from using the latest development kernel on their desktop machines. But the good news is that there are two open-source reverse-engineered (RE) drivers for Mali GPU now available: Lima and Panfrost.

A GPU Open-source Driver

The graphics stack consists of several components, mainly the `userspace` graphics driver and the kernel part of it, but also the direct rendering module (DRM), kernel mode setting (KMS) and `libdrm`.

The Userspace Part (the Mesa Library)

A graphics driver runs in userspace and makes the translation between the Graphics API (OpenGL, OpenCL, etc) to the native sequence of instructions (opcodes) in each GPU ISA.

The driver composes job chains that have to be executed on GPU. A job is a sequence of instructions—a bunch of abstract memory buffers, including temporary buffers and main data. A job also comes with a set of dependencies (called 'fences') that must be met before a job can be put on a GPU thread to run. For example, a job could depend on completion of another job. A few to a few hundred jobs per second are executed on the GPU during the graphics workload. A job is composed by the graphics driver, which is a part of the Mesa Gallium library.

The Lima Driver

You may have heard about Lima—an open-source reverse-engineered driver for Mali. It is made for Utgard micro-architecture. It targets OpenGL ES 2.0 (embedded) as well as OpenGL 2.1 (desktop). It can't do newer APIs like OpenGL ES 3.2 or OpenCL/Vulkan due to Utgard hardware limitations. Initially developed by Luc Verhaegen at the beginning of 2012, this work was sponsored by Codethink. The project was abandoned in 2013 and stagnated for some time, before it was resumed in June 2017 by Qiang Yi (a Software Engineer at AMD).

Qiang Yi begun integrating the driver into the Mesa library, and, as of March 2019, the driver was fully merged. The Lima driver is supported on various armv7 boards and Chromebooks. Although the Lima could probably be built on FreeBSD with no issues, we have not written the kernel part for it.

The Panfrost Driver

A new project covers support for Midgard and Bifrost—newer GPU architectures that are included in an enormous number of arm64 systems. It was made by tracing ARM's proprietary userspace driver and open-source kernel drivers.

The original effort began with the Lima driver to support Midgard under the name Tamil, but source code was never released. A new project with the codename Chai was founded by Alyssa Rosenzweig—a mathematics student at the University of Toronto. Alyssa's goal was to write a free software driver for Midgard GPUs, also known as the Mali Txxx series.

Alyssa did most of the Midgard reverse-engineering and driver development. A separate project, BiOpenly, began an effort to reverse-engineer the Bifrost, and then project Chai was merged with BiOpenly under the name Panfrost. Later, Alyssa joined Collabora Ltd to continue work on Panfrost.

Unlike Lima, Panfrost was done in partnership with ARM. It was initially (and almost entirely) done through a self-funded effort by the Panfrost community and Collabora Ltd, but at some point, ARM Ltd joined the effort to make the solution optimal. ARM provided the documentation and helped upstream the Mali kernel driver to Linux.

Build Instructions on FreeBSD

What follows is an example for how to build the Panfrost userspace driver under FreeBSD:

```
git clone https://gitlab.freedesktop.org/mesa/mesa
mkdir build && cd build
meson .. . -Degl=enabled -Dgles1=enabled -Dgles2=enabled -Ddri-drivers= -Dvulkan-drivers=
-Dgallium-drivers=panfrost,kmsro
ninja
```

The Panfrost Kernel Driver

I began working on the Panfrost kernel driver with the goal of creating the first-ever mobile GPU support for the FreeBSD arm64 platform. The GPU is a memory-mapped device like many other peripherals on arm64 SoC. The kernel driver of the graphics stack provides an interface (a 'glue') to the OS and to the actual GPU hardware. It is the only part of the graphics stack that directly accesses GPU memory-mapped registers. It is responsible for GPU power management, hardware initialization, GEM objects management, GPU threads management, MMU configuration, GPU interrupts and MMU pagefault handling. The GPU kernel driver accepts jobs from userspace, allocates buffers, sets up MMU mappings, puts jobs to run on GPU and controls the overall operation of GPU.

As the Panfrost driver supports various GPU models and versions, it is responsible for setting up quirks and workarounds for any hardware issues related to a particular model.

As well as interfacing to DRM and handling its requests, a GPU kernel driver provides an ioctl interface for its userspace part. The ioctl layer is mainly used to submit jobs for processing on GPU, managing buffer objects (BO), and MMU mappings management.

IOCTL calls

A set of ioctl calls has been implemented in the kernel driver:

1. `panfrost_ioctl_submit` – used to submit a job to the kernel driver.
2. `panfrost_ioctl_wait_bo` – waits for a completion of the last `panfrost_ioctl_submit` on a buffer object (BO). This is needed for the cases where multiple processes are rendering to a BO and you want to wait for all rendering to be completed.
3. `panfrost_ioctl_create_bo` – creates Panfrost BO.
4. `panfrost_ioctl_mmap_bo` – used to map Panfrost BOs to userspace. This doesn't perform any mmap but returns an offset to use in an mmap on the DRM device node.
5. `panfrost_ioctl_get_param` – allows the userspace driver to read information about the hardware, such as GPU model, version, features present, etc.
6. `panfrost_ioctl_get_bo_offset` – returns an offset for the BO in the GPU address space.
7. `panfrost_ioctl_madvise` – gives advice to the kernel in the case where buffer contents (backed pages) could be temporarily released back to OS to save memory.

The MMU

The Mali GPU has an internal MMU, and so all the buffers with which the GPU operates are virtually addressed. The format of MMU page tables is mostly compatible with arm64 CPU MMU. There are a few differences, however, that have been discovered during this work:

- The page type bit of a PTE entry is different from the CPU MMU.
- The access flag bit is inverted.
- Some of the permission bits are inconsistent.
- The control registers and TLB maintenance paradigm are wildly different.

Before we put a job on the GPU, we have to setup MMU mappings, and once the work has completed, we must tear them down. The MMU support for the Mali GPU for FreeBSD has been developed, reviewed, and already committed to the base tree. It is part of the arm64 IOMMU framework (see `sys/arm64/iommu directory`). Initially, and similarily to the ARM System MMU (SMMU), the GPU MMU management routines were introduced into the arm64's CPU page management (pmap) code. Later people started complaining that maintaining SMMU, GPU and CPU MMU in the same pmap slows down the development process significally, so we have made a decision to split out SMMU and GPU routines to a new pmap specifically made for IOMMU (`iommu_pmap.c`).

KMS

The Mali GPU only processes rendering. It processes data in memory and puts it back to memory. It does not feature display controllers and has no idea how to output any data to a monitor. Generally, graphics support of an arm64 platform consists of various hardware components:

- The video display processor that operates with hardware windows (planes). At least two planes should be registered—the main window plane and a cursor plane. This device blends them when we move the cursor on the screen (it puts one layer on top of another in hardware) and provides the final data to output to the Display Interface Controller.
- The display interface controller which outputs data to the display on the physical layer. Interfaces like HDMI, DisplayPort, LVDS, MIPI DSI are usually implemented in this device.
- Peripheral devices like I2C/SPI. An i2c controller is usually required to read information about the connected HDMI monitor (its resolution, capabilities, etc.) or to configure a MIPI display.
- The GPU itself.

Along with a set of drivers for these devices, FreeBSD should support the clock framework for a given platform, as a set of clock domains must be configured to transmit data to the display.

The KMS (kernel-mode setting) is a part of DRM. It allows the user to configure the outputs when multiple hardware planes are registered as well as when multiple resolution/connectors options are available.

The DRM

The Direct Rendering Manager (DRM) is a framework in Linux and FreeBSD for managing Graphic Processing Units (GPUs). It is designed to support complex GPUs providing unified access to hardware. It is also responsible for memory management, DMA and interrupt handling.

DRM exposes the API (a set of ioctls) allowing userspace programs to send commands/data to GPU and to configure kernel mode setting (KMS). libdrm provides a set of wrappers around drm ioctl calls to simplify the API for user applications.

We currently support two DRM/KMS projects:

1. drm-kmod and LinuxKPI

DRM appeared in FreeBSD as a '`drm-kmod`' port in 2001 to support 3D graphics cards at that time (also known as `drm1`). From 2010-2012 Konstantin Belousov developed support for the new Intel HD Graphics GPUs introducing new DRM support in FreeBSD base (`drm2`).

A new project, `drm-next`, started around 2017 to support unmodified DRM. With frequent updates upstream, it became difficult to support `drm2`—a ported version of graphical drivers—as it required going through the driver and changing Linux-KPI to FreeBSD-KPI on every driver update.

DRM has been developed in a Linux kernel environment, so it uses the Linux kernel API. To use DRM code 'as is' in FreeBSD, we must translate its calls to native FreeBSD primitives/APIs. In this new '`drm-next`' project it is done by using the LinuxKPI framework. LinuxKPI was originally made for storage devices and networking adapter drivers by Mellanox Inc. and others, but now it is also required for importing graphics drivers from Linux. Due to licensing issues, they can't be part of the FreeBSD kernel (only as a module).

LKPI makes it easier to bring Linux DRM drivers to FreeBSD with minimal modifications and keep them up to date against Linux upstream. Later in 2018, the `drm1` and `drm2` were moved from the kernel to the ports tree as `drm-legacy-kmod` port; while `drm-next port` took the original name '`drm-kmod`'.

This DRM/KMS driver is available not only on x86 with support for most modern Intel HD GPUs, but on powerpc and arm64 platforms as well, targeting PCI-e based AMD/Radeon graphics cards.

I have been using this DRM on the ARM Neoverse N1 (arm64) server with a Radeon PCIe card for several months and it is working just fine.

Installing the driver is as easy as:

```
cd /usr/ports/graphics/drm-kmod
make install
```

2. drm-subtree & DRMKPI

A new project, 'drm-subtree', to support DRM on FDT-based arm and arm64 systems has been started by Emmanuel Vadot and Michal Meloun, with the goal of moving the DRM back to the kernel.

On FDT-based platforms like arm and arm64 the situation with DRM is a bit different: we want to keep DRM in the kernel, as it better suits embedded environments and we don't want to import arm graphics drivers from Linux. One reason is that the frameworks for FDT-based resources (gpio/pinmux/clock/regulator) are different between Linux and FreeBSD and that could make the LinuxKPI abstraction very complex. Also, drivers need to be synced with DTS files that we regularly import from Linux.

This project includes a new DRMKPI framework that is a bit of a stripped-down version of LinuxKPI—but for DRM-only. In DRMKPI, most unneeded storage/networking code—like `netdevice/sysfs` code—is removed, eliminating most fields from Linux's `task_struct` with the goal of eliminating `task_struct` itself, too.

At https://github.com/evadot/drm-subtree you can find a working set of DRM/KMS drivers and instructions for Allwinner, Nvidia Tegra and Rockchip SoCs that allow the display to work under FreeBSD.

A Target Platform for Panfrost

The FreeBSD Panfrost kernel driver has been developed for the Morello Board — a collaboration between ARM and Cambridge Computer Laboratory to build the first CHERI-enabled ASIC. This work is sponsored by UKRI with support from ARM Ltd.

The CHERI CPU that is included with this board is based on ARM Neoverse N1 SoC — the first server-class architecture from ARM, the Cortex-successor. While the Morello Board SoC will include the Bifrost GPU, the N1 development platform has no GPU included, which means we can't use it for GPU driver development.

The rk3399 board used for Panfrost kernel driver development.

The Morello Board is currently in its last stages of manufacturing and will be available later this year. In its absence right now, we have chosen Rockchip RK3399 as the target platform for Panfrost kernel driver development. The RK3399 is a mature and well supported FreeBSD platform with lots of llow-cost single-board computers available on the market. It includes a Mali Midgard GPU, HDMI controller with Full-HD, 4K support and even a PCI-e controller. It is an interesting platform because it can run entirely using free software including the GPU drivers.

The Result

The project started in November 2020, and it took around 6 months for me to have the Mali GPU working. Most of the time was spent writing Rockchip DRM drivers (video engine, display controller) and making bugfixes in peripheral drivers. The most difficult part of the Panfrost driver was figuring out the MMU unit behavior, its page tables format and cache-management operations. I also had to add missing functionality to the drm-subtree (to import the DRM scheduler and its dependencies). Thanks to the people on the bsdmips and panfrost IRC channels for help and moral support!

Quick tests demonstrate good performance and an overall quick response of the desktop graphical environment. The RK3399 has two video display processors: Big and Little. While Big features 4K resolution, the support was added for the Little targeting Full-HD. Both Wayland and Xorg operate well with minor issues like memory leakages (and work is in progress to fix it).

I was able to run a 2D accelerated desktop, browsers and 3D mesa demos like `glxgears(1)` and `geartrain(1)`: the HDMI monitor max refresh rate (60 FPS) was easily reached.

I'm looking forward to switching my main desktop to the Morello Board and to using my Panfrost driver daily. The Panfrost driver for FreeBSD has been put up for review on github: https://github.com/evadot/drm-subtree.

RUSLAN BUKIN is a Senior Research Software Engineer at the University of Cambridge Computer Laboratory. His main area of interest is machine-dependent software. He has led several projects including the port of FreeBSD to RISC-V architecture, IOMMU support for arm64 and Intel SGX support for amd64. Aside from FreeBSD, he has developed the MDEPX embedded kernel used in research projects. In his free time he enjoys cycling across the UK.

Updating FreeBSD From Git

BY DREW GURKOWSKI

With FreeBSD's ongoing migration to git from subversion, the system for updating FreeBSD from source has adapted. This guide will cover getting sources from git, updating them, and how to bisect those sources. It is meant as an introduction to the new mechanics for general users.

1. Keeping Current With FreeBSD src tree

To begin, the source tree must be downloaded. This can be done quite simply. First step: cloning a tree. This downloads the entire tree. There are two ways to download. By default, git will do a deep clone, which matches what most people want. However, there are times that you may wish to do a shallow clone.

1.1 Branch names

The branch names in the new git repository are similar to the subversion names. For the stable branches, they are `stable/X` where `X` is the major release number (like 12 or 13). The development branch for `-CURRENT` in the new repository is `main`.

Note: The main branch is the default branch if you omit the `-b branch` options below.

1.2 Repositories

- The official geographically distributed mirror for the general public is git.FreeBSD.org, The access URL is: https://git.freebsd.org/src.git
- The repository is also accessible by SSH: ssh://anongit@git.freebsd.org/src.git
- There are several officially maintained external mirrors. The list is available at https://docs.freebsd.org/en/books/handbook/mirrors/#external-mirrors
- For using web browser to view the content, there is a cgit web interface at https://cgit.freebsd.org/src
- There is an old experimental github repository at https://github.com/freebsd/freebsd-legacy/

(was https://github.com/freebsd/freebsd) similar to the new Git repository. However, there are a large number of mistakes in the github repository that required us to regenerate the export when we migrated to having a git repository be the source of truth for the project.

The hashes are different between them. For migrating from the old repository to the new one, please refer to https://github.com/freebsd/freebsd-legacy/commit/de1aa3dab-23c06fec962a14da3e7b4755c5880cf

Use the repository of choice in place of $URL in the following commands.

1.2.1 Install git From Ports/Pkg

Before cloning the tree, git will need to be installed. The simplest way of doing so is through packages.

```
# pkg install git
```

There are also git-lite and git-tiny, two packages with only essential dependencies available. They are sufficient for the commands in this article.

1.2.2 Deep Clone

A deep clone pulls in the entire tree, as well as all the history and branches. It's the easiest to do. It also allows you to use git's worktree feature to have all your active branches checked out into separate directories but with only one copy of the repository.

```
% git clone $URL -b branch [dir]
```

is how you make a deep clone. **`branch`** should be one of the branches listed in the **section 1.1**, if omitted, it will be the depository's default: main. Dir is an optional directory to place it in (the default will be the name of the repository you are clone (src). For example:

```
% git clone https://git.freebsd.org/src.git
```

You'll want a deep clone if you are interested in the history, plan on making local changes, or plan on working on more than one branch. It's the easiest to keep up to date as well. If you are interested in the history, but are working with only one branch and are short on space, you can also use **`--single-branch`** to only download the one branch (though some merge commits will not reference the merged-from branch which may be important for some users who are interested in detailed versions of history).

1.2.3 Shallow Clone

A shallow clone copies just the most current code, but none or little of the history. This can be useful when you need to build a specific revision of FreeBSD, or when you are just starting out and plan to track the tree more fully. You can also use it to limit history to only so many revisions.

```
% git clone -b branch --depth 1 $URL [dir]
```

An example using the default branch:

```
% git clone --depth 1 https://git.freebsd.org/src.git
```

This clones the repository, but only has the most recent revision in the repository. The rest of the history is not downloaded. Should you change your mind later, you can do git fetch --unshallow to get the complete history.

2. Building and Updating from Source

After cloning the FreeBSD repository, the next step is to build from source. The process of building remains relatively unchanged, using make and install. Git and offer git pull and git checkout for updating and selecting specific branches or revisions.

2.1 Building From Source

Building can be done as described in the handbook [3], eg:

```
% cd src
% make buildworld
% make buildkernel
% make installkernel
% make installworld
```

Note that you can specify `-j` to make to speed up with parallelism.

2.2 Updating From Source

The following command will update both types of trees. This will pull all revisions since the last update.

```
% git pull --ff-only
```

This will update the tree. In git, a fast forward merge is one that only needs to set a new branch pointer and doesn't need to re-create the commits. By always doing a fast forward merge/pull, you'll ensure that you have an identical copy of the FreeBSD tree. This will be important if you want to maintain local patches.

See below for how to manage local changes. The simplest is to use:

```
% git pull --autostash
```

but more sophisticated options are available.

2.3 Selecting a Specific Revision

In git, the `git checkout` command can be used to checkout a specific revision as well as branches. Git's revisions are the long hashes rather than a sequential number.

When you checkout a specific revision, just specify the hash you want on the command line (the git log command can help you decide which hash you might want):

```
% git checkout 08b8197a74
```

However, as with many things git, it's not so simple. You'll be greeted with a message similar to the following:

```
Note: checking out '08b8197a742a96964d2924391bf9fdfeb788865d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 08b8197a742a hook gpiokeys.4 to the build
```

where the last line is generated from the hash you are checking out and the first line of the commit message from that revision. Hashes can also be abbreviated. That's why you'll see them have different lengths in different commands or their outputs. These super long hashes are often unique after some characters, depends on the size of the repository so git lets you abbreviate and is somewhat inconsistent about how it presents them to users. `git rev-parse --short <full-hash>` will show the short hash which has the enough length to distinguish in the repository. The current length of FreeBSD src repository is 12.

3. Bisecting/Other Considerations

3.1 Bisecting With git bisect

Sometimes, things go wrong. The last revision worked, but the one you just updated to does not. A developer may ask to bisect the problem to track down which commit caused the regression.

If you've read the last section, you may be thinking to yourself "How the heck do I bisect with crazy revision numbers like that?" then this section is for you. It's also for you if you didn't think that, but also want to bisect.

Fortunately, git offers the `git bisect` command. Here's a brief outline in how to use it. For more information, I'd suggest https://git-scm.com/docs/git-bisect for more details. The man page is good at describing what can go wrong, what to do when revisions won't build, when you want to use terms other than good and bad, etc, none of which will be covered here.

`git bisect start` will start the bisection process. Next, you need to tell a range to go through. `git bisect good XXXXXX` will tell it the working revision and `git bisect bad XXXXX` will tell it the bad revision. The bad revision will almost always be `HEAD` (a special tag for what you have checked out). The good revision will be the last one you checked out.

A quick aside: if you want to know the last revision you checked out, you should use `git reflog`:

```
5ef0bd68b515 (HEAD -> master, origin/master, origin/HEAD) HEAD@{0}: pull --ff-only: Fast-forward
a8163e165c5b (upstream/master) HEAD@{1}: checkout: moving from b6fb97efb682994f59b21fe4efb3fcfc0e5b9eeb to master
...
```

shows me moving the working tree to the master branch (a816…) and then updating from upstream (to 5ef0…). In this case, `bad` would be `HEAD` (or 5rf0bd68) and `good` would be a8163e165. As you can see from the output, HEAD@{1} also often works, but isn't foolproof if you've done other things to your git tree after updating, but before you discover the need to bisect.

Back to git bisect. Set the `good` revision first, then set the `bad` (though the order doesn't matter). When you set the bad revision, it will give you some statistics on the process:

```
% git bisect start
% git bisect good a8163e165c5b
% git bisect bad HEAD
```

```
Bisecting: 1722 revisions left to test after this (roughly 11 steps)
[c427b3158fd8225f6afc09e7e6f62326f9e4de7e] Fixup r361997 by balancing parens.
```

You'd then build/install that revision. If it's good you'd type `git bisect good` otherwise `git bisect bad`. You'll get a similar message to the above each step. When you are done, report the bad revision to the developer (or fix the bug yourself and send a patch). `git bisect reset` will end the process and return you back to where you started (usually tip of main). Again, the git-bisect manual (linked above) is a good resource for when things go wrong or for unusual cases.

3.2 Documents and Ports Considerations

The doc tree is the first repository converted to git. There is only one development branch in the repository, main, which contains the source of https://www.freebsd.org and https://docs.freebsd.org.

- The repository URL is at https://git.freebsd.org/doc.git
- The repository is also accessible with SSH: ssh://anongit@git.freebsd.org/doc.git
- And cgit web repository browser is at: https://cgit.freebsd.org/doc/

The ports tree operates a similar way. The branch names are different and the repos are in different locations.

- The repository URL is at https://git.freebsd.org/ports.git
- The repository is also accessible with SSH: ssh://anongit@git.freebsd.org/ports.git
- And cgit web repository browser is at: https://cgit.freebsd.org/ports/

As with ports, the latest development branch is main. The quarterly branches are named the same as in FreeBSD's svn repo. They are used by the latest and quarterly branches of the pkg.

3.3 Coping with Local Changes

Here's a small collection of topics that are more advanced for the user tracking FreeBSD. If you have no local changes, you can stop reading now (it's the last section and OK to skip).

One item that's important for all of them: all changes are local until pushed. Unlike svn, git uses a distributed model. For users, for most things, there's very little difference. However, if you have local changes, you can use the same tool to manage them as you use to pull in changes from FreeBSD. All changes that you've not pushed are local and can easily be modified (git rebase, discussed below does this).

3.4 Keeping local changes

The simplest way to keep local changes (especially trivial ones) is to use `git stash`. In its simplest form, you use `git stash` to record the changes (which pushes them onto the stash stack). Most people use this to save changes before updating the tree as described above. They then use `git stash apply` to re-apply them to the tree. The stash is a stack of changes that can be examined with `git stash list`. The git-stash man page (https://git-scm.com/docs/git-stash) has all the details.

This method is suitable when you have tiny tweaks to the tree. When you have anything non trivial, you'll likely be better off keeping a local branch and rebasing. It is also integrated with the `git pull` command: just add `--autostash` to the command line.

4. FreeBSD Branches

4.1 Keeping a local branch

It's much easier to keep a local branch with git than subversion. In subversion you need to merge the commit, and resolve the conflicts. This is manageable, but can lead to a convoluted history that's hard to upstream should that ever be necessary, or hard to replicate if you need to do so. Git also allows one to merge, along with the same problems. That's one way to manage the branch, but it's the least flexible.

Git has a concept of 'rebasing' which you can use to avoid these issues. The `git rebase` command will basically replay all the commits relative to the parent branch at a newer location on that parent branch. This section will briefly cover how to do this, but will not cover all scenarios.

4.2 Create a branch

Let's say you want to make a hack to FreeBSD's `ls(1)` command to never, ever do color. There's many reasons to do this, but this example will use that as a baseline. The FreeBSD `ls(1)` command changes from time to time, and you'll need to cope with those changes. Fortunately, with git rebase it usually is automatic.

```
% cd src
% git checkout main
% git checkout -b no-color-ls
% cd bin/ls
% vi ls.c # hack the changes in
% git diff # check the changes
```

```
diff --git a/bin/ls/ls.c b/bin/ls/ls.c
index 7378268867ef..cfc3f4342531 100644
--- a/bin/ls/ls.c
+++ b/bin/ls/ls.c
@@ -66,6 +66,7 @@ __FBSDID("$FreeBSD$");
 #include <stdlib.h>
 #include <string.h>
 #include <unistd.h>
+#undef COLORLS
 #ifdef COLORLS
 #include <termcap.h>
 #include <signal.h>
```

```
% # these look good, make the commit...
% git commit ls.c
```

The commit will pop you into an editor to describe what you've done. Once you enter that, you have your own `local` branch in the git repo. Build and install it like you normally would, following the directions in the handbook. git differs from other revision control systems in that you have to tell it explicitly which files to use. Here it's done on the commit command line, but you can also do it with `git add` which many of the more in depth tutorials cover.

5. Updating to New FreeBSD Releases

5.1 Updating to a New FreeBSD Revision

When it's time to bring in a new revision, it's almost the same as w/o the branches. You would update like you would above, but there's one extra command before you update, and one after. The following assumes you are starting with an unmodified tree. It's important to start rebasing operations with a clean tree (git usually requires this).

```
% git checkout main
% git pull
% git rebase -i main no-color-ls
```

This will bring up an editor that lists all the commits in it. For this example, don't change it at all. This is typically what you are doing while updating the baseline (though you also use the `git rebase` command to curate the commits you have in the branch).

Once you're done with the above, you've moved the commits to ls.c forward from the old revision of FreeBSD to the newer one.

Sometimes there's merge conflicts. That's OK. Don't panic. You'd handle them the same as you would any other merge conflicts. To keep it simple, I'll just describe a common issue you might see. A pointer to a more complete treatment can be found at the end of this section.

Let's say this includes changes upstream in a radical shift to terminfo as well as a name change for the option. When you updated, you might see something like this:

```
Auto-merging bin/ls/ls.c
CONFLICT (content): Merge conflict in bin/ls/ls.c
error: could not apply 646e0f9cda11... no color ls
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 646e0f9cda11... no color ls
```

which looks scary. If you bring up an editor, you'll see it's a typical 3-way merge conflict resolution that you may be familiar with from other source code systems (the rest of ls.c has been omitted):

```
<<<<<<< HEAD
#ifdef COLORLS_NEW
#include <terminfo.h>
=======
#undef COLORLS
#ifdef COLORLS
#include <termcap.h>
>>>>>>> 646e0f9cda11... no color ls
```

The new code is first, and your code is second. The right fix here is to just add a `#undef COLORLS_NEW` before `#ifdef` and then delete the old changes:

```
#undef COLORLS_NEW
#ifdef COLORLS_NEW
#include <terminfo.h>
```

save the file. The rebase was interrupted, so you have to complete it:

```
% git add ls.c
% git rebase --cont
```

which tells git that `ls.c` has changed and to continue the rebase operation. Since there was a conflict, you'll get kicked into the editor to maybe update the commit message.

If you get stuck during the rebase, don't panic. git rebase –abort will take you back to a clean slate. It's important, though, to start with an unmodified tree.

For more on this topic, https://www.freecodecamp.org/news/the-ultimate-guide-to-git-merge-and-git-rebase/ provides a rather extensive treatment. It goes into a lot of cases I didn't cover here for simplicity that are useful to know since they come up from time to time.

5.2 Updating to a New FreeBSD Branch

Let's say you want to main the jump from FreeBSD stable/12 to FreeBSD current. That's easy to do as well, if you have a deep clone.

```
% git checkout main
% # build and install here...
```

and you are done. If you have a local branch, though, there's one or two caveats. First, rebase will rewrite history, so you'll likely want to do something to save it. Second, jumping branches tends to encounter more conflicts. If we pretend the example above was relative to stable/12, then to move to main, I'd suggest the following:

```
% git checkout no-color-ls
% git checkout -b no-color-ls-stable-12 # create another name for this branch
% git rebase -i stable/12 no-color-ls --onto main
```

What the above does is checkout no-color-ls. Then create a new name for it (no-color-ls-stable-12) in case you need to get back to it. Then you rebase onto the main branch. This will find all the commits to the current no-color-ls branch (back to where it meets up with the stable/12 branch) and then it will replay them onto the main branch creating a new no-color-ls branch there (which is why I had you create a placeholder name).

References

[1] Using Git, FreeBSD Handbook, https://docs.freebsd.org/en/books/handbook/mirrors/#git

[2] Git, FreeBSD wiki, https://wiki.freebsd.org/Git

[3] Updating FreeBSD from Source, FreeBSD Handbook, https://docs.freebsd.org/en/books/handbook/cutting-edge/#makeworld.

DREW GURKOWSKI, FreeBSD Foundation

PRACTICAL PORTS

Want Some Toppings on Your Desk?

BY BENEDICT REUSCHLING

This column covers ports and packages for FreeBSD that are useful in some way, peculiar, or otherwise good to know about. Ports extend the base OS functionality and make sure you get something done or, simply, put a smile on your face. Come along for the ride, maybe you'll find something new.

Years ago, when I began my Unix journey as a university freshman, installers were a lot simpler. Getting to a desktop at the end was not the default. In fact, I struggled a lot at the beginning to get one going, and when I did, the computer I had back then did not have enough power to run it. So, I stayed in the terminal for a long time. This did not turn me away, as I could already do a lot more than on DOS. At least I had misc/mc to get me a little bit more than white text on a black background.

Later, I did get a working desktop running on X11R6. Nowadays, the installers are much better and often default to a graphical point-and-click interface. But when that happened, I was switching to FreeBSD where a curses-like installer is still the default. My hard-learned tricks in the non-GUI world proved useful. The FreeBSD handbook had me starting up a working desktop much faster than my stunts on other distributions. I still use a minimal desktop environment because, most of the time, I need to open a terminal to be productive. I used x11/icewm initially, then switched to x11/fluxbox. Then there was a period during my university studies where all the other students had flashy desktop effects like desktop-switching on a rotating cube, transparency, and such. It got boring after a while, because everyone had them, so they ended up being nothing special. Since I couldn't get them to work back then, I wanted to save face and ran x11-wm/enlightenment. That was interesting, because who does not compile config files into binaries to get some speed benefits, right? I guess some concepts don't catch on with other software systems—as revolutionary they might be.

Coming from Windows before I started my Unix experiences, I had certain expectations of a desktop. How do you launch applications if you don't have any icons on the desktop? And

while we're at it: how do I put a flashy background image on fluxbox (which I switched back to)? I can't work this way! I mean, transparent terminal programs at least show you parts of it when you are not browsing the web. OK, there are plenty of terminal programs available that can do that, but most terminal emulators come with a big shopping list of dependencies. No thanks, I'll pass. Speaking of emulation, how about a trip to the days of cathode ray tubes that ruined our eyes? With x11/cool-retro-term nothing stops you from reliving those days. And don't blame me for your next eye appointment!

There was also the problem of launching applications without icons (and less than favorable quick launch menus). Having seen what the Mac could do with Alfred (alfredapp.com), it was only a matter of time until someone sat down and built something similar for the Unix desktop. Since the name of Bruce Wayne's butler was already taken, the next best name was x11/albert, of course. For those who want a good file manager, I hear good reports from x11-fm/dolphin in combination with devel/dolphin-plugins. Even x11-fm/xfe gets you running wild in your filesystem with a small runtime dependency list. For those who don't want to run a desktop to have terminals side by side, there is always sysutils/screen or sysutils/tmux at your side. Detaching and re-attaching screens and sessions on a server (which does not run a desktop unless you're on Windows) is common these days. When it starts to get complex, sysutils/tmuxinator can help you manage those sessions with relative ease. Or look at sysutils/byobu for a bit more elegance and system status notifications.

> How do you launch applications if you don't have any icons on the desktop?

A while ago, I started getting into tiling window managers (perhaps because of liking tmux a little too much). Not only did that kill the "icons on the desktop" idea, it also impacted my habit of typing rather than pushing around a mouse all day. While getting an x11/i3 desktop running alongside x11/i3status, x11/py-i3-quickterm, I also needed to learn some new keystrokes to control it. The experience has been good so far and x11/i3-gaps looks like my next upgrade from there. But now that I have discovered that whole new world of mouseless desktopping, I have also learned of x11/rofi. Rofi is an application launcher like Albert. Give it a try, run rofi and it shows you what it can do. For example, ssh into one of those boxes with ease. Or browse your files or executables like on Apple's Spotlight application.

There is always a debate about work you can do better on a desktop versus in the terminal. "Surely you can't do graphics editing in the console?!" I hear skeptics say. Well, that depends on what needs doing. You can go a long way with graphics/ImageMagick7. Starting from cropping, rotating, and putting watermarks on images, or putting your scanned signature on a contract. You don't need to run graphics/gimp for that. Does it replace the Paintshop Pro experience of old? Probably not. But hey, most changes are small and easy to batch up in the terminal. Who wants to manually rotate all those holiday images from years ago taken by a shabby flip-phone camera?

At work, I discovered that some PDF meeting notes were not searchable. It turned out they were printed, signed by participants, and scanned into a PDF. That killed the full-text search and

copy-and-paste as the PDF was an image now. I had discovered textproc/ocrmypdf which basically does the reverse. It applies OCR software and machine learning to restore the text portions of a PDF and converts it back to text. Since we had many such documents, batching them up in the terminal was quick work. I have not dared to think how long it would take to manually fix that, desktop program available or not. Use the right tool for the job, even if it's a desktop-based one. And while my own journey in desktop-land continues, I'll record my software findings for you along the way.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. He serves on the board of directors of the FreeBSD Foundation as vice president. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Together with Allan Jude, he is host of the weekly bsdnow.tv podcast.

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

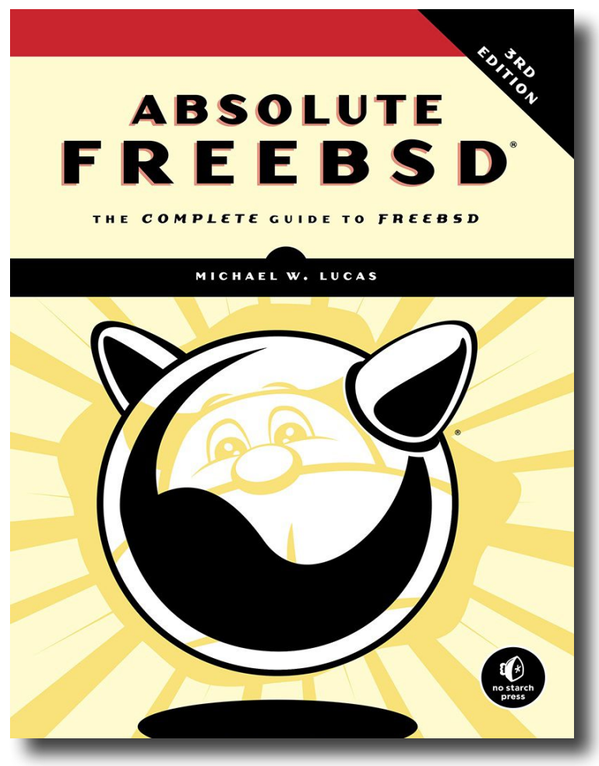
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

BOOK REVIEW

Michael W. Lucas's *Absolute FreeBSD* (3rd Edition): A Scholar's Review

BY COREY J. STEPHAN



On the penultimate page of *Absolute FreeBSD: The Complete Guide to FreeBSD* (3rd Edition), Michael Lucas suggests that the best way for a newcomer to this operating system to find a niche in which to contribute is to answer a simple question: "What do you do?" My answer is equally simple: Like Mr. Lucas, I research, write, and teach non-fiction, albeit as a historical scholar rather than as a system administrator. It seems fitting, then, that my second contribution to the FreeBSD Journal is a review of Lucas's 617-page guide to the operating system that congregates us nerds of different types.

Even as a quintessentially critical academic, I struggle to find negative comments to make about *Absolute FreeBSD*. The only pressing one pertains to the volume's subtitle, "*The Complete Guide to FreeBSD*." The volume is not lacking; to the contrary, it contains precisely what it ought to contain for its purpose as a guidebook for system administrators. However, in my opinion, a *complete* guide to this operating system should contain several chapters on desktop usage. Moreover, the use cases for FreeBSD outside of traditional server and desktop environments are growing. Low-powered, single-board ARM computers, such as those produced by Pine64 and the Raspberry Pi Foundation, are often amateur tinkerers' first points of contact with Unix-like operating systems. A nod toward the various avenues by which neophyte Beasties who are not sysadmins typically start to grow our horns would be a boon to *Absolute FreeBSD*'s eventual fourth edition.

Of course, 'so-and-so should have written such and such' is the cheapest criticism one can make of another's published writing, and the fact that it is the only one that I have for *Absolute FreeBSD* previews my overall assessment. *Absolute FreeBSD* is a grand pedagogical achievement, both for Lucas himself and for the FreeBSD project writ large. I read every word on all 617 pages. Especially in the early chapters, there were times when it was difficult for me to return *Absolute FreeBSD* to its place on my bookshelf to resume professional or familial duties.

The success of *Absolute FreeBSD* has little to do with its technical correctness. With any published work of non-fiction, accuracy is the minimum standard for acceptability. *Absolute FreeBSD* is in its third edition from a press that specializes in computer science, and it carries the imprimatur of four of FreeBSD's luminaries: Marshall Kirk McKusick, John Baldwin, Benno Rice, and George V. Neville-Neil. Thus, its factual reliability is self-evident. Yet no one would (or should) sit down to read every chapter of the reliable but strictly utilitarian and encyclopedic *FreeBSD Handbook*. Instead, *Absolute FreeBSD* has a lively character that makes it downright pleasant to read.

The same Michael Lucas who has made both my wife (a murder mystery aficionada) and me (a geek) chuckle while reading *$ git commit murder* side-by-side for date nights pours his narrative acumen into a book that is not a dry reference volume but, rather, an animated, story-like guidebook. *Absolute FreeBSD* triumphs because its author is a novelist at heart.

In the Introduction, Lucas remarks that he intends *Absolute FreeBSD* to be read "once, from front to back." Lucas leads the reader through a step-by-step, hands-on learning process that requires installing and using the operating system. For my own study, having FreeBSD on my (AMD64) home-built desktop with ZFS and on my (ARM) Raspberry Pi 4 with UFS made it so that I was able to do most of the prescribed exercises (at least to the "y/n" prompt).

The most important chapters of *Absolute FreeBSD* are the first, second, and final (twenty-fourth). In the first chapter, Lucas explains how to navigate several venues for help with FreeBSD, including man pages, the *FreeBSD Handbook*, the mailing lists that are best for beginners, and the FreeBSD Forums. In addition to obeying Lucas's advice to join #freebsd-questions and #freebsd-security-notifications, I also joined #freebsd-arm, wherein I learned how to run FreeBSD on the Raspberry Pi 4 before it gained official support with 13.0-RELEASE. Years of using other operating systems programmed me (so to speak) to open my favorite search engine at the first sign of trouble, but Lucas divulged a better way—the FreeBSD user's way, which normally starts with 'man _.' In the second chapter, Lucas outlines how to plan a new installation of FreeBSD. From EFI to swap, Lucas maintains a thoughtful simplicity that makes it easy to complete the most daunting task that he assigns. In the final chapter, Lucas details what to do when FreeBSD crashes or otherwise fails, including how to file useful bug reports.

Michael Lucas pours his narrative acumen into a book that is not a dry reference volume but, rather, an animated, story-like guidebook.

The core of *Absolute FreeBSD* runs from Chapter 3 through Chapter 23. As college classes in mathematics, philosophy, and anatomy have their own internal progressions, so too is *Absolute FreeBSD* an orderly, self-contained course. From installation (Ch. 3), boot (Ch. 4), backup (Ch. 5), and kernel configuration (Ch. 6) to advanced security (Ch. 19) and system monitoring (Ch. 21), *Absolute FreeBSD*'s forward march is unyielding.

Yet within the trek that is *Absolute FreeBSD*, Lucas displays a judiciousness that makes performing the exercises more fun than one might expect. *Absolute FreeBSD* is comprehensive without being overwhelming and detailed without being arcane. Part of Lucas's refreshing genius is that he knows what *not* to include. I am overly acquainted with 500-page scholarly monographs that should have been condensed to 200 pages apiece (if not 100) before they were published. Lucas never meanders, nor does he flood his pages with surplus facts. I did not read the first edition, which No Starch Press published before I was old enough to own a computer (and I am now married with children and finishing a Ph.D.), but this third edition still strikes me as the result of meticulous revision—both because its biography spans two decades and because it is void of wasted words.

Absolute FreeBSD's central hinge subtly arrives with Chapter 14, "Exploring /etc." Everything before "Exploring /etc" in the main text (Chapters 3 through 13) pertains to making FreeBSD work. Everything after "Exploring /etc" pertains to making things work inside FreeBSD. "Exploring /etc" thus serves as a resting point between Absolute FreeBSD's two halves, helping students pause to reflect on all that they have learned while showing how to configure many of FreeBSD's

key settings. As I opened each of the directories in "/etc" and their corresponding man pages, I came to appreciate the extent to which almost every file in this descendant of what Lucas calls "primordial Unix" has its own logical place.

Although Lucas targets information technology workers, I venture that the true audience for *Absolute FreeBSD* is the full range of computer geeks. Someone who has only a soft interest in free and open-source software and/or Unix-like operating systems probably should avoid this strenuous guide, if not FreeBSD in general (Ubuntu GNU/Linux is swell for ordinary use). Yet for me, as a computer-loving, software freedom purist, learning how to make an entire operating system bend to my will was deeply rewarding. Before I started *Absolute FreeBSD*, I spent years building computers and working inside Unix-like operating systems, but I did not have any formal training in IT. Now I never will forget either the purpose of rc.conf or what single-user mode really is, and I have gained myriad skills that have been transferable to my regular work. For professionals and hobbyists alike, *Absolute FreeBSD* promises to be a whimsically engaging learning experience. Whimsical? Lucas's cheeky wit penetrates the whole book.

COREY STEPHAN is a Ph.D. candidate in the Department of Theology at Marquette University in Milwaukee, Wisconsin with a specialization in historical theology. He proudly makes exclusive use of F.O.S.S. tools to assist his research in the history of Christian theology. His professional website is coreystephan.com.

Events Calendar

BSD Events taking place through October 2021

BY ANNE DICKISON

Please send details of any FreeBSD related events or events that are of interest for FreeBSD users which are not listed here to freebsd-doc@FreeBSD.org.

POSI 2021

September 16, 2021
VIRTUAL

The Open Source Initiative will be holding a unique, half-day event on Practical Open Source Information (POSI), which the FreeBSD Foundation will be supporting as an in-kind community sponsor. Targeting individuals and organizations interested in learning what good open-source implementation looks like in practice, POSI brings together a wide array of open source field experts — lawyers, developers, strategists — to share valuable, accessible information on open source best practices. Whether you're new to open source, or are simply seeking ways to improve your existing open source practices, this event may be for you. The event is scheduled to take place on September 16th, 2021, from 11:00 AM to 4:00 PM EDT, and registration is free.

EuroBSDcon 2021

September 16–19, 2021
Vienna, Austria

EuroBSDcon is the European annual technical conference gathering users and developers working on and with 4.4BSD (Berkeley Software Distribution) based operating systems family and related projects.

Open Source Summit 2021

September 27-30, 2021
Seattle, WA + Virtual

Open Source Summit is the leading conference for developers, architects and other technologists — as well as open source community and industry leaders — to collaborate, share information, learn about the latest technologies and gain a competitive advantage by using innovative open solutions.

Open Source Summit connects the open source ecosystem under one roof. It covers cornerstone open source technologies; helps ecosystem leaders to navigate open source transformation with the Diversity Empowerment Summit and tracks on business and compliance; and delves into the newest technologies and latest trends touching open source, including networking, cloud-native, edge computing, AI and much more. It is an extraordinary opportunity for cross-pollination between the developers, sysadmins, DevOps professionals and IT architects driving the future of technology.

This year, Deb Goodkin will be presenting a talk for the FreeBSD Foundation.

2021 Events Calendar

All Things Open 2021

October 17-19, 2021
Raleigh, NC + Virtual

All Things Open is the largest open source/open tech/open web conference on the East Coast, and one of the largest in the United States. It regularly hosts some of the most well-known experts in the world as well as nearly every major technology company. FreeBSD is proud to be a media partner for this year's All Things Open.

FreeBSD Fridays

https://freebsdfoundation.org/freebsd-fridays/
Past FreeBSD Fridays sessions are available at: https://freebsdfoundation.org/freebsd-fridays/

FreeBSD Office Hours

https://wiki.freebsd.org/OfficeHours
Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions.

Past episodes can be found at the FreeBSD YouTube Channel.

Write For Us!

Contact Jim Maurer with your article ideas.

(jmaurer@freebsdjournal.com)