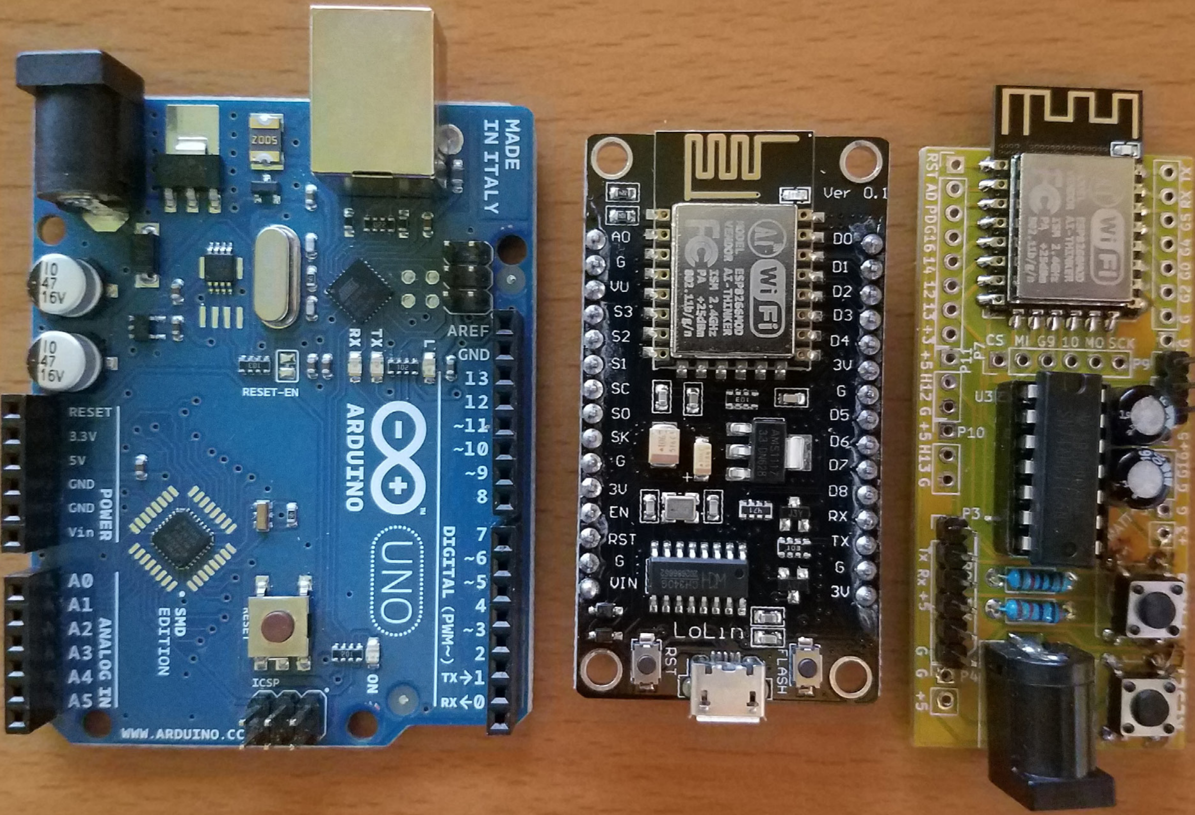
3D Printing and Vendor Lock-in

Text processing in Rust

Downsides to RPi Alternatives

LINUX JOURNAL

Since 1994: The original magazine of the Linux community



SINGLE-BOARD COMPUTERS

Arduino from the Command Line with arduino-cli

A Look at NoodlePi and TinyPi, Two RPi Zero Projects

Set Up the Mycroft Digital Assistant on an RPi 3

Introducing PiBox, a Custom Distro for RPi

ISSUE 296 | MARCH 2019
www.linuxjournal.com

CONTENTS | MARCH 2019 ISSUE 296

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Linux Journal, LLC. Subscription-related correspondence may be sent to 9597 Jones Rd, #331, Houston, TX 77065 USA. Subscription rate is $34.50/year. Subscriptions start with the next issue.

AT YOUR SERVICE

SUBSCRIPTIONS: *Linux Journal* is available as a digital magazine, in PDF, EPUB and MOBI formats. Renewing your subscription, changing your email address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly online: https://www.linuxjournal.com/subs. Email us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Please remember to include your complete name and address when contacting us.

ACCESSING THE DIGITAL ARCHIVE: Your monthly download notifications will have links to the different formats and to the digital archive. To access the digital archive at any time, log in at https://www.linuxjournal.com/digital.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at https://www.linuxjournal.com/contact or mail them to *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 USA. Letters may be edited for space and clarity.

SPONSORSHIP: We take digital privacy and digital responsibility seriously. We've wiped off all old advertising from *Linux Journal* and are starting with a clean slate. Ads we feature will no longer be of the spying kind you find on most sites, generally called "adtech". The one form of advertising we have brought back is sponsorship. That's where advertisers support *Linux Journal* because they like what we do and want to reach our readers in general. At their best, ads in a publication and on a site like *Linux Journal* provide useful information as well as financial support. There is symbiosis there. For further information, email: sponsorship@linuxjournal.com or call +1-281-944-5188.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found online: https://www.linuxjournal.com/author.

NEWSLETTERS: Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on https://www.linuxjournal.com. Subscribe for free today: https://www.linuxjournal.com/enewsletters.

LINUX JOURNAL

EDITOR IN CHIEF: Doc Searls, doc@linuxjournal.com

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

DEPUTY EDITOR: Bryan Lunduke, bryan@lunduke.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

EDITOR AT LARGE: Petros Koutoupis, petros@linux.com

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

SENIOR COLUMNIST: Reuven Lerner, reuven@lerner.co.il

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

DIRECTOR OF DIGITAL EXPERIENCE:
Katherine Druckman, webmistress@linuxjournal.com

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

Linux Journal is published by, and is a registered trade name of,
Linux Journal, LLC. 4643 S. Ulster St. Ste 1120 Denver, CO 80237

SUBSCRIPTIONS
E-MAIL: subs@inuxjournal.com
URL: www.linuxjournal.com/subscribe
Mail: 9597 Jones Rd, #331, Houston, TX 77065

SPONSORSHIPS
E-MAIL: sponsorship@linuxjournal.com
Contact: Publisher Carlie Fairchild
Phone: +1-281-944-5188

LINUX is a registered trademark of Linus Torvalds.

privateinternetaccess®
always use protection®
Private Internet Access is a proud sponsor of *Linux Journal*.

Join a community with a deep appreciation for open-source philosophies, digital freedoms and privacy.

Subscribe to *Linux Journal* Digital Edition for only **$2.88 an issue.**

SUBSCRIBE TODAY!

THE SINGLE-BOARD COMPUTERS ISSUE

Bryan Lunduke is a former Software Tester, former Programmer, former VP of Technology, former Linux Marketing Guy (tm), former openSUSE Board Member...and current Deputy Editor of *Linux Journal* as well as host of the (aptly named) *Lunduke Show*.

When I was a child in the 1980s, I had a computer—a very 1980s computer.

It had a hefty, rectangular, grey case made of some sort of industrial sheet metal. Two plain (but rather large), square buttons adorned the front, begging to be pressed: "Reset" and "Turbo". On the right side of the case, far in the back (nearly out of reach), sat an almost comically large, red power switch. It was the kind of lever that would look right at home in an action movie—used to cut the electricity to all of New York City.

When you "threw the switch", the PC turned on with a deeply satisfying, soul-reverberating, "ka-THUNK".

Inside, sat an Intel 286 CPU decked out with 640k of RAM, which, as some unnamed person may or may not have said, "ought to be enough for anybody". For mass storage, it had a big, double tall hard drive. The connection for this drive wasn't SATA, or SCSI, or even IDE. We're talking about an MFM connection here, baby (MFM stands for Modified Frequency Modulation). As a child, I simply assumed MFM had something to do with the fact that you could hear the hard drive spinning up from down the street.

I kid, I kid. You couldn't actually hear the hard drive—not over the roar of the fan in the power supply.

It was, to say the least, a *beast*—beastly in size, beastly in power usage and beastly in price.

Flash-forward [counts on fingers, gets depressed at own age, downs a pint of ice cream, resumes writing article] 35 years later. We now have single-board computers (SBCs) with no fans—heck, no moving parts whatsoever—running completely silently.

These SBCs have several hundred times (in some cases, several thousand times) the RAM. Ditto for storage. With significantly faster networking (including wireless, which wasn't even a thing on that old 286) and processing speed that, even among the slowest SBCs, is so much faster, it's almost mind-boggling.

All of this is contained within a physical size often smaller than a credit card and at a price somewhere roughly between one hamburger and...a couple more hamburgers.

These small, silent, low-power, low-cost computers have changed things. They've made general-purpose computing more affordable (and durable), bringing down costs in data centers and allowing solo makers and small companies to create computer-driven hardware projects that would have been nearly impossible to tackle in days gone by.

Here in 2019, we've even got a whole heaping helping of SBCs from which to choose: Arduino, BeagleBoard, Gumstix, ODROID, Pine64, Raspberry Pi—the list goes on and on. We are spoiled for choices.

In this issue of *Linux Journal*, we look at SBCs from multiple angles. We start the Deep Dive section with "Arduino from the Command Line: Break Free from the GUI with Git and Vim!"—a detailed look into how to utilize the newly created arduino-cli application to work with Arduino devices directly from, you guessed it, the command line, written by Matthew Hoskins, the Senior Enterprise Architect at New Jersey Institute of Technology.

Then, in "Indie Makers Using Single-Board Computers", I sit down with the minds behind two solo projects based around the Raspberry Pi Zero (the smallest of the Raspberry Pi family): the small, PDA-like NoodlePi and the "oh my gosh, it's like an itty-bitty gameboy" TinyPi.

Next up, Professor Jan Newmarch takes us on a guided tour of how to set up Mycroft (a personal digital assistant that is open-source and Linux-based) on a Raspberry Pi 3—complete with a look at how to build Mycroft "skills" (in this case, interfacing with smart bulbs) using Python.

Turning to the Enterprise space—because these single-board computers aren't just for hobbyist projects—Charles Fisher gives us a review of the newly released Raspberry Pi 3 version of Oracle Linux, including instructions for installing Oracle on the Pi 3 using the Btrfs filesystem.

We finish with a look into PiBox—a custom Linux distribution for the Raspberry Pi, with cross-compiled applications, intended for distributed media playback, written by the man behind the project himself, Michael J. Hammel.

These single-board computers are incredibly versatile. The size and price point alone open up whole worlds of possibilities that seemed out of reach just a few years ago, both in mobile computing and enterprise-grade server farms—truly amazing.

Though, if I'm being honest, I do rather miss that big, red power switch.

Also, now I want a hamburger. ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Thanks to Sponsor

PULSEWAY

for Supporting *Linux Journal*

System Management at Your Fingertips.

www.pulseway.com

The Digital Unconformity

Will our digital lives leave a fossil record? Or any record at all?

By Doc Searls

Doc Searls is a veteran journalist, author and part-time academic who spent more than two decades elsewhere on the *Linux Journal* masthead before becoming Editor in Chief when the magazine was reborn in January 2018. His two books are *The Cluetrain Manifesto*, which he co-wrote for Basic Books in 2000 and updated in 2010, and *The Intention Economy: When Customers Take Charge*, which he wrote for Harvard Business Review Press in 2012. On the academic front, Doc runs ProjectVRM, hosted at Harvard's Berkman Klein Center for Internet and Society, where he served as a fellow from 2006–2010. He was also a visiting scholar at NYU's graduate school of journalism from 2012–2014, and he has been a fellow at UC Santa Barbara's Center for Information Technology and Society since 2006, studying the internet as a form of infrastructure.

In the library of Earth's history, there are missing books. All were written in rock that is now gone. The greatest example of "gone" rock first was observed by John Wesley Powell in 1869, on his expedition by boat through the Grand Canyon. Floating down the Colorado river, he saw the canyon's mile-thick layers of reddish sedimentary rock resting on a basement of gray non-sedimentary rock, and he correctly assumed that the upper layers did not continue from the bottom one. He

knew time had passed between the basement rock and the floors of rock above it, but he didn't know how much. The answer turned out to be more than a billion years. The walls of the Grand Canyon say nothing about what happened during that time. Geology calls that nothing an *unconformity*.

In fact, Powell's unconformity prevails worldwide. The name for this worldwide missing rock is **the Great Unconformity**. Because of that unconformity, geology knows comparatively little about what happened in the world through stretches of time ranging regionally up to 1.6 billion years. All of those stretches end abruptly with the **Cambrian Explosion**, which began about 541 million years ago. Many theories attempt to explain what erased all that geological history, but the prevailing paradigm is perhaps best expressed in **"Neoproterozoic glacial origin of the Great Unconformity"**, published on the last day of 2018 by nine geologists writing for the National Academy of Sciences.

Put simply, they blame snow. Lots of it—enough to turn the planet into one giant snowball, already informally called **Snowball Earth**. A more accurate name for this time would be Glacierball Earth, because glaciers, all formed from snow, apparently covered most or all of Earth's land during the Great Unconformity—and most or all of the seas as well.

The relevant fact about glaciers is that they don't sit still. They spread and slide sideways, pressing and pushing immensities of accumulated ice down on landscapes that they pulverize and scrape against adjacent landscapes, abrading their way through mountains and across hills and plains like a trowel spreading wet cement. Thus, it seems glaciers scraped a vastness of geological history off the Earth's surface and let plate tectonics hide the rest of the evidence. As a result, the stories of Earth's missing history are told only by younger rock that remembers only that a layer of moving ice had erased pretty much everything other than a signature on its work.

I bring all this up because I see something analogous to Glacierball Earth happening right now, right here, across our new worldwide digital sphere. A snowstorm of bits is falling on the virtual surface of that virtual sphere, which itself is made of bits even more provisional and temporary than the glaciers that once covered the physical Earth. All of this digital storm, vivid and present in our current moment in time, is not only doomed to vanish, but it lacks even a glacier's talent for accumulation.

There is nothing about a bit that lends itself to persistence, other than the media it is written on, if it is written at all. Form follows function, and right now, most digital functions, even those we call "storage", are temporary. The largest commercial facilities for storing digital goods are what we fittingly call "clouds". By design, these are built to remember no more of what they contained than does an empty closet. Stop paying for cloud storage, and away goes your stuff, leaving no fossil imprints. Old hard drives, CDs and DVDs might persist in landfills, but people in the far future may look at a CD or a DVD the way a geologist today looks at Cambrian zircons: as signatures of digital activity in a lost period of time. If those fossils speak of what's happening now at all, it will be of a self-erasing Digital Earth that began in the late 20th century.

This isn't my theory. It comes from my wife, who has long claimed that future historians will look on our digital age as an invisible one, because it sucks so royally at archiving itself. I think she's right.

For example, this laptop currently sits atop a stack of books on my desk. Two of those books are self-published compilations of essays I wrote about technology in the mid-1980s, mostly for publications that are long gone. The originals are on floppy disks that can be read only by PCs and apps of that time, some of which are buried in lower strata of boxes in my garage. I just found a floppy with some of those essays. (It's the one with a blue edge in the wood case near the right end of the photo.) But I'll need to find an old machine to read it. Old apps too, all of them also on floppies. If those still retain readable files, I am sure there are ways to recover at least the raw ASCII text. But I'm still betting the paper copies of the books under this laptop will live a lot longer than the floppies or the stored PCs will—if those aren't already bricked by decades of un-use.

As for other media, the prospect isn't any better.

At the base of my video collection is a stratum of VHS videotapes, atop of which are strata of Video8 and Hi8 tapes, and then one of digital stuff burned onto CDs and stored in hard drives, most of which have been disconnected for years. Some of those drives have interfaces and connections no longer supported by any computers being made today. Although I've saved machines to play all of them, none I've checked still work. One choked to death on a CD I stuck in it. And that was just one failure among many that stopped me from making Christmas presents of family memories recorded on old tapes and DVDs. I may take up the project again sometime before next Christmas, but the odds are long against that (hey, I'm busy) and short toward the chance that nobody will ever see or hear those recordings again. And I'm not even counting my parents' 8mm and 16mm movies made from the 1930s to the 1960s. In 1989, my sister and I had all of those copied over to VHS tape. We then recorded my mother annotating the tapes onto companion cassette tapes while we watched the show. I still have the original film in a box somewhere, but I can't find any of the tapes.

The base stratum of my audio past is a few dozen open reel tapes recorded in the 1950s and 1960s. Above that are cassette and microcassete tapes, plus some Sony MiniDisks recorded in ATRAC, a proprietary and incompatible compression algorithm now used by nobody, including Sony. Although I do have ways to play some (but not all) of those, I'm cautious about converting any of them to digital formats (Ogg, MPEG or whatever), because all digital storage media becomes obsolete, dies, or both—as do formats, algorithms and codecs. Already I have dozens of dead external hard drives in boxes and drawers. And no commercial cloud service is committed to digital preservation in perpetuity in the absence of payment. This means my saved files in clouds are sure to be flushed after neither my heirs nor I continue paying for their preservation.

Same goes for my photographs. My old photographs are stored in boxes and albums of photos, negatives and Kodak slide carousels. My digital photographs are spread across a mess of duplicated back-up drives totaling many terabytes, plus a handful of CDs. About 60,000 photos are exposed to the world on Flickr's cloud, where I maintain two Pro accounts (**here** and **here**) for $50/year a piece. More are in the Berkman Klein Center's pro account (**here**) and *Linux Journal*'s (**here**). It is unclear currently whether any of that will survive after any of those entities stop paying the yearly fee. SmugMug, which now owns Flickr, has said **some encouraging things** about photos such as mine, all of which are **Creative Commons**-licensed to encourage re-use. But, as **Geoffrey West** tells us, companies are mortal. All of them die.

As for my digital works as a whole (or anybody's), there is great promise in what the Internet Archive and Wikimedia Commons do, but there is no guarantee that either will last for decades more, much less for centuries or millennia. And neither are able to archive everything that matters (much as they might like to).

It also should be sobering to recognize that we only rent the "sites", "locations" and "addresses" at "domains" that we talk about "owning". The plain and awful fact is that nobody "owns" a domain on the internet. They pay a sum to a registrar for the right to use a domain name for a finite period of time. There are no permanent domain names or IP addresses. In the digital world, finitude rules.

So the historic progression I see, and try to illustrate in the photo at the beginning of this article, is from hard physical records through digital ones we hold for ourselves, and then up into clouds that go away, inevitably. Everything digital is snow falling and disappearing on the waters of time.

Will there ever be a way to save for the very long term what we ironically call our digital "assets" for more than a few dozen years? Or is all of it doomed by its own nature to disappear, leaving little more evidence of its passage than a digital unconformity?

I can't think of any technical questions more serious than those two. ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

WITI

25TH ANNUAL

WOMEN IN TECHNOLOGY

SUMMIT

BECOME CONNECTED COACHED INSPIRED

For 25 years running, the Women in Technology Summit has served as the premier gathering for more than 1,500 women and men from around the world, who converge in Silicon Valley for three special days to build relationships, collaborate, and update their technical and leadership skills.

JUNE 9-11, 2019 | SAN JOSE, CA

Women in Technology International, the leading advocate for innovation, inclusivity and STEAM, is celebrating 30 years of supporting and fostering women in technology.

WITI30 30TH ANNIVERSARY

REGISTER AT WITI.COM/SUMMIT

LETTERS

The Light at the End of the Pipe

I read Kyle Rankin's "Put Down the Pipe" article from the January 2019 issue with interest, as I'm a big fan of Bash and love to learn new tricks. I can't help feeling that the last example using `find` missed one of the most powerful options to `find`: `-exec`. This is a built-in `xargs` replacement and is very powerful. One of my favourite uses is this:

```
find . -name "*.c" -exec grep "thing" {} /dev/null \;
```

This searches all ".c" files beneath the current working directory for "thing" and outputs the name of the file and the line containing "thing". The `{}` is replaced by the files found, one by one. `/dev/null` forces `grep` to output the name of the file containing `thing`, as it doesn't otherwise do so if there's only one file passed to it. The beauty of `-exec` is that it can be used with any other shell command. The above example improves on `grep -R` as there are many other options that can be used with `find` to filter the files searched (for example, `-type`).

—Steve Kerr

Thanks for "Have a Plan for Netplan"

I just wanted to thank Shawn Powers for his article "Have a Plan for Netplan" in the October 2018 issue. I also wanted to mention a configuration detail that may be related to the peculiar DNS mis-behaviors that Shawn reported in the article. As it turns out, similar mis-behaviors also had been a problem for my network configuration.

My Xubuntu 18.04 desktop system uses a Netplan YAML file with networkd as the renderer. I installed Stubby with the intent of using DNS over TLS for the name service. However, when I first applied the Netplan configuration, I also was observing (uncontrollable) DNS-resolution behaviors similar to those reported in the article. After reading the article, I realized that I could disable the systemd-resolved service and allow the Stubby resolver to exclusively take over the DNS-resolution function. This approach ultimately solved my problems—thanks!

Being a Linux-newbie, it has taken considerable time and effort to figure out what (I think) was happening. I'll provide a dialog below to describe what I did and what may be the cause of the problems that led to the DNS mis-behaviors.

To start, it is not readily apparent that systemd-resolved provides sufficient controllability from Netplan—for preventing automatic access to the gateway's DNS server.

For example, configuring with NetworkManager via the GNOME desktop allows for invocation of a "DNS Automatic OFF" control. This option is available from the NetworkManager configuration GUI, but I couldn't find a similar directive for Netplan. I'm thinking that NetworkManager has the necessary "hooks" for configuring the systemd-resolved service with "DNS Automatic OFF", whereas Netplan does not?

There's a possibly related Netplan bug report **here**.

I used **this web page** for installing Stubby on Ubuntu 18.04, which uses NetworkManager and sets "DNS Automatic OFF" via the GNOME desktop.

I performed all installation steps up to the section titled "Switching to Stubby", which describes configuration using NetworkManager in that section.

Of course, for my implementation, the reference in the YAML file, setting the renderer to "NetworkManager" was removed and replaced with networkd.

I found that using netplan and running stubby as the replacement stub resolver can be made to work if systemd-resolved is disabled, the netplan-directed nameserver is set to 127.0.0.1, and stubby is configured to provide encrypted DNS over TLS. Of course, Stubby is configured to use appropriate DNS servers that will support DNS over TLS service.

Again, thanks for the article!

—Jon

Shawn Powers replies: I'm glad the DNS bit helped! I recently found a glitch (feature? LOL) with my Ubuntu MATE desktop install. I followed my own article, because it's not the sort of thing you do often enough to remember all the details, and I discovered that while I had to use "DNS" in the NetworkManager.conf file to make it work when I wrote the article, I actually had to use "dns" in my current setup. Whoever invented capital letters should be slugged in the arm.

I'll have to check out Stubby. The only other DNS tool I've used on the desktop is DNSMasq, which I've also written about extensively. Stubby is new to me, and one more thing I'll have to investigate! Thanks!

Re: Dave Taylor and Roman Numerals

I always enjoy Dave Taylor's immensely useful Bash column, but I think his latest series over-complicates the problem (see Part I, Part II and Part III of Dave's Roman Numerals and Bash series).

Converting to Roman numerals can be done by taking each of the one- and two-letter values you list and repeatedly subtracting them in one go—no need to repeat the whole table. This is effectively the "greedy algorithm" approach to change-making.

I also dislike having to use lots of similar if-blocks, so I'd rather jump through some regex hoops to avoid it. See my suggestion below for a more succinct solution:

```
#!/bin/bash
#
# toroman
#
DECVALUE="$1"
ROMANVALUE=""
TABLE="M=1000 CM=900 D=500 CD=400 C=100 XC=90"
TABLE+=" L=50 XL=40 X=10 IX=9 V=5 IV=4 I=1"
for T in $TABLE ; do
  R=${T/=*/} ; D=${T/*=/}
  while [[ $DECVALUE -ge $D ]]; do
    ROMANVALUE+=$R
    let DECVALUE-=$D
  done
done
echo $ROMANVALUE
```

—Mike

Dave Taylor replies: Thanks for your note, Mike. I have to admit that there's something about the world of regular expressions that is still daunting. And trying to explain a complicated regex to someone who isn't familiar with the concept (my primary reader) is more than a bit overwhelming. Still, it's a graceful solution. Where were you when I was a compsci undergrad at UCSD?

Possible Article Suggestion

Regarding Kyle Rankin's "Back to Basics: Sort and Uniq" from the January 2019 issue: I am a relative newbie to Linux, about five years or so, on my personal computer at home, so I enjoyed this article on going back to basics. Understanding commands and how to use them well is how one gets more comfortable with the system.

I was thinking an article on how to read `man` pages might be a good idea. I think I may have read something on it in the past, but understanding the basics and how to understand fully how to run commands might help others too.

—Patrick Donahue

Bad Language in the Kernel Code

Ricky Gervais put it succinctly: "Nobody has the RIGHT to NOT be offended" (my emphasis).

Yes, if you write a comment that says "xyz is a fucking idiot"—it's probably inappropriate to attack someone explicitly; that's basically just rude. On the other hand, if you write a comment that says, "This chunk of code is fucking awful" (and you can say why), then why not? They're just words to emphasize your opinion of the code.

Complaints about bad language are a red herring. The real issue is confusing who people are (male, female, black, white, gay, other, etc.) with what they believe (homophobic, or not, religious, atheist, pro-life, pro-choice, Republican, Democrat, perl is a good thing, etc.). Now everyone, please, get a fucking life!

—David Jameson

The Security Issue

There was a great deal of good information in the February 2019 issue. As a retired individual having a bit more time than I used to have, I have taken on the task of removing everything I can from "the cloud". My reasons are not that important, but suffice it to say that I dislike any of my data being anywhere but on my hardware unless I specifically send it to someone/some-group. At the same time, I want access to my data on all my devices regardless of my location. Thus, I established my own cloud using Nextcloud. Doing so can be a daunting task once the rather simple initial setup is complete. Security is of great importance.

There are a number of basic tasks in the February 2019 issue that I have not taken on, but that I should. So thank you much for an excellent selection of applicable articles!

As a side note: Passman, which is an add-on to Nextcloud does a pretty darn good job of handling passwords. There is a plugin for browsers that works quite well connecting to and retrieving passwords from the encrypted database on the server. Also, there is an app for mobile devices in beta as well.

Finally, "Open Science, Open Source and R" by Andy Wills is very pertinent from my perspective, as it addresses a rather disturbing problem of the real and/or perceived veracity (or lack thereof) of some research and the broader problem of the recent attacks on science. Only the scientific community can address this issue, and the author does a good job of showing one way in which that can be done. It is critically important that we re-establish our confidence in science and scientific methodology, and doing so means the scientific community is going to have to re-dedicate itself to a higher level of discipline.

I also want to mention that I agree fully with Glyn Moody (see "If Software Is Funded from a Public Source, Its Code Should Be Open Source"). Well said!

And more generally, as usual, *Linux Journal* delivers! Well done!

—Wesley J. Wieland

From Social Media

In Response to Receiving a Free Issue of *LJ*:

Danny Hernandez @danny13_33: Aww Yeah! Thank you @linuxjournal for the subscription to your magazine! Just a quick look and there are tons of useful articles that I can apply to my everyday SysAdmining. #Linux #DevOps #LinuxJournal #SysAdmin

In Response to Bryan Lunduke's "Why Is Linux Spelled Incorrectly":

Steeve McCauley: oh god no, that's worse than linux. I used to think it should be linix, but I gave that up 20+ years ago.

Jason Lisonbee: It's possible the person who first called it Linux thought of your spelling but the "u" made it more clear that the commands and behavior were intended to be naturally familiar to Unix users. 'Ux' can be short hand for 'unix'.

Rob Kingsboro: GNU+LINICS

Chris Franz: "Linux" would be too close to Minix for Torvalds' taste, I suspect.

Still surprises me that there is actual audio of how the man himself pronounces it—he's the foremost authority given that it's based on his own damn name—

and people still intentionally pronounce it LIE-nux.

Jason Lisonbee: IIRC someone else mashed the name Linus with Unix to call it Linux. Just like someone other than Linus witnessed or learned that he had some incident with a penguin and made that the mascot; then or later calling it Tux, which is now the name of some derivatives, including probably a distro and a filesystem, which AFAIK are distinct and separate products/projects.

Mihalej Archnalej Korcak: it's LINUKS!

Abdullah Leghari: In Asia it's pronounced like Lienuks or Lyenuks.

Rob Kingsboro: Well, that was interesting. Arbitrarily putting Xs in things is corny.

SEND *LJ* A LETTER *We'd love to hear your feedback on the magazine and specific articles. Please write us here or send email to ljeditor@linuxjournal.com.*

PHOTOS *Send your Linux-related photos to ljeditor@linuxjournal.com, and we'll publish the best ones here.*

LinuxFest Northwest

April 26 - 28, 2019 • Bellingham Technical College

20th Anniversary! "Past, Present & Future"

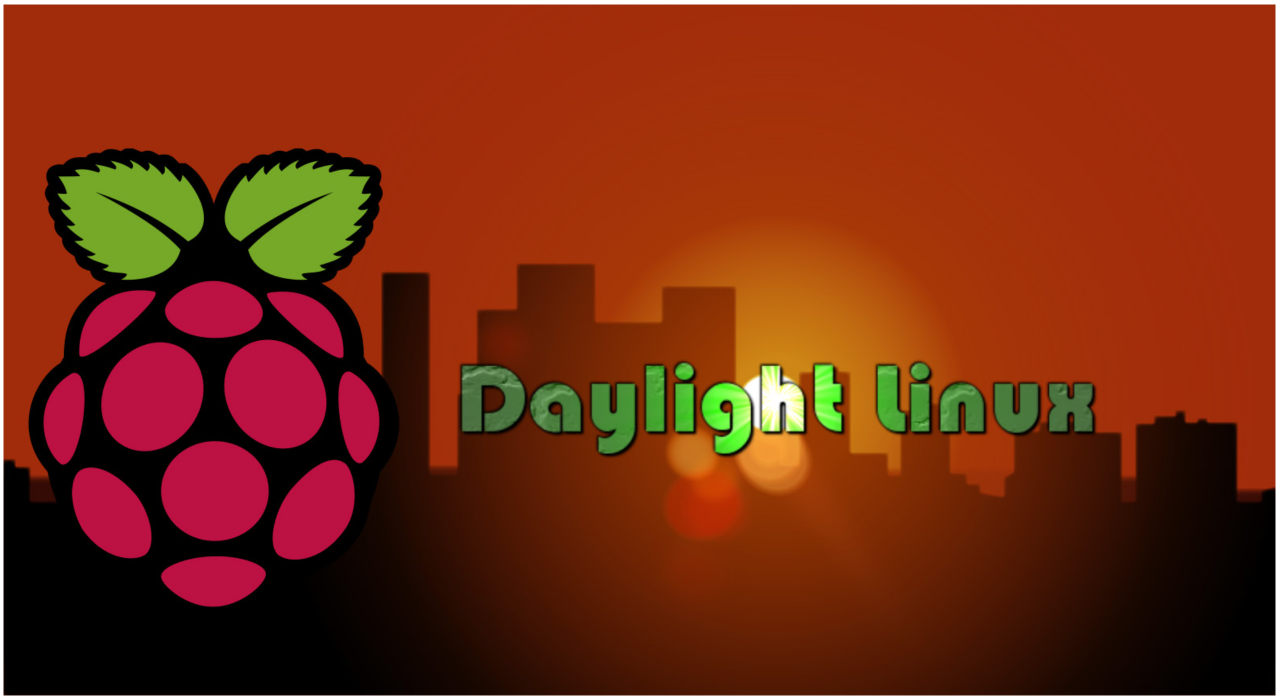
1800+ attendees • 100+ sessions • 40+ vendors
Open hardware • Social events • Job fair • World famous raffle
All free to attend

Bellingham TECHNICAL COLLEGE

Bellingham BLUG Linux User Group

LinuxFest Northwest (est. 2000), an annual Open Source event co-produced by *Bellingham Linux Users Group* and the *Information Technology department at BTC*. LFNW features presentations and exhibits on free and open source topics, as well as Linux distributions & applications, InfoSec, and privacy; something for everyone from the novice to the professional!

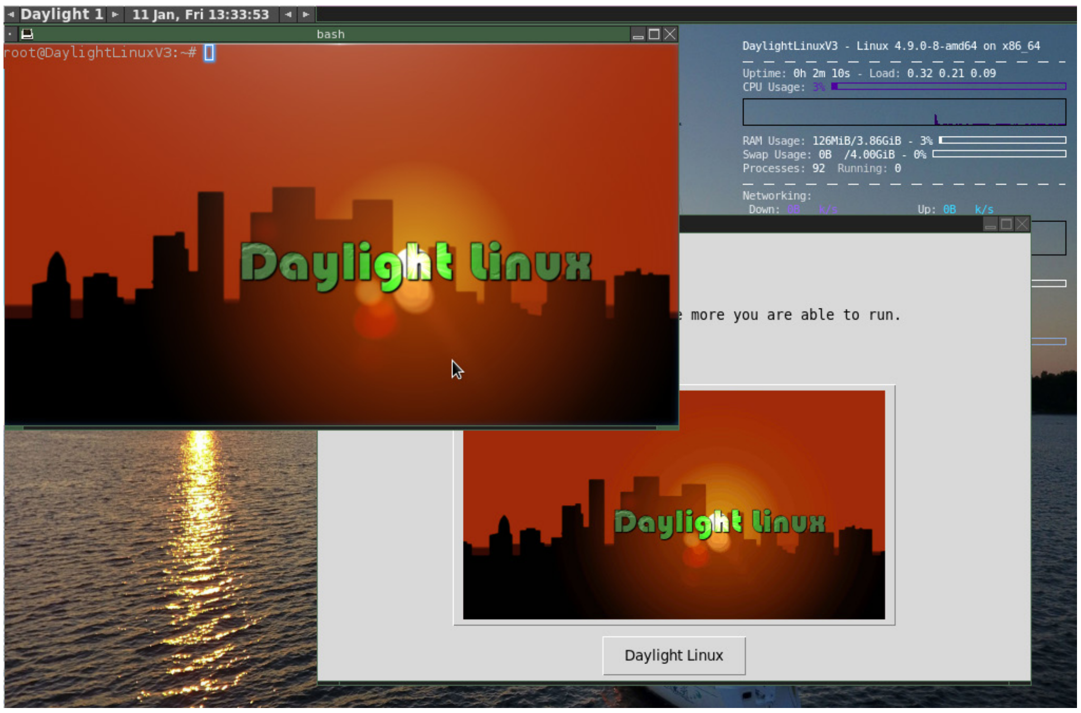
https://blog.lfnw.org linuxfestnorthwest @lfnw linuxfestnorthwest info@linuxfestnorthwest.org



FOSS Project Spotlight: Daylight Linux Version 3

Daylight Linux is the only official distribution for the Raspberry Pi to work with the Fluxbox interface. With Fluxbox, Daylight Linux is one of the lightest and fastest distributions for all Raspberry Pi models.

Many programs, games and system tools were developed during a long year of work in Python 3 to create version 3.

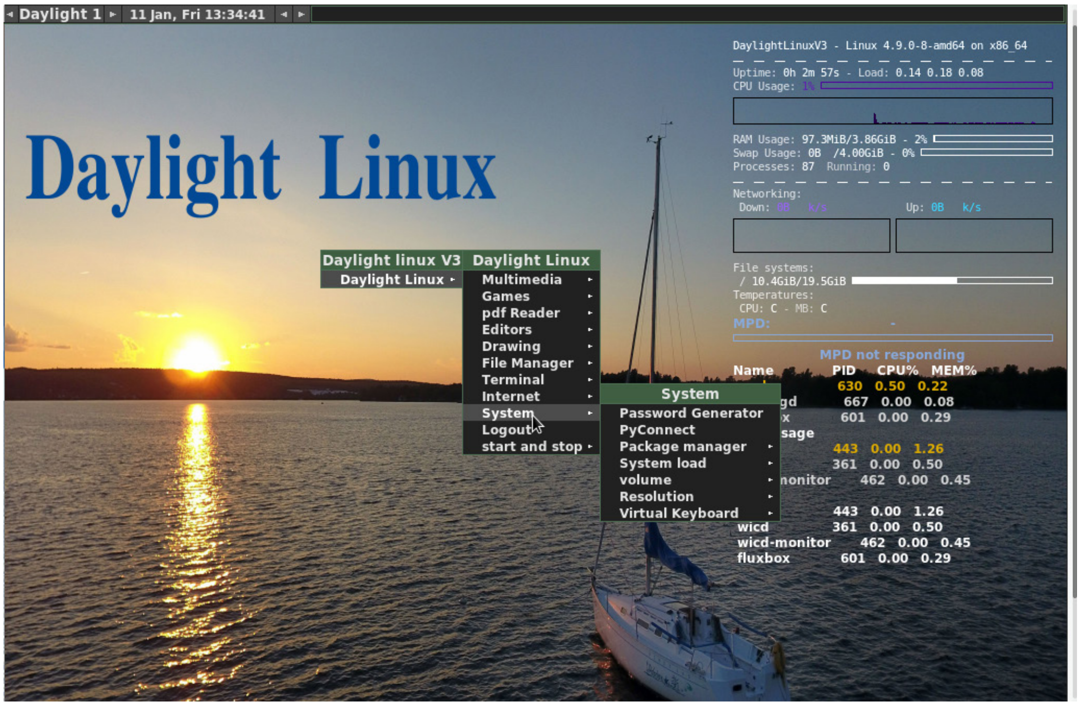


```
root@DaylightLinuxV3:~#
```

```
DaylightLinuxV3 - Linux 4.9.0-8-amd64 on x86_64
Uptime: 0h 2m 10s - Load: 0.32 0.21 0.09
CPU Usage: 3%
RAM Usage: 126MiB/3.86GiB - 3%
Swap Usage: 0B  /4.00GiB - 0%
Processes: 92  Running: 0
Networking:
 Down: 0B   k/s              Up: 0B   k/s
```

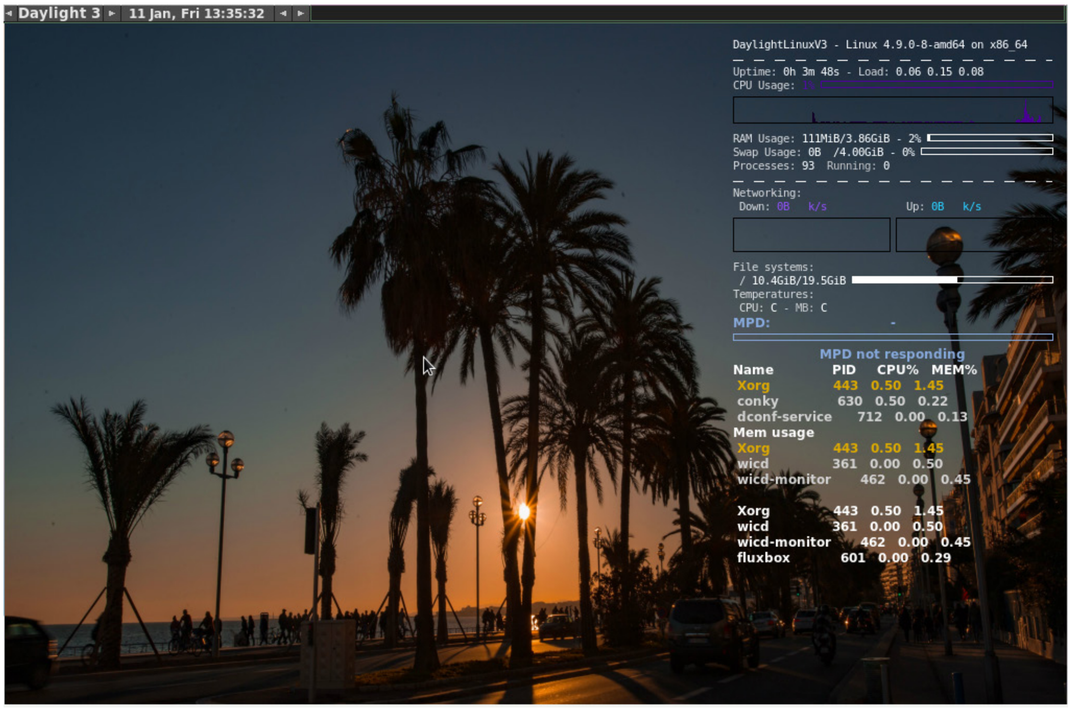
```
more you are able to run.
```

Figure 1. The System at Boot



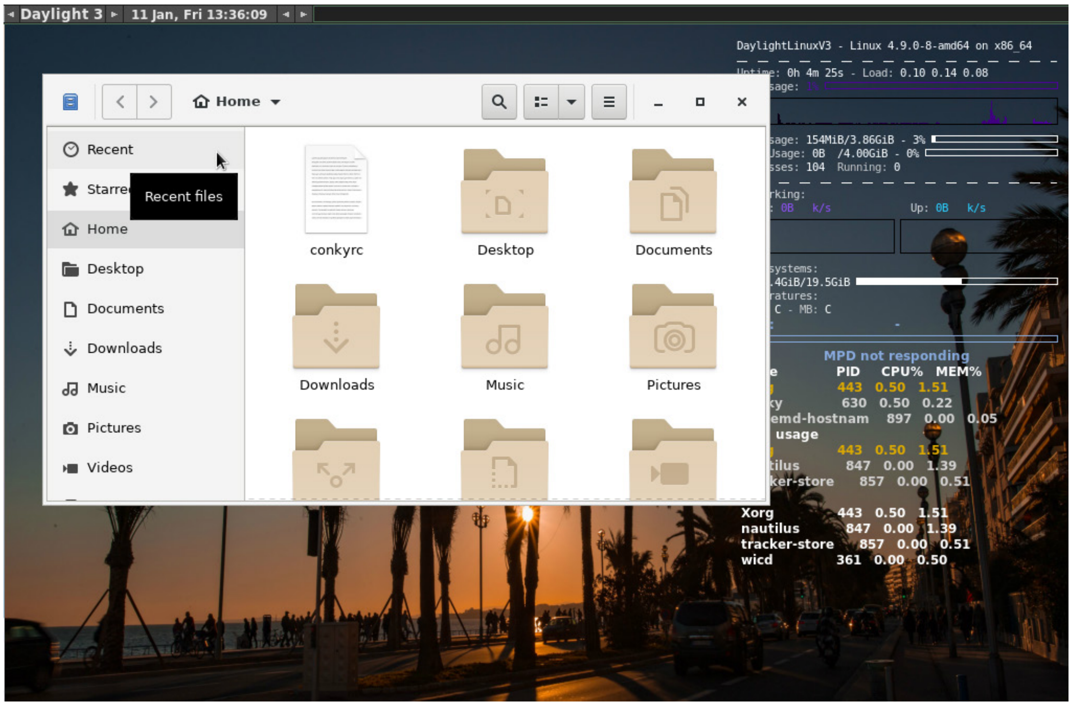
```
DaylightLinuxV3 - Linux 4.9.0-8-amd64 on x86_64
Uptime: 0h 2m 57s - Load: 0.14 0.18 0.08
CPU Usage: 1%
RAM Usage: 97.3MiB/3.86GiB - 2%
Swap Usage: 0B  /4.00GiB - 0%
Processes: 87  Running: 0
Networking:
 Down: 0B   k/s              Up: 0B   k/s
File systems:
 / 10.4GiB/19.5GiB
Temperatures:
 CPU: C - MB: C
MPD:            -
          MPD not responding
Name            PID   CPU%   MEM%
                630   0.50   0.22
   gd            667   0.00   0.08
   x            601   0.00   0.29
   sage
                443   0.00   1.26
                361   0.00   0.50
   nonitor        462   0.00   0.45

                443   0.00   1.26
 wicd           361   0.00   0.50
 wicd-monitor      462   0.00   0.45
 fluxbox         601   0.00   0.29
```

Figure 2. The Daylight Linux Menu



```
DaylightLinuxV3 - Linux 4.9.0-8-amd64 on x86_64
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Uptime: 0h 3m 48s - Load: 0.06 0.15 0.08
CPU Usage: 1%

RAM Usage: 111MiB/3.86GiB - 2%
Swap Usage: 0B  /4.00GiB - 0%
Processes: 93  Running: 0
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Networking:
 Down: 0B   k/s                Up: 0B   k/s

File systems:
 / 10.4GiB/19.5GiB
Temperatures:
 CPU: C - MB: C
MPD:                    -

           MPD not responding
Name             PID    CPU%   MEM%
 Xorg            443   0.50   1.45
 conky           630   0.50   0.22
 dconf-service     712   0.00   0.13
Mem usage
 Xorg            443   0.50   1.45
 wicd            361   0.00   0.50
 wicd-monitor      462   0.00   0.45

 Xorg            443   0.50   1.45
 wicd            361   0.00   0.50
 wicd-monitor      462   0.00   0.45
 fluxbox          601   0.00   0.29
```

Figure 3. The Daylight Linux Desktop with System Information



```
DaylightLinuxV3 - Linux 4.9.0-8-amd64 on x86_64
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
Uptime: 0h 4m 25s - Load: 0.10 0.14 0.08
sage: 1%

sage: 154MiB/3.86GiB - 3%
Usage: 0B  /4.00GiB - 0%
sses: 104  Running: 0
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
rking:
: 0B   k/s                Up: 0B   k/s

systems:
.4GiB/19.5GiB
ratures:
 C - MB: C
:                    -

           MPD not responding
e                PID    CPU%   MEM%
g                443   0.50   1.51
ky               630   0.50   0.22
emd-hostnam      897   0.00   0.05
 usage
g                443   0.50   1.51
tilus            847   0.00   1.39
ker-store          857   0.00   0.51

 Xorg            443   0.50   1.51
 nautilus         847   0.00   1.39
 tracker-store      857   0.00   0.51
 wicd            361   0.00   0.50
```

Figure 4. The Daylight Linux File Manager

The system works with autologin, but you also can use these login/passwords: “root”/”toor” and “Daylight”/”toor”.

A live version also is available for computers. This version aims to provide Debian-based Linux with the lightness of Daylight Linux.

Daylight Linux version 3 runs on all Raspberry Pi models, and it’s based on Debian Buster. Visit the official website for more information and to download.

—Hamdy Abou El Anein

Antennas in Linux

For this article, I want to introduce a piece of software I've actually used recently in my own work. My new day job involves studying the ionosphere using an instrument called an ionosonde. This device is basically a giant radio transmitter that bounces radio waves off the ionosphere to see its structure and composition. Obviously, an important part of this is knowing the radiation pattern of the various transmitters and receivers.

Several methods exist for modeling the electromagnetic fields around conductors, but here I'm covering one called NEC2 (Numerical Electromagnetics Code). It originally was developed in FORTRAN at the Lawrence Livermore National Laboratory in the 1970s. Since then, it's been re-implemented several times in various languages. Specifically, let's look at xnec2c. This package implements NEC2 in C, and it also provides a GTK front end for interacting with the core engine.

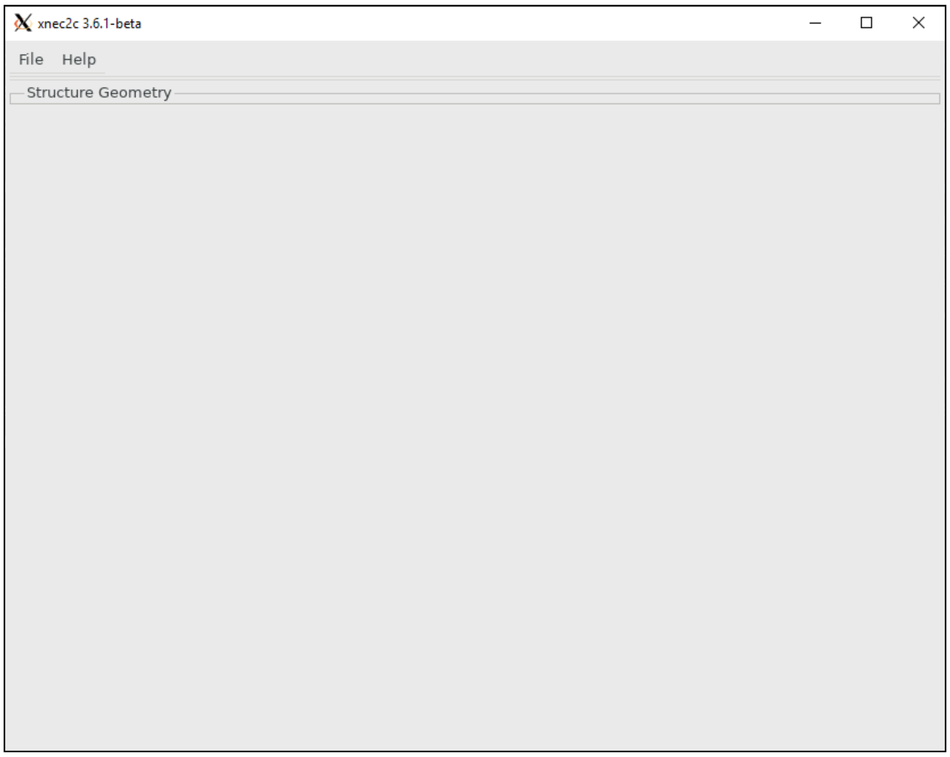


Figure 1. Launching xnec2c gives you a pretty boring starting point.

xnec2c should be available in most Linux distributions. In Debian-based distributions, you can install it with the command:

```
sudo apt-get install xnec2c
```

Once it's installed, you can start it with `xnec2c`. The default display doesn't show anything until you actually start using it.

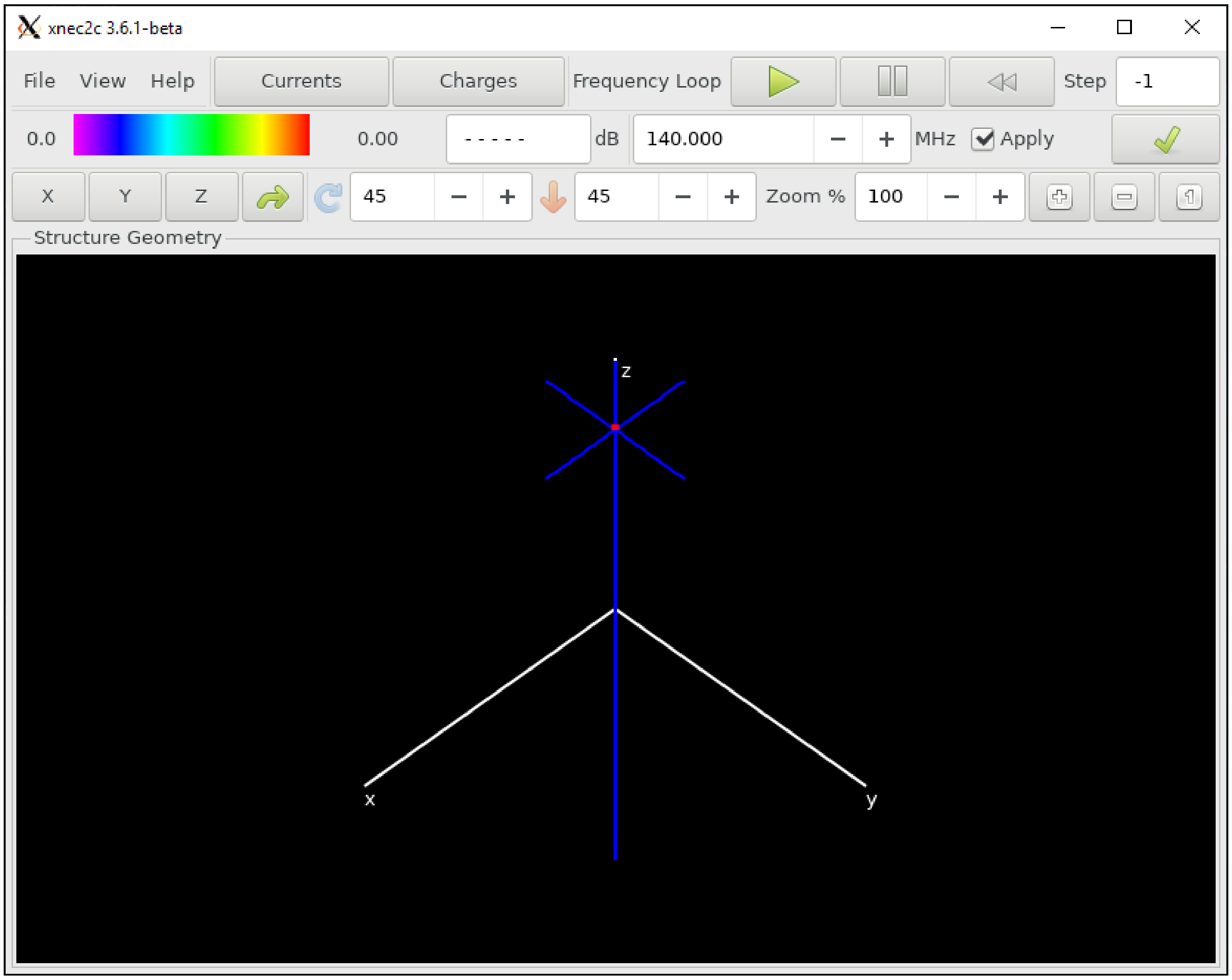


Figure 2. Loading an input file, you begin with a geometric view of the relevant antenna wires, other conductors and any ground planes.

xnec2c's history still affects how it behaves to the present day. This is most clear when you look at the input file's format. The basic structure is based on the idea of a punch card, where each "command" to xnec2c is given by a command card—a definite holdover from its FORTRAN roots. Luckily, the GTK front end to xnec2c provides a reasonably functional way of building up these input files.

Several example files should be available with your installation of xnec2c. In my Ubuntu distribution, they're located in /usr/share/doc/xnec2c/examples. These input files have a filename ending of ".nec". Select one as a starting off point to play with xnec2c, and then go ahead and make the required alterations necessary for your own project.

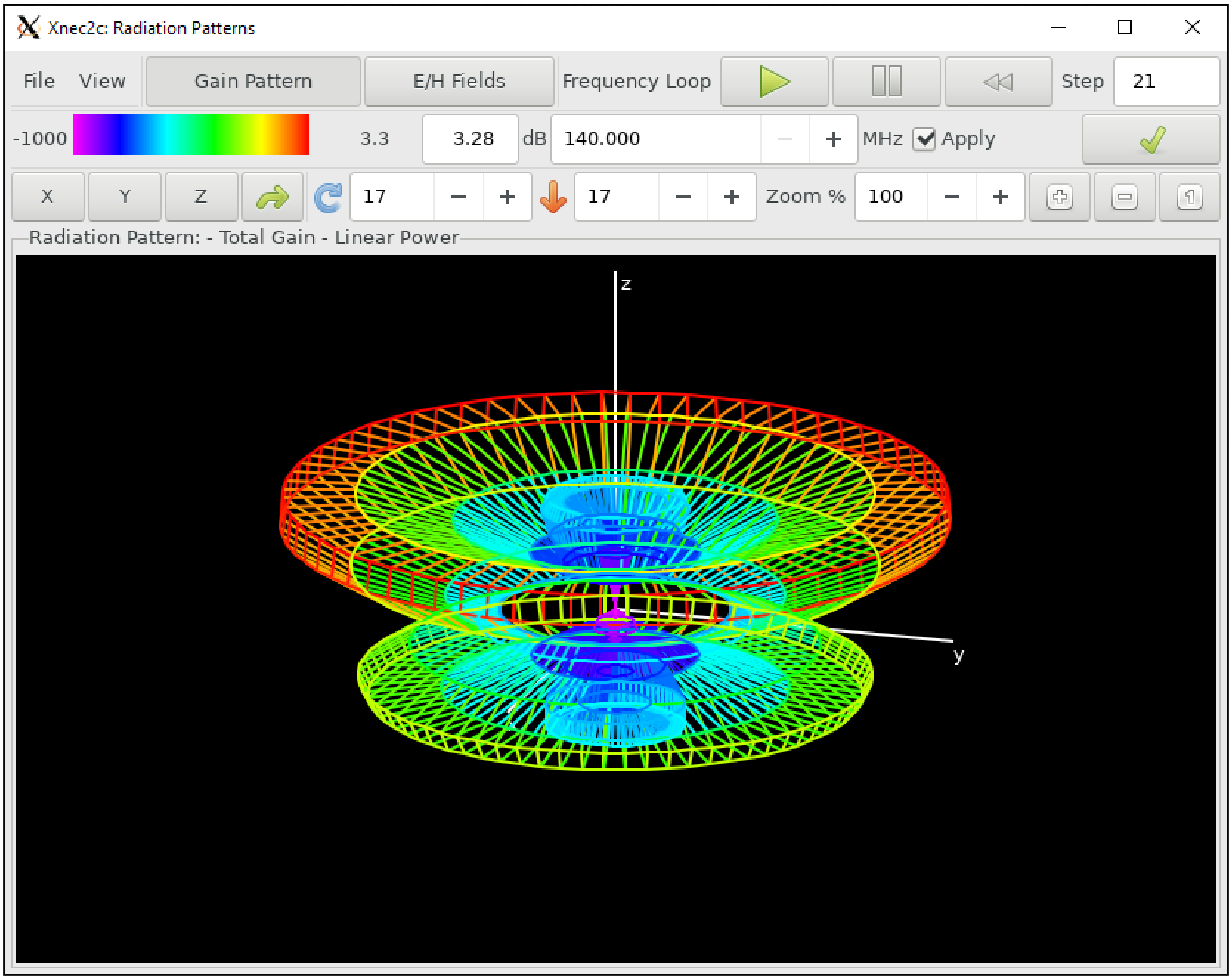


Figure 3. You easily can model the radiation pattern from an antenna system.

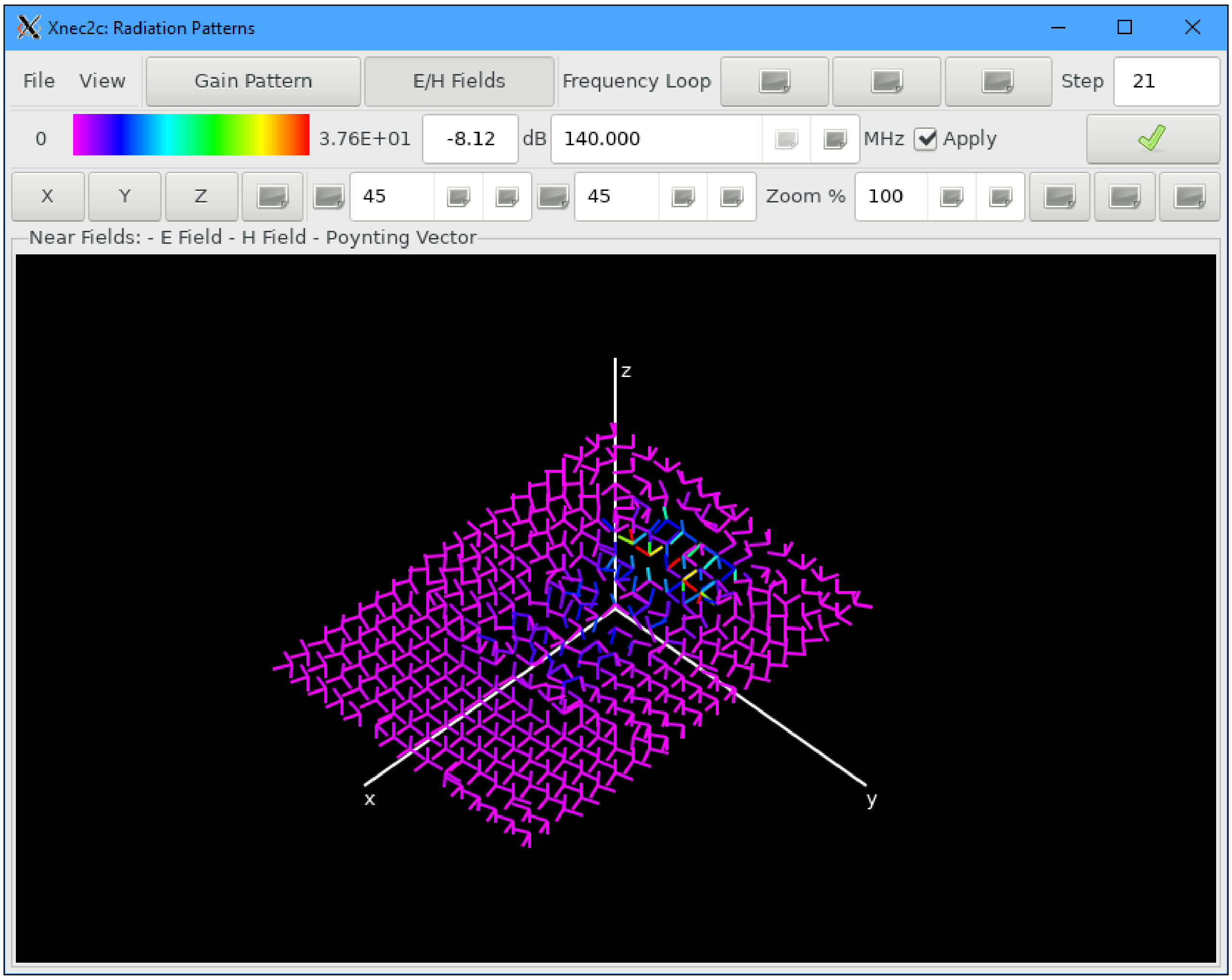


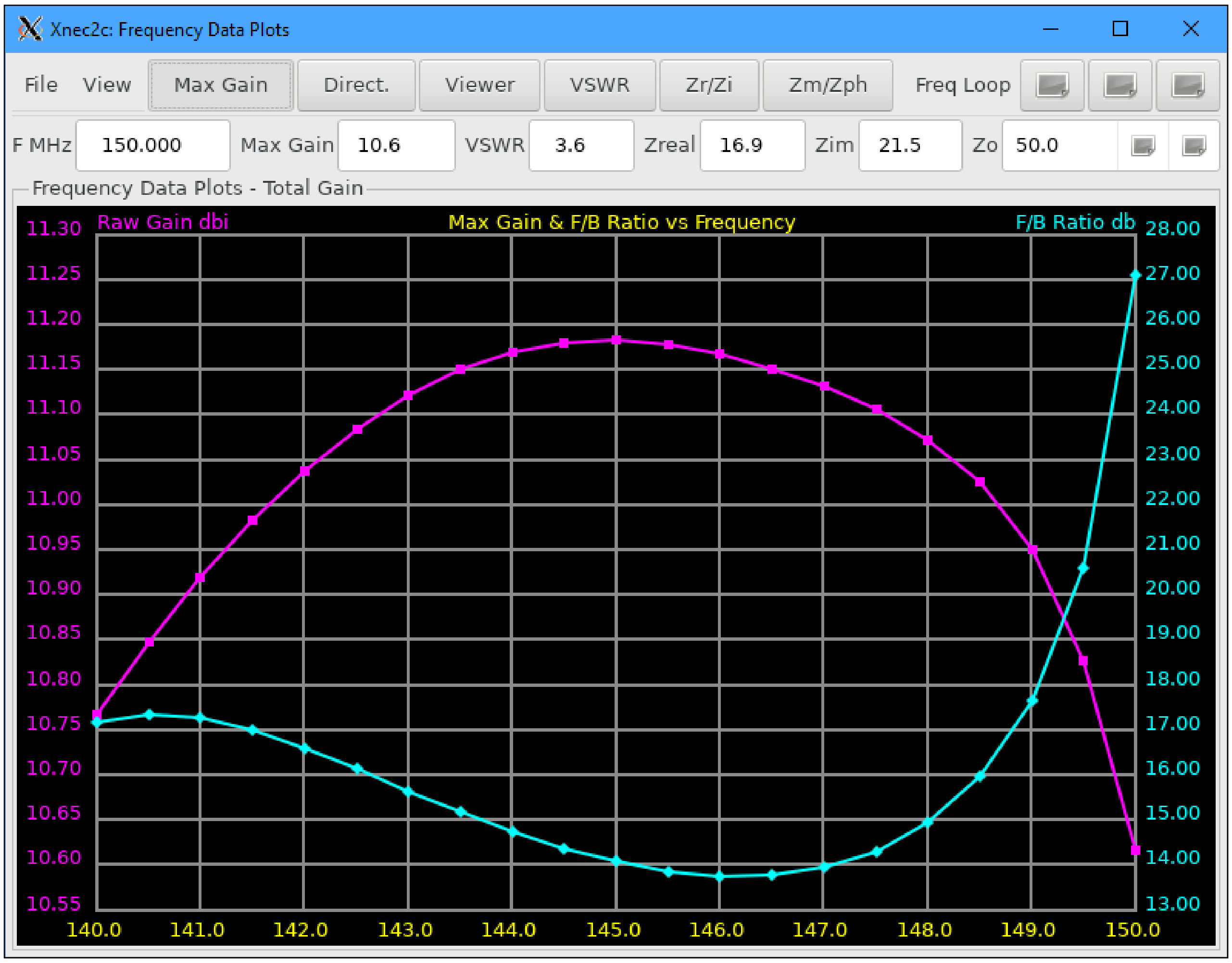
Figure 4. If your input file has the appropriate entries, you can see a display of the electric and magnetic fields.

The central window pane provides a geometric view of the actual antenna structure in three dimensions. You can click and drag the diagram to rotate the view and see it from all angles. There are two larger buttons at the top of the window, named Currents and Charges. Selecting them alternately will show either the distribution of currents or the distribution of charges caused by the driving current.

Once you're sure the physical layout is correct, you can start to see what the electromagnetic behavior of the system looks like. On the menu bar at the top, there is an entry named View. Clicking "Radiation Pattern" pops up a new window where you can view what the electromagnetic radiation pattern looks like. A few different options are available. The first, "Gain Pattern", provides what most people probably think of as the radiation pattern.

Again, you can click and drag the image to get a better view of the pattern from various angles.

In the same window, there is a second display option. Assuming your input file has the appropriate entries, click the "E/H Fields" button to see a view of the electric and/or magnetic fields for your configuration.



F MHz 150.000 Max Gain 10.6 VSWR 3.6 Zreal 16.9 Zim 21.5 Zo 50.0

Figure 5. You can get several detailed plots of the electrical behavior of your antenna design.

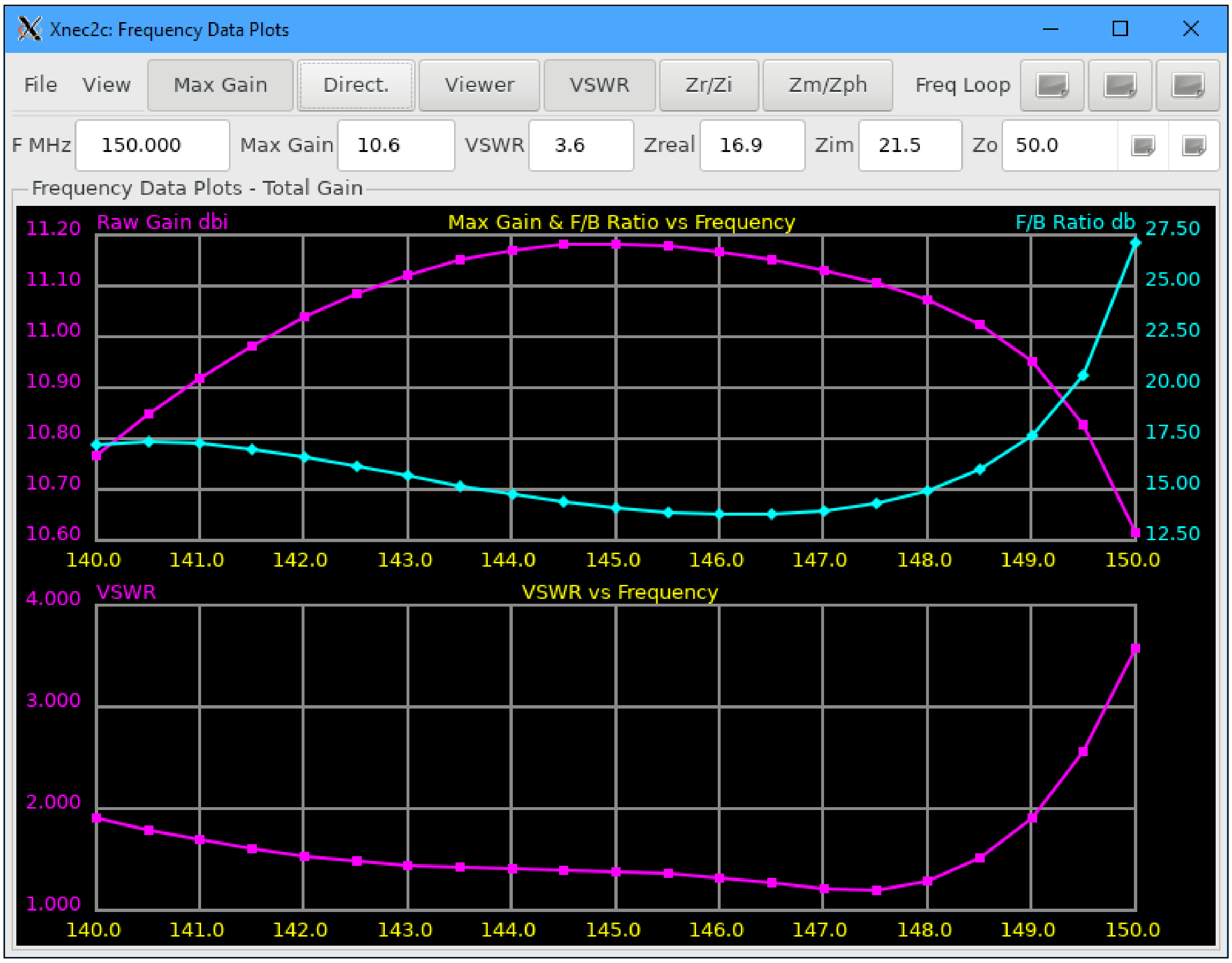


Figure 6. You can see multiple plots of the electrical behavior stacked on top of each other to get a better overall view of your antenna configuration.

Several options are available at the top of the window. For example, you can change the frequency of the driving current to see how your system behaves at different frequencies.

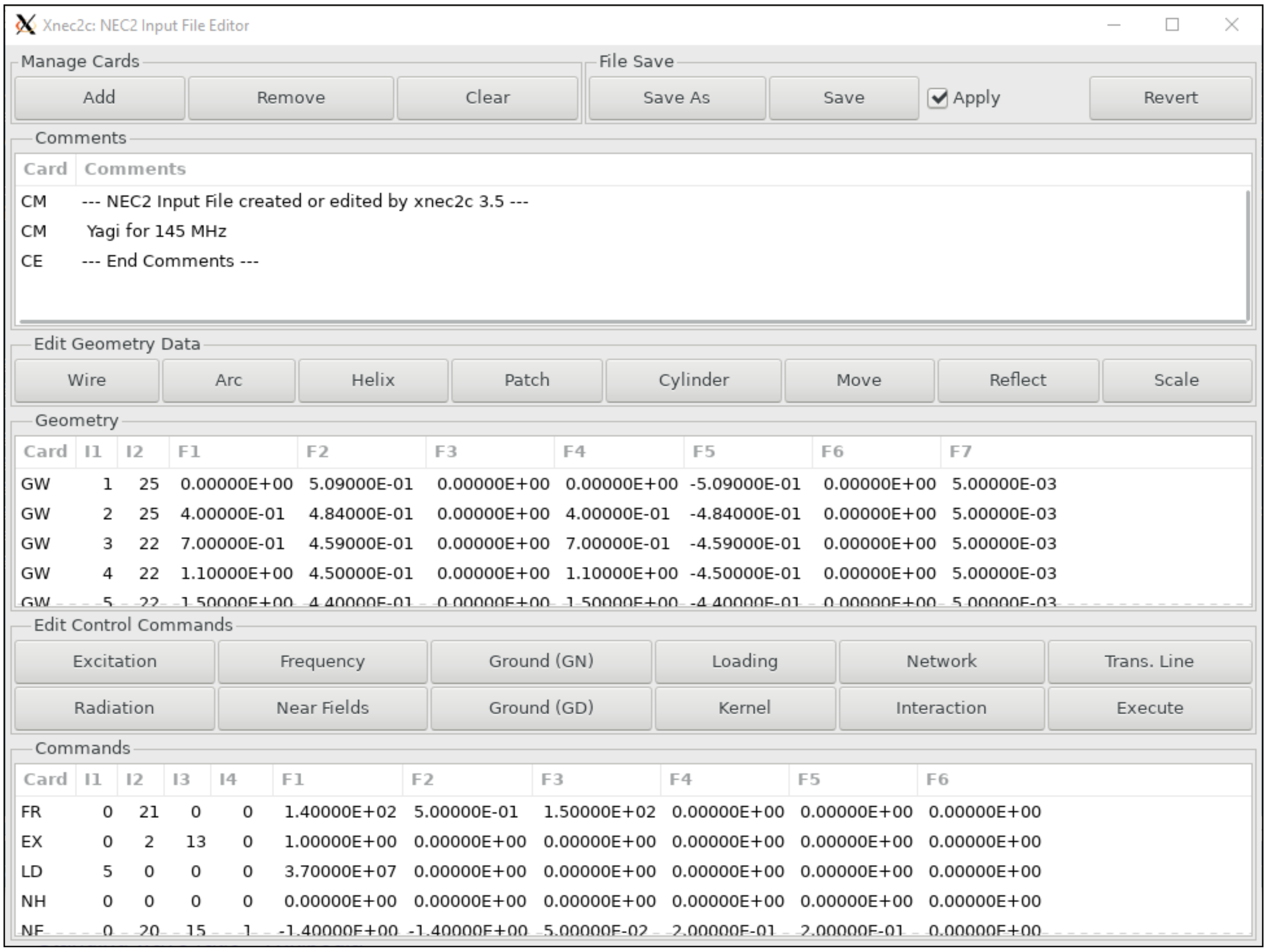
The second major display that's available shows how your antenna design behaves over a series of driving frequencies. Clicking View→Frequency Plots pops up a new window.

The first button shows you how the gain changes over a given frequency sweep.

Clicking on each subsequent button adds a new graph to the same window. For example, click the VSWR button to see how the Voltage Standing Wave Ratio (VSWR) changes over frequency.

You probably will need to resize the window if you want to view more than a couple graphs simultaneously. As with the E/H Fields plot from earlier, whether these plots behave as you expect depends on the input file you're using.

The example files provide a great place to start, but they're not likely to match your situation exactly. In those cases, you'll want to edit the input files to configure the



Card	Comments
CM	--- NEC2 Input File created or edited by xnec2c 3.5 ---
CM	Yagi for 145 MHz
CE	--- End Comments ---

Card	I1	I2	F1	F2	F3	F4	F5	F6	F7
GW	1	25	0.00000E+00	5.09000E-01	0.00000E+00	0.00000E+00	-5.09000E-01	0.00000E+00	5.00000E-03
GW	2	25	4.00000E-01	4.84000E-01	0.00000E+00	4.00000E-01	-4.84000E-01	0.00000E+00	5.00000E-03
GW	3	22	7.00000E-01	4.59000E-01	0.00000E+00	7.00000E-01	-4.59000E-01	0.00000E+00	5.00000E-03
GW	4	22	1.10000E+00	4.50000E-01	0.00000E+00	1.10000E+00	-4.50000E-01	0.00000E+00	5.00000E-03
GW	5	22	1.50000E+00	4.40000E-01	0.00000E+00	1.50000E+00	-4.40000E-01	0.00000E+00	5.00000E-03

Card	I1	I2	I3	I4	F1	F2	F3	F4	F5	F6
FR	0	21	0	0	1.40000E+02	5.00000E-01	1.50000E+02	0.00000E+00	0.00000E+00	0.00000E+00
EX	0	2	13	0	1.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
LD	5	0	0	0	3.70000E+07	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
NH	0	0	0	0	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00	0.00000E+00
NE	0	20	15	1	-1.40000E+00	-1.40000E+00	5.00000E-02	2.00000E-01	2.00000E-01	0.00000E+00

Figure 7. You can use the built-in editor to customize the input file to your personal specifications.

system exactly as you need it to be.

Although you simply can open the input file in any text editor, xnec2c includes an edit function built in. Clicking the File→Edit menu item pops open a new window where you can see all of the input cards.

You can change the values for already existent cards, but the best feature of this editor is that you can add new cards with the click of a button. If you are new to NEC2 and its input format, you may not necessarily remember the codes and formats for the various types of input cards. The editor window helps with this. There even are buttons, like mirror and scale, that aid in the design phase by taking advantage of symmetries you may have in your antenna.

In each of these displays, you have a couple different options when it comes to saving your results. The simplest is clicking File→Save As. This lets you save the resulting image as a PNG file. If you select File→Save As gnuplot, the actual data is exported into a file, and the format is appropriate for the type of data that you might want to feed into gnuplot. This way, you can tailor the resulting graphical display for your particular situation.

Results that xnec2c generates are not directly comparable with those from other implementations of NEC2. Its focus is more on the functionality of being interactive. If you want more traditional output, you can use xnec2c to view your antenna configuration, and then use nec2c to generate your final results.

—Joey Bernard

Patreon and *Linux Journal*

PATREON

Together with the help of *Linux Journal* supporters and subscribers, we can offer trusted reporting for the world of open-source today, tomorrow and in the future. To our subscribers, old and new, we sincerely thank you for your continued support. In addition to magazine subscriptions, we are now receiving support from readers via Patreon on our website. *LJ* community members who pledge $20 per month or more will be featured each month in the magazine. A very special thank you this month goes to:

- Appahost.com
- Chris Short
- Christel Dahlskjaer
- David Breakey
- Dr. Stuart Makowski
- Fred
- Henrik Halbritter (Albritter)
- James Mayes
- Josh Simmons

BECOME A PATRON

Reality 2.0: a *Linux Journal* Podcast

Join us each week as Doc Searls and Katherine Druckman navigate the realities of the new digital world: https://www.linuxjournal.com/podcast.

Lessons in Vendor Lock-in: 3D Printers

The open nature of the consumer 3D printing industry has made for a much more consumer-friendly world.

This article continues a series that aims to illustrate some of the various problems associated with vendor lock-in. In past articles, I've given examples showing how proprietary systems from disposable razors to messaging apps have replaced more open systems leading to vendor lock-in. This article highlights an ecosystem that, so far, has largely avoided vendor lock-in and describes the benefits that openness has provided members of the community, myself included: 3D printing.

I've been involved in 3D printing for several years. I've owned a number of printers, and I've seen incredible growth in the area from an incredibly geeky fringe to the much more accessible hobby that it is today. I've also written quite a few articles in *Linux Journal* about 3D printing, including a multi-part series on the current state of 3D printing hardware and software (see the Resources section for links to Kyle's previous *Linux Journal* articles on 3D printing). I even gave a keynote at SCALE 11x on the free software and open hardware history of 3D printing and how it mirrors the history of the growth of Linux distributions.

The Birth of 3D Printing in the Hobbyist Market

One interesting thing about the hobbyist 3D printing market is that it was founded on free software and open hardware ideals starting with the RepRap project. The idea behind that project was to design a 3D printer from off-the-shelf parts that could print as many of its own parts as possible (especially more complex, custom parts like gears). Because of this, the first generation of 3D printers were all homemade using Arduinos, stepper motors, 3D-printed gears and hardware you could find in the local hardware store.

As the movement grew, a few individuals started small businesses selling 3D printer kits that collected all the hardware plus the 3D printed parts and electronics for you to assemble at home. Later, these kits turned into fully assembled and supported printers, and after the successful Printrbot kickstarter campaign, the race was on to create cheaper and more user-friendly printers with each iteration. Sites like Thingiverse and YouMagine allowed people to create and share their designs, so even if you didn't have any design skills yourself, you could download and print everyone else's. These sites even provided the hardware diagrams for some of the more popular 3D printers. The Free Software ethos was everywhere you looked.

3D Printing Enters the Consumer Market

As 3D printers entered the mainstream, and consumer brands started to enter the market, I figured the honeymoon was over. Under the guise of ease of use, many of these first mainstream efforts bundled their printers with proprietary, branded software. Some even went as far as copying the old inkjet printer lock-in models and sold their 3D printer hardware at a discount or a loss and made up the cost with proprietary filament. Even MakerBot—one of the original 3D printer brands—abandoned releasing its designs to the public.

Of course, not all brands went the proprietary route. Some of the original brands—like Ultimaker, Printrbot and Lulzbot—all maintained their commitment to free software and open hardware. The pure RepRap hobbyist community also continued to improve and publish new designs. I figured if everything else went the proprietary route, I would just stick with my Printrbot.

The fact that most hobbyists stuck with open designs and free software and firmware led to some amazing innovations in 3D printing, such as the OctoPrint front end. Instead of always connecting your computer to your printer to start a job, OctoPrint allowed you to monitor your printer remotely and set up new print jobs all over the network via a web interface. OctoPrint even offered an image for Raspberry Pis, and that combination became one of the most popular ways to talk to your printer.

Thingiverse and YouMagine were full of case designs to let you mount a Raspberry Pi and camera to your printer.

OctoPrint flourishes because most of the printers were based off a standard and open hardware and firmware platform. They all spoke G-code—a series of common commands that allowed you to move stepper motors, set temperatures and otherwise monitor and control your printer. Using slicing software, like Slic3r or Cura, you would load an STL file containing your 3D design into the program, decide which printer quality settings you wanted to use, and then your slicer would convert the STL file into G-code. Normally, you then would take that G-code file and either copy it to an SD card in your printer or use a different program to load the G-code into your printer and monitor the print (Cura was able to act both as a slicer and as a front end). Because Cura was free software though, OctoPrint was able to incorporate its slicing engine into its own software, so you could upload STL files directly to the OctoPrint web interface, and it would slice them for you.

Bring in the Clones

One of the side effects of high-quality free hardware, firmware and software for 3D printers being readily available is that it enabled a large number of low-cost clones of popular 3D printer designs to flood the market. Where before a lower-cost printer was between $600–$1000, and higher-end printers were closer to $2000–$3000, now you could find clones of those models for half the price. What's more, although some of those $200 and $300 printers cut corners (and some caught fire!), for the most part, they actually produced good quality prints.

An interesting result of those clones was that since they used free software and firmware, they all worked with the same free software tools everyone already was using. In fact, some of them even released their own Cura profiles or complete forks of Cura with their own printer's settings enabled by default. The quality of the free software tools made it so that supporting them became critical in many consumers' decision-making process. Many of the proprietary consumer models couldn't compete, especially those that were incompatible with free software tools.

Oh Snap

Of course, those clones had another side effect. They made it much more difficult for some of the established brands to compete. My personal favorite brand, Printrbot, announced it was filing for bankruptcy this past fall. The next day, while working on my printer, I managed to snap the micro-USB port off the motherboard. Normally, I'd just order a replacement board, but of course, now that wasn't an option. Yet, thanks to the open standards in 3D printing, there were a number of generic third-party RepRap boards I could replace it with. Sure, it would mean spending quite a bit of time with the board to connect electronics, calibrate things and flash firmware, but the point is that it was an option at all (and in general, it's an option for any average 3D printer on the market).

Unfortunately, I didn't really have the free time to invest in such a project, so instead, my 3D printer sat in its broken state. Fortunately, I was surprised on Christmas with a new 3D printer—one of those low-cost clones. Because of all of the open standards in this industry, I was able to connect it directly to my existing OctoPrint server, tell the server about my printer's dimensions and settings, and get right back to printing. So far, I've been pleased and surprised with the quality of the prints, and I understand that without the open hardware and free software firmware and tools, it would have been a much different story.

The moral here is that although there sometimes are risks in a world without vendor lock-in, that world is much better for the end user. Open standards and interoperability provide so many more options than you would have in a proprietary world, including giving you options even if your vendor closes shop.

—Kyle Rankin

Resources

- “Lessons in Vendor Lock-in: Shaving” by Kyle Rankin
- “Lessons in Vendor Lock-in: Messaging” by Kyle Rankin
- “Getting Started with 3D Printing, the Software” by Kyle Rankin
- “Getting Started with 3D Printing, the Hardware” by Kyle Rankin
- “What’s New in 3D Printing, Part I: Introduction” by Kyle Rankin
- “What’s New in 3D Printing, Part II: the Hardware” by Kyle Rankin
- “What’s New in 3D Printing, Part III: the Software” by Kyle Rankin
- “What’s New in 3D Printing, Part IV: OctoPrint” by Kyle Rankin
- RepRap.org
- Thingiverse
- YouMagine
- MakerBot
- Ultimaker
- Lulzbot
- Printrbot (Wikipedia)
- OctoPrint
- G-code
- Slic3r
- Cura

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Groboards has launched a new "tiny, Adafruit Feather form-factor 'Giant Board' SBC that runs Linux on Microchip's SiP implementation of its Cortex-A5-based SAMA5D SoC and offers 128MB RAM, micro-USB, microSD and I/O including ADC and PWM", Linux Gizmos reports. There's no pricing or availability information at the time of this writing, but see the OSH Park blog and the Groboards site for specs and more info.

- A new open-source hardware project called Alias will keep Amazon and Google smart assistants from spying on you. According to the project's GitHub page, "Alias is a teachable 'parasite' that is designed to give users more control over their smart assistants, both when it comes to customisation and privacy. Through a simple app the user can train Alias to react on a custom wake-word/sound, and once trained, Alias can take control over your home assistant by activating it for you."

- A new major release of the open-source Metasploit Framework is now available. According to the Rapid7 blog post, version 5.0 of the penetration-testing tool is the first milestone update since version 4.0 came out in 2011. Along with a new release cadence, "Metasploit's new database and automation APIs, evasion modules and libraries, expanded language support, improved performance, and ease-of-use lay the groundwork for better teamwork capabilities, tool integration, and exploitation at scale."

- Orange Pi 3 SBC is now available. Linux Gizmos reports that the open-source hardware platform, Allwinner H6-based Orange Pi 3 SBC is now available for $30, or for $40 with 2GB of RAM and 8GB eMMC. Also, other highlights include "GbE, HDMI 2.0, 4x USB 3.0, WiFi-ac, and mini-PCIe." For more info, visit the Orange Pi 3 AliExpress page.

- Inkscape is finally reaching the 1.0 milestone after 15 years of development. Softpedia News reports that Inkscape 1.0 will feature "an updated user interface that offers better support for 4K/HiDPI screens and theming support, the ability to rotate and mirror canvases, new options for exporting to the PNG image format, variable fonts (requires pango 1.41.1 or higher), as well as much faster path operations and deselection of a large amounts of paths." You can download the pre-release alpha as an Appimage from here and see the release notes here.

- Creative Commons recently announced that 30,000 high-quality digital images from the Cleveland Museum of Art are now available. The free and open digital images are now under the CC0 and available via their API. The "CC0 allows anyone to use, re-use, and remix a work without restriction." Museum Director William M. Griswold said "Open Access with Creative Commons will provide countless new opportunities to engage with works of art in our collection. With this move, we have transformed not only access to the CMA's collection, but also its usability—inside as well as outside the walls of our museum."

- Dell launched its new XPS 13 9380 Developer Edition laptop, which runs Ubuntu out of the box. According to *Forbes*, highlights include Intel 8th generation i3, i5 and i7 processors; Ubuntu 18.04 LTS preloaded; InfinityEdge display with *top* camera placement; and much more. See Dell.com for more information.

- Raspberry Pi announces its Computer Module 3+ (CM3+) is now available for $25. The CM3+ is the "newest version of our flexible board for industrial applications and offers over ten times the ARM performance, twice the RAM capacity, and up to eight times the Flash capacity of the original Compute Module." The company also has released a refreshed Compute Module Development kit. The CM3+ will be available until at least January 2026.

- Firefox 65.0 was released to Channel users. New features include enhanced tracking protection, better experience for multilingual users, support for HandOff on macOS, better video streaming for Windows users, and improved performance and web compatibility, with support for the WebP image format.

Go here to download Firefox.

- Lucern Custom Instruments from the UK teamed up with Tracktion Corporation of Seattle to create Spirit Animal, an electric guitar with a Raspberry Pi synthesizer built in. According to the Raspberry Pi Blog, the guitar "boasts an onboard Li-ion battery granting about 8 hours of play time, and a standard 1/4" audio jack for connecting to an amp. To permit screen-sharing, updates, and control via SSH, the guitar allows access to the Pi's Ethernet port and wireless functionality." See also the Gear News website and the Lucern Instruments Facebook page for more information.

- Kodi 18.0 "Leia" is now available for all supported platforms. This is a major release, reflecting nearly 10,000 commits, 9,000 changed files and half a million lines of code added. This new release features support for gaming emulators, ROMs and controls; DRM decryption support; significant improvements to the music library; live TV improvements; and much more. See the changelog for more details, and go here to download.

- ZaReason debuted its new Gamerbox 9400, "the ultimate Linux gaming PC". And, the Gamerbox is just the beginning, ZDNet reports, quoting ZaReason CEO Cathy Malmrose: "Our current team is mostly gamers so, not surprisingly, that is the direction we are going. We have a full line of gaming machines in R&D." The Gamerbox runs Ubuntu 18.04, with a 64-bit Pentium 3.8Ghz G5500 Coffee Lake processor and 8GB of DDR4 memory.

- Google announced two new audio apps for Android to help people who are deaf or hard of hearing: Live Transcribe and Sound Amplifier. Live Transcribe "takes real-world speech and turns it into real-time captions using just the phone's microphone". Live Transcribe will roll out gradually as a limited beta via the Play Store and pre-installed on Pixel 3 devices. You can sign up here to be notified when it's more widely available. Sound Amplifier makes "audio more clear and easier to hear. You can use Sound Amplifier on your Android smartphone with wired headphones to filter, augment and amplify the sounds in your environment.

It works by increasing quiet sounds, while not over-boosting loud sounds." Sound Amplifier is available now via the Play Store and supports Android 9 Pie or later and comes pre-installed on Pixel 3.

- The Document Foundation announced the official release of LibreOffice 6.2 with NotebookBar. This is a major new release that "features a radical new approach to the user interface—based on the MUFFIN concept—and provides user experience options capable of satisfying all users' preferences, while leveraging all screen sizes in the best way." This version has many new features, including substantial changes to icon themes, context menus are tidied up and interoperability with proprietary file formats has been improved. See this video for details on all the new features. Note that LibreOffice 6.1.5 also was released recently for enterprise-class deployments. You can download LibreOffice 6.2 or LibreOffice 6.1.5 from here.

- Raspberry Pi has opened a store in the Grand Arcade, Cambridge, UK. See this video for details and follow #RPiStore for more photos and info.

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

The Free Software Foundation
presents

LibrePlanet 2019
"Trailblazing Free Software"

March 23-24, 2019
Massachusetts Institute of Technology (MIT)
Stata Center
Cambridge, MA

LibrePlanet brings together free software enthusiasts and professionals to learn skills, celebrate free software accomplishments, and face challenges to software freedom.

Featuring keynote speeches by:

* **Bdale Garbee** - Free software activist, **Debian Project** contributor
* **Micky Metts** - Free software activist, co-owner of **Agaric** tech cooperative
* **Tarek Loubani** - Physician & founder, **Glia Project**
* **Richard Stallman** - President & founder, **Free Software Foundation**

Free Software Foundation associate members and students attend gratis!

Libreplanet.org/2019
Register at u.fsf.org/lp19regp

Downsides to Raspberry Pi Alternatives

Learn about some of the risks when choosing an alternative to a Raspberry Pi for your project.

By Kyle Rankin

Kyle Rankin is a Tech Editor and columnist at *Linux Journal* and the Chief Security Officer at Purism. He is the author of *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting*, *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks* and *Ubuntu Hacks*, and also a contributor to a number of other O'Reilly books. Rankin speaks frequently on security and open-source software including at BsidesLV, O'Reilly Security Conference, OSCON, SCALE, CactusCon, Linux World Expo and Penguicon. You can follow him at @kylerankin.

I have a lot of low-cost single-board computers (SBCs) at my house. And, I've written a number of articles for *Linux Journal* that discuss how I put those computers to use—whether it's controlling my beer fridge, replacing a rackmount file server, acting as a media PC connected to my TV or as an off-site backup server in my RV (plus many more). Even more recently, I wrote a "Pi-ventory" article where I tried to count up just how many of these machines I had in my home.

Although the majority of the SBCs I use are some form of Raspberry Pi, I also sometimes use Pi alternatives—SBCs that mimic the Raspberry Pi while also offering expanded features—whether that's gigabit Ethernet, faster CPUs, SATA ports, USB3 support or any number of other improvements. These boards often even mimic the Raspberry Pi by having "Pi" in their names, so you have Orange Pi and Banana Pi among others. Although Pi alternatives allow you to solve some problems better than a Raspberry Pi, and in many cases they provide hardware with better specifications for the same price, they aren't without their drawbacks. So in this article, I take a look at the downsides of going with a Pi alternative based on my personal experience.

Third-Party Support

The initial Raspberry Pi was a runaway success, and all of the subsequent models have sold incredibly well. There are only a few variants on the Raspberry Pi platform, and later hardware upgrades have done a good job at maintaining backward-compatibility where possible (in particular with overall board dimensions and placement of ports). There also have been only a few "official" Raspberry Pi peripherals through the years (the camera being the best example). When you have this many of a particular hardware device out in the world, and the primary vendor is mostly focused on the hardware itself, you have a strong market for add-ons and peripherals from third parties.

The secondary Raspberry Pi market is full of cases, peripherals and add-on hardware like USB WiFi dongles that promise to be compatible out of the box with earlier models that didn't include WiFi. Adafruit is a good example of an electronics vendor who has jumped into the Raspberry Pi secondary market with a lot of different hobbyist kits that feature the Raspberry Pi as the core computing and electronics platform. That company and others also have created custom add-on shields intended to stack on top of the Raspberry Pi and add additional features including a number of different screen options, sensors and even cellular support. There's even a company that offers a case to turn a Raspberry Pi into a small laptop.

By contrast, the secondary market for Pi alternatives is pretty small. You might find cases or other basic peripherals, but because there just aren't as many of those devices out in the world, it's much riskier for a company to go to the trouble of making specific add-ons just for those devices. Fortunately, you often can re-use peripherals intended for a Raspberry Pi in these Pi alternatives. Many competitors make a point to ensure they are pin-for-pin compatible with the Raspberry Pi's GPIO pins, which are used for many add-ons. Even with this though, compatibility is not a sure thing—you have to do your research in many cases or risk wasting money on an add-on that might fit but might not have software support once you boot.

The Community

One of the major things Raspberry Pi has going for it is its community. Because of its popularity and how many of these devices were sold compared to those of competitors, it has an enormous community of users that spread across all kinds of disciplines—from electrical engineers to gamers to sysadmins to artists.

There are a lot of advantages to having a large and diverse community behind a project. It means that when you want to do something with your SBC, you are less likely to be the first. There's a good chance you will find more complete (and more up-to-date) documentation and HOWTOs for whatever you want to do. It also means if you run into problems following one of these guides, there is a community on forums and chat rooms who can help walk you through it.

Many of the Pi alternatives do have communities in their own right with forums and chat rooms that you can turn to for support. There are also guides online for some of the more common use cases for a particular board (for instance, boards that feature SATA and gigabit Ethernet being used as a file server). Some of the common software projects (like media PC projects) often also include Pi alternatives in their documentation alongside Raspberry Pi if a particular board becomes popular for that use. So in many cases, the community is there, but if you start to stray from the common uses for your particular board, you are more likely to be on your own, or at least you'll have to adapt a Raspberry Pi guide to your SBC.

OS Support

There are a couple official operating systems for the Raspberry Pi direct from the vendor. Due to the popularity of the Raspberry Pi and the number of people using them, these operating systems support the hardware well and receive regular updates. Raspbian, one of the most popular OS options for the Raspberry Pi, essentially behaves like a standard Debian distribution with Raspberry Pi-specific scripts and kernel patches so you can get the most out of the hardware (and perform overclocking and other tweaks).

The community also has created a wide range of special-purpose OSes for the Raspberry Pi based on Raspbian that add specific features. Some examples include OSMC, which adds the Kodi media PC software, and OctoPrint, which adds a network-based 3D printer front end. There are also projects that build a custom Raspberry Pi OS from the ground up, such as OpenELEC and LibreELEC, which include Kodi just like OSMC does but with a stripped-down, special-purpose OS that boots much faster.

OS support, in particular kernel support, is probably the biggest downside to choosing a Pi alternative. Each vendor tends to provide a few different OS options with their boards, some that mimic Raspbian and others that are based on Android. These distributions also provide custom kernels with patches to support the hardware, but unfortunately compared to the Raspberry Pi world, it's a toss-up as to how well these kernels support the hardware. I've had some Pi alternatives work perfectly well and stably with the vendor-provided Linux OS, while others seemed incredibly unstable. These custom kernels also are problematic because in some cases, it can be challenging to track down the source code for their custom patches. Even if you do find the patches, you might be on your own if you want to apply them to a newer kernel.

The other problem with Pi alternatives is fragmentation and long-term support for the provided OS. Vendors tend to create their own custom distributions for their boards instead of using Raspbian or other shared distributions. These vendors also tend to create updated models every few years, but the OS for a particular revision tends to freeze in time. This means outdated Linux distributions and Android versions as time goes on. If you are someone who wants to use these SBCs as a server platform in particular, having regular security updates and overall updates for your services is very important. In these cases, the kernel packages also tend to freeze in place.

Fortunately, the Armbian project has stepped up to provide a general-purpose Raspbian-like OS for Pi alternatives. Depending on your board, Armbian will have Debian- and Ubuntu-based images you can use, each with kernels patched to support your hardware. Unfortunately, for some boards, the kernel is a bit less stable, and Armbian tries to gauge stability for all of the hardware it supports. Armbian also gets regular updates, so a board that was unstable one day might improve over time. More important, you will get more regular updates to the rest of the software on your system with less worry that the device will be abandoned. As we all accumulate more and more of these computers around our homes, regular security updates are incredibly important.

Conclusion

If you are debating between a Raspberry Pi and another board for your project, I hope this article has been helpful. In general, my advice is just to do your research before you buy any particular board—even a Raspberry Pi. Without proper research, you may end up with a board that won't work well for your project or that works only with an ancient OS. Defaulting to a Raspberry Pi is also not a silver bullet. I've chosen Pi alternatives for many of my projects specifically because a Raspberry Pi would have been under-powered (or didn't have proper hardware support) for my needs. ■

Send comments or feedback
via http://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Resources

- “Temper Temper” by Kyle Rankin
- “Temper Pi” by Kyle Rankin
- “Working on My Temper” by Kyle Rankin
- “N900 with a Slice of Raspberry Pi” by Kyle Rankin
- “Raspberry Strudel: My Raspberry Pi in Austria” by Kyle Rankin
- “Flash ROMs with a Raspberry Pi” by Kyle Rankin
- “LJ Tech Editor’s Personal Stash of Raspberry Pis and Other Single Board Computers” by Kyle Rankin
- “Papa’s Got a Brand New NAS” by Kyle Rankin
- “Banana Backups” by Kyle Rankin
- “Super Pi Brothers” by Kyle Rankin
- “DIY RV Offsite Backup and Media Server” by Kyle Rankin
- “Two Portable DIY Retro Gaming Consoles” by Kyle Rankin
- “Raspberry Pi Alternatives” by Kyle Rankin
- Raspbian
- OSMC
- Kodi
- OctoPrint
- OpenELEC
- LibreELEC
- Armbian
- Adafruit

Become Queen Bee for a Day Using Python's Built-in Data Types

Cheaters never win, but at least they can use Python.

By Reuven M. Lerner

Reuven Lerner teaches Python, data science and Git to companies around the world. You can subscribe to his free, weekly "better developers" e-mail list, and learn from his books and courses at http://lerner.co.il. Reuven lives with his wife and children in Modi'in, Israel.

Like many other nerds, I love word puzzles. I'm not always great at them, and I don't always have time to do them, but when I do, I really enjoy them.

I recently discovered a new daily puzzle, known as "spelling bee", that the *New York Times* offers online. The idea is simple. There are seven different letters, one in the center of a circle and six around it. Your job is to make as many different words as you can from those seven letters. Each word must be at least four letters long, and each word also must contain the center letter. You can use each letter as many times as you want.

So if the letters are "eoncylt", with a center letter of "y", some of the words you could create might be "cyclone", "eyelet" and "nylon".

The online game gives you a score based on how many words you've made from the potential pool. If you get them all, you're awarded "queen bee" status.

I do pretty well at this puzzle, but I've never managed to find all of the hidden words. Nevertheless, I have become queen bee on a few occasions. How? The answer is simple. I cheated. How? Using Python, of course.

Now, cheating at games isn't necessarily the first order of business when it comes to programming. And cheating at word games in which you're competing against yourself is probably a sign of unhealthy competition. But, doing so also provides a great way to review some of the ways you can use Python's built-in data types and the ease with which you can process words and text.

So in this article, I explore a number of ways you can cheat—and yes, become the queen bee, if only for a day.

Trying All Combinations

To start, you simply might try to form all of the possible combinations you can with the letters you're given. As you might remember from high-school math class, there's a difference between "permutations" and "combinations". When you generate "permutations", the order is important, but when you generate "combinations", the order is not important.

You easily can see this using Python's `itertools` module, a part of the standard library that has functions named `permutations` and `combinations`. Each takes both an iterable data structure and the number of items you want in each resulting list. For example:

```
>>> list(itertools.combinations(['a', 'b', 'c', 'd'], 2))
[('a', 'b'), ('a', 'c'), ('a', 'd'), ('b', 'c'), ('b', 'd'),
 ↪('c', 'd')]

>>> list(itertools.permutations(['a', 'b', 'c', 'd'], 2))
```

```
[('a', 'b'),
 ('a', 'c'),
 ('a', 'd'),
 ('b', 'a'),
 ('b', 'c'),
 ('b', 'd'),
 ('c', 'a'),
 ('c', 'b'),
 ('c', 'd'),
 ('d', 'a'),
 ('d', 'b'),
 ('d', 'c')]
```

As you can see, the output from `combinations` saw the order as unimportant and thus returned just `('a', 'b')`. But `permutations` saw the order as important, and thus returned both `('a', 'b')` and `('b', 'a')`.

You could use this to generate all of the letter combinations for possible words and then sift through that, right? Well, not really, for two reasons. First, the game allows you to repeat letters. And second, these functions let you specify a only single number of outputs.

You can solve the first problem by using the `combinations_with_replacement` function, which not only has a long name, but (as the name states) also allows letters to appear more than once in the output. For example:

```
>>>  list(itertools.combinations_with_replacement(['a', 'b',
 ↪'c', 'd'], 2))
[('a', 'a'),
 ('a', 'b'),
 ('a', 'c'),
 ('a', 'd'),
 ('b', 'b'),
 ('b', 'c'),
 ('b', 'd'),
 ('c', 'c'),
 ('c', 'd'),
 ('d', 'd')]
```

However, you want to find words that are at least four letters long. Let's assume that the longest possible word would be 12 letters long. You could use a `for` loop, appending the results from each iteration to a list. But an even more Pythonic way to do this would be to use a list comprehension—or even better, a nested list comprehension. Comprehensions are, in my experience, one of the hardest concepts for new Python developers to use. However, they are perfect for creating and transforming sequences, which is precisely what's happening here:

```
>>> one_combination
    for n in range(4, 13)
    for one_combination in
       itertools.combinations_with_replacement('abc', n)]
```

This code iterates over the range from 4 to 13, thus producing the integers from 4 through 12. For each of these values of n, you then produce all of the combinations, with replacement, of that length and with the letters "a", "b" and "c". The results then are output as a list.

This is great, but it's missing at least two things. First, you aren't interested in combinations, so much as words. And there aren't just a few letters for which you're searching, but seven letters—one of which must appear in the word.

So let's beef up the comprehension a bit in order to get all of the words:

```
>>> all_letters = 'eoncylt'
>>> center_letter = 'y'
>>> [''.join(one_combination)
```

```
    for n in range(4, 13)
    for one_combination in
       itertools.combinations_with_replacement(all_letters, n)
       if center_letter in one_combination]
```

The good news is that this now indeed has generated all of the possible combinations that might give you the answer. The bad news is that it's also generated a lot of combinations that aren't really words. Indeed, according to my count, this created 31,788 "words", including such greats as "occccccylt" and "eeeyt".

So yes, you could become the queen bee by going through and entering all of these words, one by one, as input into the game. Somehow though, I think entering 31,788 words takes the fun out of cheating.

Enter the Dictionary

To make cheating more efficient and fun, let's try a different strategy. Instead of generating all of the possible combinations of letters, let's instead search through only those combinations that are correct. How can you know what's correct? Via the dictionary, of course—and the fact that Linux comes with an English-language dictionary makes this easier.

Indeed, although it often can be useful to generate combinations, it's probably a wiser strategy just to start with the dictionary and choose the words that fit your needs.

The dictionary that I'm using for this example is in /usr/share/dict/american-english, and it contains 102,401 different words, each on a line by itself. The fact that each word is on a separate line turns out to be a great advantage, because it means that you can (once again) create a list comprehension. In this case, the source of the list comprehension won't be an iterator from `itertools`, but rather the dictionary file itself. Iterating over a file in Python gives you one line per iteration. Here's how you can get these words:

```
[one_word.strip()
```

```
    for one_word in open(words_file)]
```

But, wait a moment. This will return all of the words. You're interested only in those words that contain the seven letters in the puzzle, as well as the center letter.

One solution to this would be to write a function that checks whether the word fits your needs. For example:

```
>>>  def is_legal(word, all_letters, center_letter):
        word = word.strip()

        if len(word) < 4:
            return False

        if center_letter not in word:
            return False

        for one_letter in word:
            if one_letter not in all_letters:
                return False

        return True
```

The function would appear to work too:

```
>>> is_legal('cy', all_letters, center_letter)
    False

>>> is_legal('cycle', all_letters, center_letter)
    True

>>> is_legal('hairbrush', all_letters, center_letter)
    False
```

Armed with this function, you now can read through the dictionary and get the words that fit:

```
[one_word.strip()
    for one_word in open(words_file)
   if is_legal(one_word, all_letters, center_letter)]
```

Note the use of the `strip` method to remove leading and trailing whitespace from the word, mostly because of the newline that you'll get from each word in the file. For this reason, you'll also use `strip` in the `is_legal` function to ensure that you don't have to deal with the newline.

So, does it work? The answer is yes, mostly. On some days, my Python program finds words that weren't in the game. And on other days, the game is looking for words that aren't in the Linux dictionary. But for the most part, everything seems to work well, and although I try hard to win the game each day, there are definitely some days when I give up, run my program and feel smug at my programming skills, if not my word-game skills.

Conclusion

Whoever said that "cheaters never win" didn't take into consideration that cheating might lead to a better understanding of programming and data structures. And indeed, I often tell people that the key to good programming is often not knowing the best algorithms, but rather knowing which libraries to apply and how to combine the strengths of the language you're using, so that you can do as little work as possible. This article shows how to tackle a simple real-world problem that involved very little of your own code. Rather, understanding the problem, Python's standard library and its data structures combined to give the answers. ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Fun with Mail Merge and Cool Bash Arrays

Creating a sed-based file substitution tool.

By Dave Taylor

Dave Taylor has been hacking shell scripts on Unix and Linux systems for a really long time. He's the author of *Learning Unix for Mac OS X* and *Wicked Cool Shell Scripts*. You can find him on Twitter as @DaveTaylor, and you can reach him through his tech Q&A site Ask Dave Taylor.

A few weeks ago, I was digging through my spam folder and found an email message that started out like this:

```
Dear #name#
Congratulations on winning the $15.7 million
lottery payout!
To learn how to claim your winnings, please...
```

Obviously, it was a scam (does anyone actually fall for these?), but what captured my attention was the `#name#` sequence. Clearly that was a fail on the part of the sender who presumably didn't know how to use AnnoyingSpamTool 1.3 or whatever the heck he or she was using.

The more general notation for bulk email and file transformations is pretty interesting, however. There are plenty of legitimate reasons to use this sort of substitution, ranging from email newsletters (like the one I send every week from AskDaveTaylor.com—check it out!) to stockholder announcements and much more.

With that as the inspiration, let's build a tool that offers just this capability.

The simple version will be a 1:1 substitution, so `#name#` becomes, say, "Rick Deckard", while `#first#` might be "Rick" and `#last#` might be "Deckard". Let's build on that, but let's start small.

Simple Word Substitution in Linux

There are plenty of ways to tackle the word substitution from the command line, ranging from Perl to awk, but here I'm using the original UNIX command `sed` (stream editor) designed for exactly this purpose. General notation for a substitution is s/old/new/, and if you tack on a `g` at the end, it matches every occurrence on a line, not only the first, so the full command is s/old/new/g.

Before going further, here's a simple document that has necessary substitutions embedded:

```
$ cat convertme.txt
#date#

Dear #name#, I wanted to start by again thanking you for your
generous donation of #amount# in #month#. We couldn't do our
work without support from humans like you, #first#.

This year we're looking at some unexpected expenses,
particularly in Sector 5, which encompasses #state#, as you
know. I'm hoping you can start the year with an additional
contribution? Even #suggested# would be tremendously helpful.

Thanks for your ongoing support. With regards,

Rick Deckard
Society for the Prevention of Cruelty to Replicants
```

Scan through it, and you'll see there's a lot of substitutions to do: `#date#`,

`#name#`, `#amount#`, `#month#`, `#first#`, `#state#` and `#suggested#`. It turns out that `#date#` will be replaced with the current date, and `#suggested#` is one that'll be calculated as the letter is processed, but that's for a bit later, so stay tuned for that.

To make life easy, a source file that's a comma-separated list allows for easy interaction with a source spreadsheet, so a sample input data file might look like this:

```
name:first:amount:month:state
Eldon Tyrell:Eldon:500:July:California
```

At its most basic, the first line defines variable names (without the # notation), and subsequent lines are a set of values for a particular donor or recipient. To start, let's read in the variable names:

```
while IFS=',' read -r f1 f2 f3 f4 f5 f6 f7
do
  declare -a varname=($f1 $f2 $f3 $f4 $f5 $f6 $f7)
done
```

Key to understanding this is to know about IFS, the internal field separator. Normally, it's white space, which is why, for example, `ls my file name` looks for three files called my, file and name. But you can change it, as I demonstrate by changing IFS to a comma.

Those Cool Bash Arrays

I declare an array called `varname` that receives each of the fields read into the script. There are only five fields in use at this point, but let's support up to seven to make the resultant script a bit more flexible.

Arrays are really cool in Bash actually, but the notation is a smidge funky. That is, you can't just use `$array[index]`, because it won't be parsed correctly, so curly braces are a necessary addition:

```
echo ${varname[1]}
```

That works just fine.

For a basic algorithm, you're going to have two parallel arrays (parallel in that their indices will match up): one that retains all the variable names, and the other that contains the values for this instance of the data entry list.

This means you'll need to differentiate between the situation when the script is reading the first line and when subsequent lines of the data file are read. Easily done:

```
(( lines++ ))

if [ $lines -eq 1 ] ; then   # field names
  # variable names
  declare -a varname=($f1 $f2 $f3 $f4 $f5 $f6 $f7)
else
  # values for this line (can contain spaces)
  declare -a value=("$f1" "$f2" "$f3" "$f4" "$f5"
     "$f6" "$f7")
fi
```

As with most code, this makes assumptions here, but they're safe: variable names aren't quoted because they're always a single word, but variable values might have spaces, so they do end up quoted in the declare statement. Otherwise, this should be easy, and the `(( lines++ ))` notation should make you cheer—it's a nice Bash shortcut!

Once you're past the very first line, the script can look in `varname[x]` for the xth variable name, and `value[x]` for the value of that named variable, expressed as a series of `sed`-friendly substitution commands:

```
for ((i=0; i<${#value[*]}; i++))
do
  if [ ! -z "${value[$i]}" ] ; then
    echo "s/#${varname[$i]}#/${value[$i]}/g"
  fi
done
```

Which produces this:

```
s/#name#/Eldon Tyrell/g
s/#first#/Eldon/g
s/#amount#/500/g
s/#month#/July/g
s/#state#/California/g
```

That's pretty darn close to what you want actually. Let's push forward.

Working with sed

The stream editor `sed` is far more powerful than its modest and ancient history might suggest. It's perfect for this job, as shown above.

You could write the above lines into a temp file and invoke `sed` directly, but let's avoid the file I/O and turn it all into a command-line argument as necessary. That's done by simply separating each command with a semicolon, which you can do by building it in a temp variable instead:

```
for ((i=0; i<${#value[*]}; i++))
do
  if [ ! -z "${value[$i]}" ] ; then
    if [ -z "$SUBS" ] ; then
      SUBS="s/#${varname[$i]}#/${value[$i]}/g"
    else
      SUBS="$SUBS;s/#${varname[$i]}#/${value[$i]}/g"
    fi
  fi
done
```

There's undoubtedly a way to avoid the innermost if-then-else statement to omit the unnecessary ; prefix, but sometimes it's easier to have a few lines of code than yet more gobbledygook.

Otherwise, the above is a simple expansion from the previous `for` loop shown. This time, it builds the entire `sed` command within the `SUBS` substitution variable. Here's how to test:

```
echo "   sed \"$SUBS\" $inputfile"
```

When you run this with the input data file, here's what's pushed out to the terminal:

```
sed "s/$name$/Eldon Tyrell/g;s/$first$/Eldon/g;
    s/$amount$/500/g;s/$month$/July/g;
    s/$state$/California/g" convertme.txt
sed "s/$name$/Rachel/g;s/$first$/Rachel/g;
    s/$amount$/100/g;s/$month$/March/g;
    s/$state$/New York/g" convertme.txt
```

(Note: line breaks added for formatting purposes only.)

It's actually a very small step from here to invoke the command, so let's do that:

```
$ sub.sh
#date#

Dear Eldon Tyrell, I wanted to start by again thanking you
for your generous donation of 500 in July. We couldn't do
our work without support from humans like you, Eldon.
```

```
This year we're looking at some unexpected expenses,
particularly in Sector 5, which encompasses California, as
you know. I'm hoping you can start the year with an
additional contribution? Even #suggested# would be
tremendously helpful.

Thanks for your ongoing support. With regards,

Rick Deckard
Society for the Prevention of Cruelty to Replicants
$
```

Generally, this looks good. `#date#` and `#suggested#` are still untranslated, but that's as expected. What is a bit odd is that it didn't get the second entry too. A bug.

I'm going to stop here, however, and maybe next time, I'll add some system substitutions like `#date#` and figure out how to calculate `#suggested#`, which can be 50% of the actual donation. See you soon! ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

diff -u

What's New in Kernel Development

By Zack Brown

Zack Brown is a tech journalist at *Linux Journal* and *Linux Magazine*, and is a former author of the "Kernel Traffic" weekly newsletter and the "Learn Plover" stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the *Crumble* pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.

Chasing Archives

Kernel development is truly impossible to keep track of. The main mailing list alone is vast beyond belief. Then there are all the side lists and IRC channels, not to mention all the corporate mailing lists dedicated to kernel development that never see the light of day. In some ways, kernel development has become fundamentally mysterious.

Once in a while, some lunatic decides to try to reach back into the past and study as much of the corpus of kernel discussion as he or she can find. One such person is **Joey Pabalinas**, who recently wanted to gather everything together in **Maildir** format, so he could do searches, calculate statistics, generate pseudo-hacker AI bots and whatnot.

He couldn't find any existing giant corpus, so he tried to create his own by piecing together mail archived on various sites. It turned out to be more than a million separate files, which was too much to host on either **GitHub** or **GitLab**. He asked the **linux kernel mailing list** for suggestions on better hosting opportunities. Although he acknowledged, "It's possible I'm the only weirdo who finds this kind of thing useful, but I figured I should share it just in case I'm not."

Joe Perches suggested plumbing the archives at kernel.org/lore.html, which go back decades. But Joey said he'd tried that, and he found it all but impossible to convert those archives to the Mailbox format he wanted. Instead, he'd spent the previous several weeks scraping the lkml.org archive and scripting his own conversion routines.

Konstantin Ryabitsev remarked:

> The maildir format is kind of terrible for LKML, because having millions of messages in a single directory is very hard on the underlying FS. If you break it up into multiple folders, then it becomes difficult to search. This is the main reason why we have chosen to go with the public-inbox format, which solves both of these problems and allows for a very efficient archive updating and replication using git.

Meanwhile, **Jasper Spaans** raised his eyebrows at Joey's statement that he'd gotten more than a million separate files by scraping lkml.org. Jasper said:

> First of all, there are more than 3M messages stored in the lkml.org database, so I guess you've missed some messages or something is really broken. Besides, unless you figured out how to get to the raw data, you've just scraped a rendering which discards stuff like pgp signatures etc and has very incomplete headers. Unless you don't care for those of course.

Jasper added that he'd also been working on extracting Maildir-type data out of the lore website, and he sent Joey the code he'd been using to do that.

Eric Wong also sent Joey a script he'd been using to convert slrn threaded **Usenet** repositories to Maildir; although like others, he recommended against putting millions (and millions) of files into a single directory.

The discussion wasn't headed anywhere; it was just various people sharing knowledge and making judgment calls.

Once upon a time, and a very long time ago it was, I wanted to get a hold of the earliest archives of Linux kernel development discussions. I asked everyone where I could find them, and one of the developers replied that he had a lot of that stuff mixed up in his mail archives, along with all manner of other email messages. I wrote back and eagerly told him I'd love to get my hands on it. He wrote back again, explaining that there was just no way he could take the time to extract the private stuff from the public stuff. And, that was the end of that. I've always wondered why he responded to my initial email in the first place, if he was just going to say no at the end. And, that's the tale of how I came *this close* to writing up summaries of the very earliest Linux developments.

Considering Fresh C Extensions

Matthew Wilcox recently realized there might be a value in depending on **C** extensions provided by the **Plan 9** variant of the C programming language. All it would require is using the `-fplan9-extensions` command-line argument when compiling the kernel. As Matthew pointed out, Plan 9 extensions have been supported in **GCC** as of version 4.6, which is the minimum version supported by the kernel. So theoretically, there would be no conflict.

Nick Desaulniers felt that any addition of `-f` compiler flags to any project always would need careful consideration. Depending on what the extensions are needed for, they could be either helpful or downright dangerous.

In the current case, Matthew wanted to use the Plan 9 extensions to shave precious bytes off of a cyclic memory allocation that needed to store a reference to the "next" value. Using the extensions, Matthew said, he could embed the "next" value without breaking various existing function calls.

Nick also suggested making any such extension dependencies optional, so that other compilers would continue to be able to compile the kernel.

It looked as though there would be some back and forth on the right way to proceed, but **Linus Torvalds** immediately jumped in to veto the entire concept, saying:

> Please don't.
>
> The subset of the plan9 extensions that are called just "ms" extensions is fine. That's a reasonable thing, and is a very natural expansion of the unnamed structures we already have—namely being able to pre-declare that unnamed structure/union.
>
> But the full plan9 extensions are nasty, and makes it much too easy to write "convenient" code that is really hard to read as an outsider because of how the types are silently converted.
>
> And I think what you want is explicitly that silent conversion.
>
> So no. Don't do it. Use a macro or an inline function that makes the conversion explicit so that it's shown when grepping.
>
> The "one extra argument" is not a strong argument for something that simply isn't that common. The upsides of a nonstandard feature like that needs to be pretty compelling.
>
> We've used various gcc extensions since day #1 ("inline" being perhaps the biggest one that took _forever_ to become standard C), but these things need to have very strong arguments.
>
> "One extra argument" that could be hidden by a macro or a helper inline is simply not a strong argument.

Nick was sympathetic to this point, and said:

> Implicit conversions are the most pointed to "defect" in languages like JavaScript. I understand why they're convenient, but the resulting surprising bugs don't outweigh their cost (again, my opinion), and developers frequently can't keep these implicit conversion rules straight (whether we're talking about JavaScript or C).

There was no real conversation after Linus' denunciation. But, the question of which compiler extensions to use and which not to use is very interesting, especially for a project like Linux that's used everywhere and runs everything. With all the different environments it needs to support, that support needs to include the possibility of all sorts of development platforms. To some extent, Linus' argument that Matthew was trying to create hard-to-read code is relevant, but it's also relevant that other compilers and other build environments also need to be able to support Linux development.

Handling Complex Memory Situations

Jérôme Glisse felt that the time had come for the Linux kernel to address seriously the issue of having many different types of memory installed on a single running system. There was main system memory and device-specific memory, and associated hierarchies regarding which memory to use at which time and under which circumstances. This complicated new situation, Jérôme said, was actually now the norm, and it should be treated as such.

The physical connections between the various CPUs and devices and RAM chips—that is, the bus topology—also was relevant, because it could influence the various speeds of each of those components.

Jérôme wanted to be clear that his proposal went beyond existing efforts to handle heterogeneous RAM. He wanted to take account of the wide range of hardware and its topological relationships to eek out the absolute highest performance from a given system. He said:

> One of the reasons for radical change is the advance of accelerator like GPU or FPGA means that CPU is no longer the only piece where computation happens. It is becoming more and more common for an application to use a mix and match of different accelerator to perform its computation. So we can no longer satisfy our self with a CPU centric and flat view of a system like NUMA and NUMA distance.

He posted some patches to accomplish several different things. First, he wanted to expose the bus topology and memory variety to userspace as a clear API, so that both the kernel and user applications could make the best possible use of the particular hardware configuration on a given system. A part of this, he said, would have to take account of the fact that not all memory on the system always would be equally available to all devices, CPUs or users.

To accomplish all this, his patches first identified four basic elements that could be used to construct an arbitrarily complex graph of CPU, memory and bus topology on a given system.

These included "targets", which were any sort of memory; "initiators", which were CPUs or any other device that might access memory; "links", which were any sort of bus-type connection between a target and an initiator; and "bridges", which could connect groups of initiators to remote targets.

Aspects like bandwidth and latency would be associated with their relevant links and bridges. And, the whole graph of the system would be exposed to userspace via files in the SysFS hierarchy.

In addition, Jérôme's patches provided a way to express memory policy. A system's memory policy is the mechanism it uses to decide which memory to use for a given task. For example, it might use faster memory first and slower memory only when fast memory is full. But, the kernel's current memory policy was organized on a per-CPU basis, which Jérôme felt was not good enough. But, he also acknowledged that trying to change that aspect of kernel infrastructure directly might break a lot of existing code. To deal with this, his patch added an entirely new memory policy API that new user code could take advantage of and old user code simply could ignore.

Aneesh Kumar responded to all of this, in particular praising Jérôme's approach of keeping the new API separate from the old. But Aneesh said, "that is also the drawback isn't it? We now have multiple entities tracking cpu and memory."

Aneesh also wanted to confirm that "once we have these different types of targets, ideally the system should be able to place them in the ideal location based on the affinity of the access. ie. we should automatically place the memory such that initiator can access the target optimally."

Jérôme seemed to agree with this in principle, but he also seemed to feel that making any of this automatic was still not guaranteed. The first step, he felt, was to expose the APIs and data structures, and then see what could be accomplished.

Meanwhile, **Dave Hansen** pointed out that there were existing elements of the kernel that dealt with heterogeneous memory. Dave said that **HMAT** (Heterogeneous Memory Attribute Table) existed in firmware specifically to express the topology to the kernel. Dave also said that **NUMA** (Non-Uniform Memory Access) was already part of the kernel, and wasn't lying fallow. Additionally, he pointed out that the **ACPI** (Advanced Configuration and Power Interface) specification had officially embraced NUMA, and there were Linux developers actively contributing patches to support this.

So, Dave was not immediately enthusiastic about ditching this ongoing momentum in one direction, in order to accept Jérôme's radical solution that went in an entirely new direction.

But, Jérôme replied that he was not trying to overthrow the existing work or any kernel patches that made use of HMAT. He said that all of that was still useful just on its own. But he added:

> I do not see how to evolve NUMA to support device memory. [...] I can not expose device memory as NUMA node as device memory is not cache coherent on AMD and Intel platform today. [...] In some case that memory is not visible at all by the CPU which is not something you can express in the current NUMA node.

Somewhat mollified, Dave replied:

> Yeah, our NUMA mechanisms are for managing memory that the kernel itself manages in the "normal" allocator and supports a full feature set on. That has a bunch of

> implications, like that the memory is cache coherent and accessible from everywhere.
>
> The HMAT patches only comprehend this "normal" memory, which is why we're extending the existing /sys/devices/system/node infrastructure.
>
> This series has a much more aggressive goal, which is comprehending the connections of every memory-target to every memory-initiator, no matter who is managing the memory, who can access it, or what it can be used for.
>
> Theoretically, HMS could be used for everything that we're doing with /sys/devices/system/node, as long as it's tied back into the existing NUMA infrastructure *somehow*.

Jérôme agreed with all of the above, and Dave seemed to be on board with Jérôme's approach. But, he did have some practical objections. For one thing, he said:

> We support 1024 NUMA nodes on x86. The ACPI HMAT expresses the connections between each node. Let's suppose that each node has some CPUs and some memory.
>
> That means we'll have 1024 target directories in sysfs, 1024 initiator directories in sysfs, and 1024*1024 link directories. Or, would the kernel be responsible for "compiling" the firmware-provided information down into a more manageable number of links?
>
> Some idiot made the mistake of having one sysfs directory per 128MB of memory way back when, and now we have hundreds of thousands of /sys/devices/system/memory/memoryX directories. That sucks to manage. Isn't this potentially repeating that mistake?

Dave also was worried that if Jérôme went forward with his patches, it could be four or five years before all the problems were solved, in which case, some portions of memory management would be bottle-necked waiting for those solutions, which could have been solved sooner using existing NUMA projects.

Additionally, Dave was curious how Jérôme's code would scale. He said, "It's quite easy to represent a laptop, but can this scale to the largest systems that we expect to encounter over the next 20 years that this ABI will live?"

At this point, Jérôme and Dave were joined by several other folks and began diving into the technical details, further objections and possible solutions that might come out of Jérôme's work. Ultimately, it seemed as if these patches did not represent a threat to existing approaches to memory, and that Jérôme would have support—or at least tolerance—from NUMA-related projects.

The things I love about this discussion are, first of all, that one developer got a big idea that seemed to go against current thinking, but that solved problems he saw as real. Second, that a developer on the other side of the issue was actually interested in the new approach and willing to take it seriously rather than tear it down.

Also, the whole direction of hardware resources is really becoming so strange. The kernel tries to eek out absolutely everything it can from the various devices on the system—even to the point of going beyond the ways in which those devices thought they would be used! And then once the kernel starts using them that way, other devices come out that use the kernel's new infrastructure. And so we end up with some kind of crazy situation requiring crazy solutions like what Jérôme has proposed.

Note: if you're mentioned in this article and want to send a response, please send a message with your response text to ljeditor@linuxjournal.com and we'll run it in the next Letters section and post it on the website as an addendum to the original article. ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Decentralized
Certificate Authority
and Naming

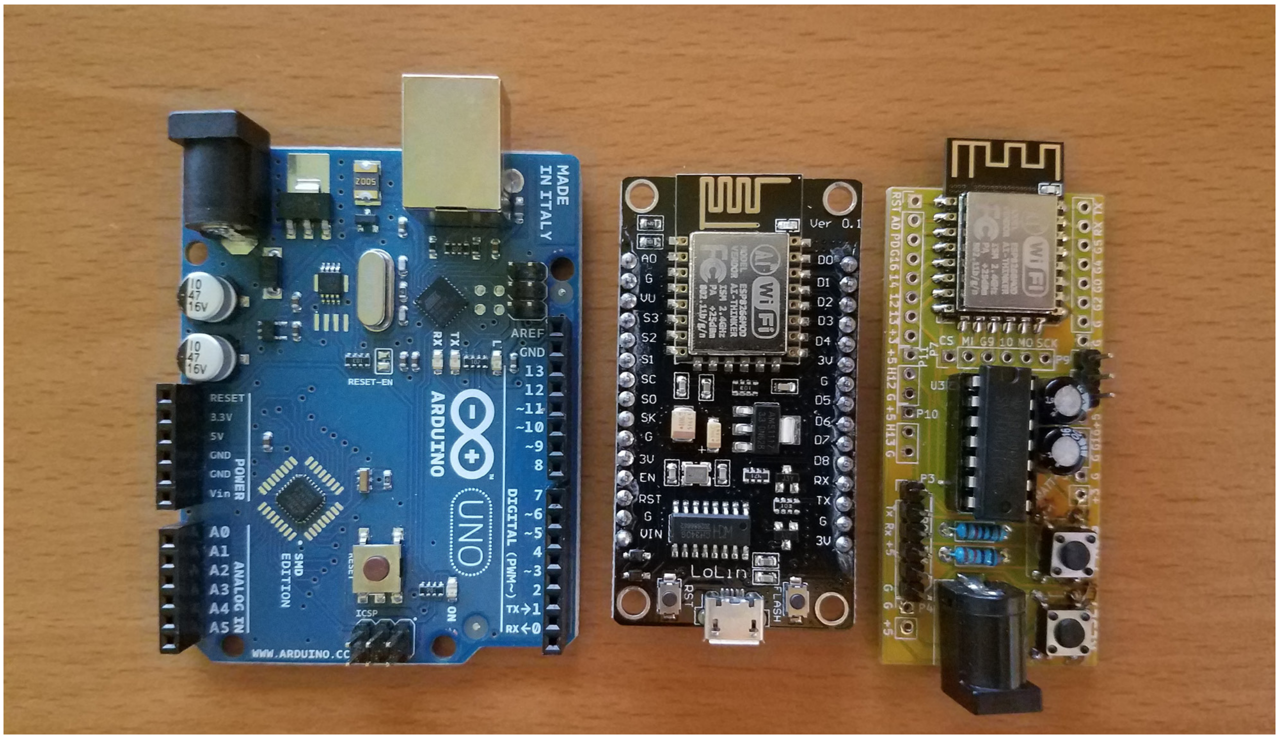
Free and open source contributors only:

handshake.org/signup

DEEP DIVE

SINGLE-BOARD COMPUTERS

Arduino from the Command Line: Break Free from the GUI with Git and Vim!



Love Arduino but hate the GUI? Try arduino-cli.

By Matthew Hoskins

In this article, I explore a new tool released by the Arduino team that can free you from the existing Java-based Arduino graphical user interface. This allows developers to use their preferred tools and workflow. And perhaps more important, it'll enable easier and deeper innovation into the Arduino toolchain itself.

The Good-Old Days

When I started building hobby electronics projects with microprocessors in the 1990s, the process entailed a discrete processor, RAM, ROM and masses of glue logic chips connected together using a point-to-point or "wire wrapping" technique. (Look it up kids!) Programs were stored on glass-windowed EPROM chips that needed

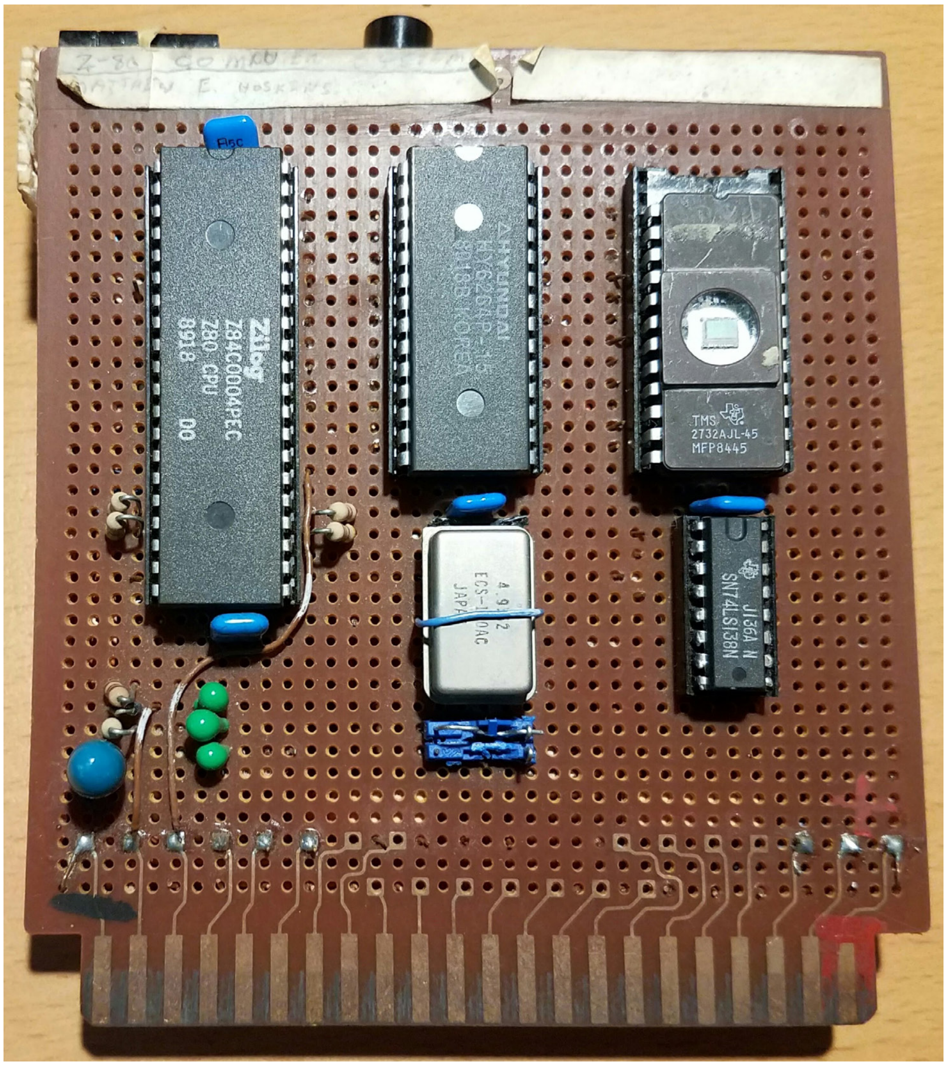


Figure 1. Example Mid-1990s Microprocessor

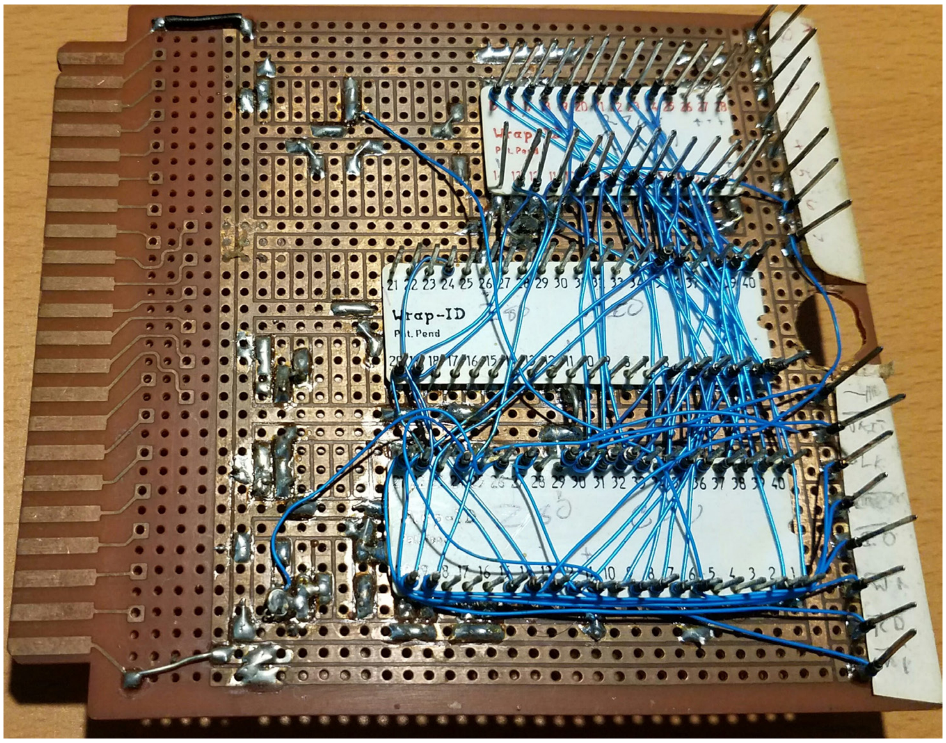


Figure 2. Example Mid-1990s Microprocessor

to be erased under UV light. All the tools were expensive and difficult to use, and development cycles were very slow. Figures 1–3 show some examples of my mid-1990s microprocessor projects with discrete CPU, RAM and ROM. Note: no Flash, no I/O, no DACs, no ADCs, no timers—all that means more chips!

It all changed in 2003 with Arduino.

The word “Arduino” often invokes a wide range of opinions and sometimes emotion. For many, it represents a very low bar to entry into the world of microcontrollers. This world before 2003 often required costly, obscure and closed-source development tools. Arduino has been a great equalizer, blowing the doors off the walled garden. Arduino now represents a huge ecosystem of hardware that speaks a (mostly) common language and eases transition from one hardware platform to another. Today, if you are a company that sells microcontrollers, it’s in your best interest to get your dev boards working with Arduino. It offers a low-friction path to getting your products into lots of hands quickly.

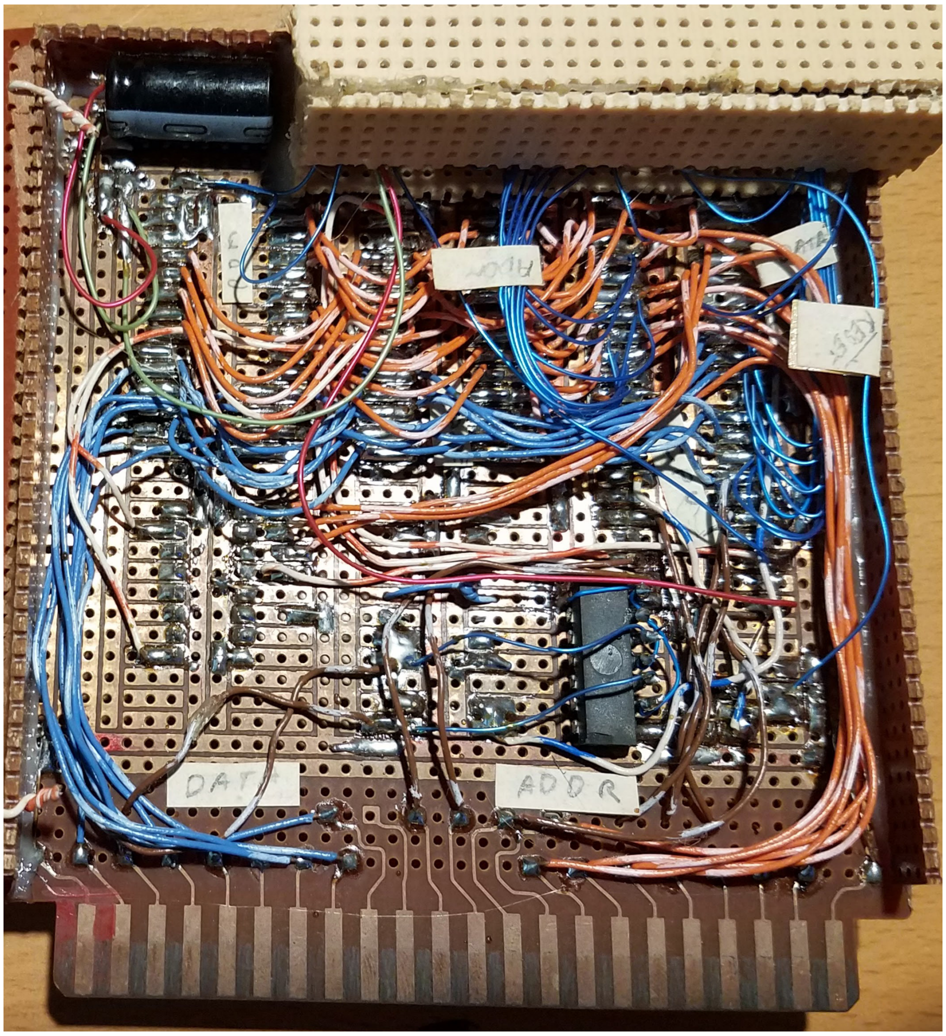


Figure 3. Example Mid-1990s Microprocessor

It's also important to note that Arduino's simplicity does not inhibit digging deep into the microcontroller. Nothing stops you from directly twiddling registers and using advanced features. It does, however, decrease your portability between boards.

For people new to embedded development, the Arduino software is exceptionally easy to get running. That has been a major reason for its success—it dumps out of the box ready to rock. But for more seasoned developers, the included graphical user interface can be frustrating. It can be a barrier to using modern development tools and version control like Git. I know the compiler and toolchain is buried deep in that GUI somewhere. I just want to use my favorite editor, compile my code and upload my project to a dev board using my favored workflow. For many developers, this is a command-line or scripted process.

Enter arduino-cli

There have been a couple attempts to break out Arduino to the command line, but most failed to get wide support. However, now the Arduino team has alpha-released arduino-cli. This new framework not only provides a comprehensive set of command-line functions, but the team also says it will be used as the core underneath the next generation of the Arduino graphical interface. This is exciting news and shows commitment to this new concept.

For me, my preferred development workflow is using Git for version control and the Vim editor, so that's what I demonstrate in the remainder of this article.

Installing arduino-cli

At the time of this writing, the arduino-cli is in alpha release. It's being distributed both as a Go source package and pre-built binaries. The Go language produces very portable binaries, so you just need to download the correct file and put the binary somewhere in your $PATH. Most users reading this likely will want the Linux 64-bit for Intel/AMD systems. In the examples here, my system happens to be running Fedora 29, but any recent Linux should work. Check the project's GitHub page for updated versions; at the time of this writing, 0.3.2 is the latest alpha release. Finally, make sure your user can access serial- and USB-connected Arduino devboards by adding them to the "dialout" group (note: you'll need to re-log in to pick up the new group membership, and substitute "me" for your user name in the last command):

```
me@mybox:~ $ wget https://downloads.arduino.cc/arduino-cli/
↪arduino-cli-0.3.2-alpha.preview-linux64.tar.bz2
***downloading***
me@mybox:~ $ tar -xjf arduino-cli-0.3.2-alpha.preview-
↪linux64.tar.bz2
me@mybox:~ $ sudo mv arduino-cli-0.3.2-alpha.preview-linux64
 ↪/usr/local/bin/arduino-cli
me@mybox:~ $ sudo usermod -a -G dialout me
```

Assuming /usr/local/bin is in your $PATH, you should be able to run arduino-cli as any user on your Linux system.

Alternatively, if you want to build the Go package from source, you can use the `get` function to download, build and install the source package from the Arduino GitHub repository:

```
me@mybox:~ $ sudo dnf -y install golang
me@mybox:~ $ cd ; export GOPATH='pwd'/go
me@mybox:~ $ go get -u github.com/arduino/arduino-cli
me@mybox:~ $ sudo mv $GOPATH/bin/arduino-cli /usr/local/bin/
```

Arduino from the Command Line

First, let's do some housekeeping. You need to pull over the current index of Arduino "cores" and search for the core that supports your dev board. In this first example, let's install support for the classic UNO board powered by an ATMega AVR processor:

```
me@mybox:~ $ arduino-cli core update-index
Updating index: package_index.json downloaded
me@mybox:~ $ arduino-cli core search avr
Searching for platforms matching 'avr'

ID                    Version   Name
arduino:avr           1.6.23    Arduino AVR Boards
arduino:megaavr       1.6.24    Arduino megaAVR Boards
atmel-avr-xminis:avr  0.6.0     Atmel AVR Xplained-minis
emoro:avr             3.2.2     EMORO 2560
littleBits:avr        1.0.0     littleBits Arduino AVR Modules

me@mybox:~ $ arduino-cli core install arduino:avr
*** lots of downloading omitted ***
me@mybox:~ $ arduino-cli  core list
ID                   Installed   Latest   Name
```

```
arduino:avr@1.6.23   1.6.23      1.6.23  Arduino AVR Boards
```

That's it. You have everything you need to create a new Arduino project for an Arduino UNO board. Now, let's create a project called myBlinky. You'll also initialize and set up a Git repository to manage version control, then make your first commit:

```
me@mybox:~ $ arduino-cli sketch new myBlinky
Sketch created in: /home/me/Arduino/myBlinky
me@mybox:~ $ cd /home/me/Arduino/myBlinky
me@mybox:~/Arduino/myBlinky $ git config --global
 ↪user.email "me@mybox.com"
me@mybox:~/Arduino/myBlinky $ git config --global
 ↪user.name "My Name"
me@mybox:~/Arduino/myBlinky $ git init
Initialized empty Git repository in /home/me/Arduino/
↪myBlinky/.git/
me@mybox:~/Arduino/myBlinky $ ls -la
total 16
drwxr-xr-x 3 me me 4096 Nov 22 10:45 .
drwxr-xr-x 3 me me 4096 Nov 22 10:45 ..
drwxr-xr-x 7 me me 4096 Nov 22 10:45 .git
-rw-r--r-- 1 me me   35 Nov 22 10:45 myBlinky.ino

me@mybox:~/Arduino/myBlinky $ cat myBlinky.ino

void setup() {
}

void loop() {
}

me@mybox:~/Arduino/myBlinky $ git add -A
me@mybox:~/Arduino/myBlinky $ git commit -m "Initial Commit"
```

```
[master (root-commit) ee95972] Initial Commit
1 file changed, 6 insertions(+)
create mode 100644 myBlinky.ino

me@mybox:~/Arduino/myBlinky $ git log
commit ee9597269c5da49d573d6662fe8f8137083b7768
 ↪(HEAD -> master)
Author: My Name <me@mybox.com>
Date:   Thu Nov 22 10:48:33 2018 -0500

   Initial Commit
```

Nice! The tool creates the project under the same Arduino directory structure where the graphical tools would expect to find them allowing you to flip between tools if you wish. It also creates a template .ino file with the familiar `setup()` and `loop()` functions.

The empty Git repository was initialized, and you can see it created the .git subdirectory where all the version data will be kept. Time to code!

Now, simply open up myBlinky.ino in your preferred editor—which in the interest of maximum street cred is Vim of course. Never EMACS (joking!)...seriously, use any editor you like (I hear Eclipse is nice)—then type in and save a classic "hello world" blinky program. Something like this:

```
// Simple Demo Blinker -MEH
#define PIN_LED 13

void setup() {
       pinMode(PIN_LED,OUTPUT);
}

void loop() {
       digitalWrite(PIN_LED,HIGH);
       delay(500);
       digitalWrite(PIN_LED,LOW);
       delay(500);
}
```

Now, let's compile and upload it to the Arduino UNO. Use the `board` set of commands to search for the upload:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli   board list
FQBN             Port           ID          Board Name
arduino:avr:uno /dev/ttyACM0   2341:0001   Arduino/Genuino Uno
```

It found the board. Now compile:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli  compile -b
 ↪arduino:avr:uno
Build options changed, rebuilding all
Sketch uses 930 bytes (2%) of program storage space. Maximum
 ↪is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving
 ↪2039 bytes for local variables. Maximum is 2048 bytes.
```

Compile was clean. Next, upload to the board using the `upload` command, board name and port you discovered earlier:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli  upload
 ↪-b arduino:avr:uno -p /dev/ttyACM0
```

As is common with command-line tools, silence is golden. The `upload` command completes, and the UNO is happily blinking away. You better lock in this good fortune with a `git commit`:

```
me@mybox:~/Arduino/myBlinky $ git commit -a -m "It works!
 ↪First blink."
[master 35309a0] It works! First blink.
1 file changed, 7 insertions(+)
```

The `-m` option takes a commit message; it should be a note about what's included in this commit. If you omit the message, `git` will open a template message in a text editor (the default is Vim, but you can change it by setting `$EDITOR`).

Support for Third-Party Development Boards

Now for something a little more ambitious, let's try to use a non-Arduino development board and walk through the steps of adding a third-party core. This can be tricky even with the graphical user interface, but it's pretty simple with arduino-cli. Let's target the very popular ESP8266.

For starters, you need to add the third-party repository to a file so arduino-cli knows how to locate the core. This is done with a file named .cli-config.yml. You might think this should go in your home directory or in the project directory, and you would be right to think that. But, one quirk of this early release of arduino-cli is that this file lands in the same directory where the arduino-cli program lives. For now, this is /usr/local/bin, but keep an eye on the website, as this is likely to change in future releases! Below, you'll add a new board config definition. This file uses YAML format, so be careful to use only spaces in the indenting. Edit (with `sudo`), and place the following text in /usr/local/bin/.cli-config.yml:

```
board_manager:
 additional_urls:
   - http://arduino.esp8266.com/stable/
↪package_esp8266com_index.json
```

Now, like before, update the index and install the core:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli core update-index
```

```
Updating index: package_index.json downloaded
Updating index: package_esp8266com_index.json downloaded
```

You can see that it found and downloaded the index for esp8266 cores. Good. Now, let's download and install the core itself:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli core search esp
Searching for platforms matching 'esp'

ID                Version Name
esp8266:esp8266 2.4.2   esp8266

me@mybox:~/Arduino/myBlinky $ arduino-cli core install
 ↪esp8266:esp8266
Downloading esp8266:esptool@0.4.13...
esp8266:esptool@0.4.13 downloaded  *** much omitted ***
```

Now, you can rebuild your myBlinky project for the esp8266 and upload it. You first need to edit your myBlinky.ino and change the `#define PIN_LED` to whichever pin has the LED. On my dev board, it's pin 2. Make that modification and save it:

```
#define PIN_LED 2
```

After plugging in the esp8266 dev board, you again run the `board list` command to try to find it:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli board list
FQBN    Port            ID              Board Name
       /dev/ttyUSB0    1a86:7523       unknown
```

It detects it, but it can't determine what it is. That is quite common for boards like the esp8266 that require a button push or special programming mode.

Next, compile and upload after resetting your esp8266 board while pressing the program button. Like last time, use the board name and port discovered during the `list` operation:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli board listall
 ↪|grep esp8266
Generic ESP8266 Module                  esp8266:esp8266:generic
(omitted long list)
me@mybox:~/Arduino/myBlinky $ arduino-cli upload  -b
 ↪esp8266:esp8266:generic -p /dev/ttyUSB0
Uploading 252000 bytes from /home/me/Arduino/myBlinky/
↪myBlinky.esp8266.esp8266.generic.bin to flash at 0x00000000
.................................................... [ 32% ]
.................................................... [ 64% ]
.................................................... [ 97% ]
.......                                              [ 100% ]
```

And, it blinks! Make another `git commit` and save your progress:

```
me@mybox:~/Arduino/myBlinky $ git add -A
me@mybox:~/Arduino/myBlinky $ git commit  -m
 ↪"Blinking on esp8266 board"
[master 2ccff1d] Blinking on esp8266 board
5 files changed, 61 insertions(+), 1 deletion(-)
create mode 100755 myBlinky.arduino.avr.uno.elf
create mode 100644 myBlinky.arduino.avr.uno.hex
create mode 100644 myBlinky.esp8266.esp8266.generic.bin
create mode 100755 myBlinky.esp8266.esp8266.generic.elf
```

As you can see, it saves the compiled binary versions of the compiled code in the project directory. You can add *.bin, *.hex and *.elf to a .gitignore file if you wish to omit these from your commits. If you do save them, you can use the `-i` option and the .bin file to upload a specific binary.

Adding Libraries

Building on your success, you should download and install a library. Let's up the blinky game and install support for some Adafruit NeoPixels (aka WS2812B, PL9823 and so on). First, search for it using the `lib` command, then download and install:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli  lib  search neopixel
(omitting large list)
me@mybox:~/Arduino/myBlinky $ arduino-cli  lib  install
 ↪"Adafruit NeoPixel"
Adafruit NeoPixel@1.1.7 downloaded

Installed Adafruit NeoPixel@1.1.7
```

Now you need to modify the program; edit the .ino file with these modifications:

```
// Fancy NeoPixel Blinky Blinker

#include <Adafruit_NeoPixel.h>
#define PIN_LED 14

Adafruit_NeoPixel strip = Adafruit_NeoPixel(1, PIN_LED,
 ↪NEO_GRB + NEO_KHZ800);

void setup() {
       strip.begin();
}

void loop() {

       strip.setPixelColor(0,strip.Color(255,0,0))'
       delay(200);
       strip.setPixelColor(0,strip.Color(0,255,0));
       delay(200);
```

```
        strip.setPixelColor(0,strip.Color(0,0,255));
        delay(200);
}
```

Next, do the same compile and upload dance, after, of course, attaching a NeoPixel or compatible LED to pin 14:

```
me@mybox:~/Arduino/myBlinky $ arduino-cli compile -b
 ↪esp8266:esp8266:generic
Build options changed, rebuilding all
Sketch uses 248592 bytes (49%) of program storage space.
 ↪Maximum is 499696 bytes.
Global variables use 28008 bytes (34%) of dynamic memory,
 ↪leaving 53912 bytes for local variables. Maximum
 ↪is 81920 bytes.
me@mybox:~/Arduino/myBlinky $ arduino-cli upload  -b
 ↪esp8266:esp8266:generic -p /dev/ttyUSB0
Uploading 252960 bytes from /home/me/Arduino/myBlinky/
↪myBlinky.esp8266.esp8266.generic.bin to flash at 0x00000000
....................................................  [ 32% ]
....................................................  [ 64% ]
....................................................  [ 96% ]
........                                              [ 100% ]
```

And, you should have a colorful blinky—pretty slick. It's another good time to commit your changes to capture your progress:

```
me@mybox:~/Arduino/myBlinky $ git add -A
me@mybox:~/Arduino/myBlinky $ git commit -m
 ↪"Blinky with NeoPixels"
[master 122911f] Blinky with NeoPixels
3 files changed, 20 insertions(+), 13 deletions(-)
rewrite myBlinky.ino (81%)
```

Using GitHub

Up to now, the git repository has been completely local in your project directory. There's nothing wrong with that, but let's say you want to publish your work to GitHub. It's pretty quick and easy to get started. First, log in to github.com and create an account if you don't already have one. Then, click the button to create a "new repository".

Fill in the details for your project, and be sure to leave unchecked the option to initialize the repository with a README. This is because you already have a

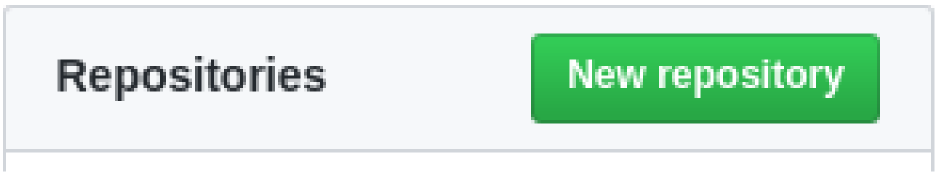
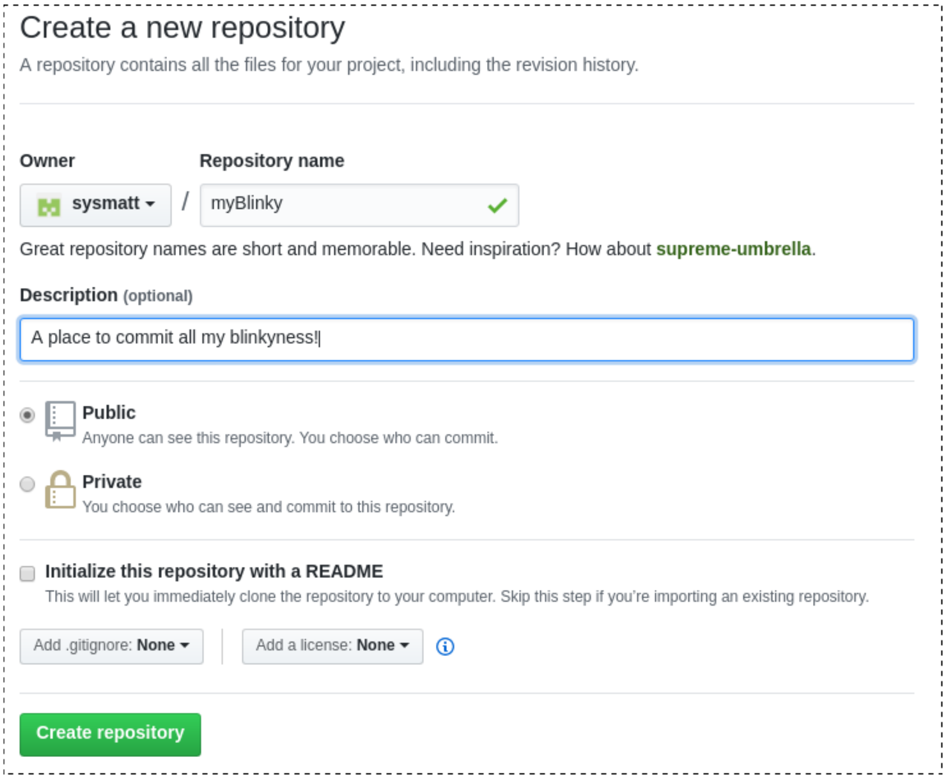


Figure 4. New Repository



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: sysmatt / Repository name: myBlinky

Great repository names are short and memorable. Need inspiration? How about supreme-umbrella.

Description (optional): A place to commit all my blinkyness!

[✓ Public] Anyone can see this repository. You choose who can commit.

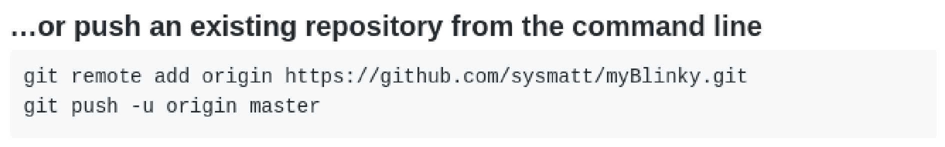
Private — You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Create repository

Figure 5. Create Repo


...or push an existing repository from the command line

```
git remote add origin https://github.com/sysmatt/myBlinky.git
git push -u origin master
```

Figure 6. Push Existing Repository

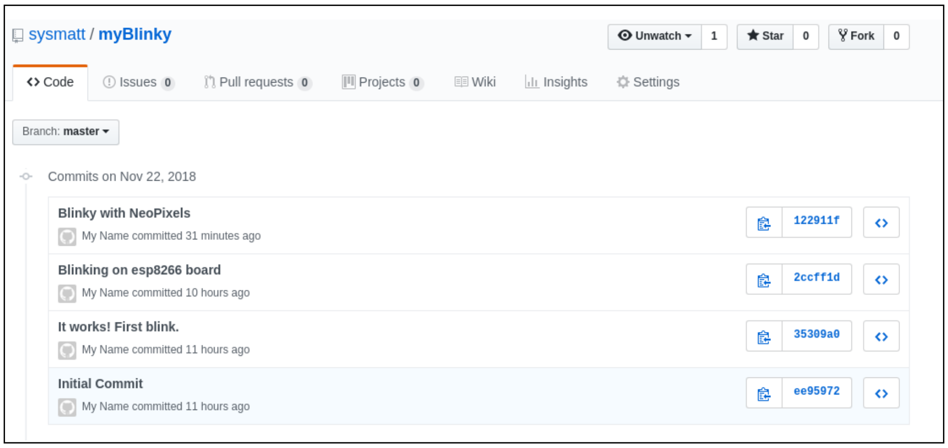
repository created, and you'll be "pushing" your local repository to GitHub, so you want it empty.

After creating it, you will be presented with some helpful options on how to push code into GitHub. You want the commands for "push existing repository".

Now let's do it! Follow the instructions to create a git "remote" entry named "origin", which will represent GitHub. Then you will push, and it will prompt you for your GitHub user name and password (substitute your own GitHub URL, user name and password):

```
me@mybox:~/Arduino/myBlinky $ git remote add origin
 ↪https://github.com/sysmatt/myBlinky.git
me@mybox:~/Arduino/myBlinky $ git push -u origin master
Username for 'https://github.com': sysmatt
Password for 'https://sysmatt@github.com':  ******************
Counting objects: 18, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (15/15), done.
Writing objects: 100% (18/18), 825.31 KiB | 5.00 MiB/s, done.
Total 18 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
* [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Now, if you browse to the repository on GitHub.com, you'll see all your files and every one of your commits. You can travel back in time and look at every



sysmatt / myBlinky

Branch: master

Commits on Nov 22, 2018

Blinky with NeoPixels
My Name committed 31 minutes ago — 122911f

Blinking on esp8266 board
My Name committed 10 hours ago — 2ccff1d

It works! First blink.
My Name committed 11 hours ago — 35309a0

Initial Commit
My Name committed 11 hours ago — ee95972

Figure 7. Commit History

revision. In practice, you can use SSH keys to eliminate having to enter your password every time.

Time Travel (aka Breaking the Laws of Physics with Git)

Time travel, you say? Yes indeed. Let's say you want to jump back in time and review your version of myBlinky from when you had it working with the Arduino UNO. It seems like ages ago. It's easy! You just need to identify the commit id and "checkout" that version.

Use the `log` command to list all your commits:

```
me@mybox:~/Arduino/myBlinky $ git log
commit 122911f99dddce2dabbb251c3b640c8c7f9f98d9 (HEAD ->
 ↪master, origin/master)
Author: My Name <me@mybox.com>
Date:   Thu Nov 22 21:22:59 2018 -0500

   Blinky with NeoPixels

commit 2ccff1d7326b1523649510f24644c96df6dc6e12
Author: My Name <me@mybox.com>
```

```
Date:   Thu Nov 22 11:42:02 2018 -0500

   Blinking on esp8266 board
commit 35309a0c9e90668052abc9644f77f906ab57949c
Author: My Name <me@mybox.com>
Date:   Thu Nov 22 11:09:44 2018 -0500

   It works! First blink.

commit ee9597269c5da49d573d6662fe8f8137083b7768
Author: My Name <me@mybox.com>
Date:   Thu Nov 22 10:48:33 2018 -0500

   Initial Commit
```

It looks like the commit starting with 35309a0c is the one you're after. Note: you can shorten the commit hash string to as few as four characters, as long as it uniquely identifies only one commit. Let's explore that version:

```
me@mybox:~/Arduino/myBlinky $ git checkout 35309a0c
HEAD is now at 35309a0... It works! First blink.
me@mybox:~/Arduino/myBlinky $ ls -l
-rw-r--r-- 1 me me  191 Nov 22 22:01 myBlinky.ino
me@mybox:~/Arduino/myBlinky $ cat myBlinky.ino
// Simple Demo Blinker -MEH
#define PIN_LED 13

void setup() {
        pinMode(PIN_LED,OUTPUT);
}

void loop() {
        digitalWrite(PIN_LED,HIGH);
        delay(500);
        digitalWrite(PIN_LED,LOW);
        delay(500);
}
```

Now let's say you're done poking around, so let's time travel forward to the current day and get things back to before you broke the laws of physics. In the simple git repository, this means jumping back to the current commit in the "master" branch:

```
me@mybox:~/Arduino/myBlinky $ git checkout master
Previous HEAD position was 35309a0... It works! First blink.
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
me@mybox:~/Arduino/myBlinky $ ls -l
total 1332
-rwxr-xr-x 1 me me   13956 Nov 22 22:04
 ↪myBlinky.arduino.avr.uno.elf
-rw-r--r-- 1 me me    2640 Nov 22 22:04
 ↪myBlinky.arduino.avr.uno.hex
-rw-r--r-- 1 me me  252960 Nov 22 22:04
 ↪myBlinky.esp8266.esp8266.generic.bin
-rwxr-xr-x 1 me me 1082905 Nov 22 22:04
 ↪myBlinky.esp8266.esp8266.generic.elf
-rw-r--r-- 1 me me     436 Nov 22 22:04 myBlinky.ino
```

Nice! You're back to the NeoPixel blinky. You see, git stores all the versions of your committed code under the .git subdirectory. This is actually the real location of the repository. The files you edit are just the "work area". When you jump around between commits, git is doing all the magic of modifying the files in the work area. If you wanted to jump back and start modifying the old version of code, you could create a new "branch" to contain that work and move forward with an AVR and esp8266 fork of your code. It's very powerful.

I've barely scratched the surface here. Git, GitHub and arduino-cli are all quite comprehensive tools. I hope this article has given you a taste of what's possible when you harness good programming workflows for your Arduino projects. ■

Matthew Hoskins is the Senior Enterprise Architect at New Jersey Institute of Technology where he leads the team of talented professionals that keep our dizzying array of services working together. Systems, storage, on-premises and in the cloud, we do it all. When it's done right, no one notices. Matt is @SYSMATT on Twitter.

Resources

- Arduino-CLI Announcement
- Arduino-CLI GitHub

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Indie Makers Using Single-Board Computers

Possibly the most amazing thing, to me, about single-board computers (SBCs) is that they allow small teams of people (and even lone individuals) to create new gadgets using not much more than SBCs and 3D printers. That opportunity for makers and small companies is absolutely astounding.

By Bryan Lunduke

Two such projects have really caught my attention lately: the Noodle Pi and the TinyPi.

The Noodle Pi is a simple, handheld computer (about the size of a deck of playing cards). And, when I say simple, I mean *simple*. It's got a micro-USB charging port, another for plugging in USB devices, a touch screen and a battery. Think of it like an old-school PDA without any buttons (other than a small power toggle) and the ability to run a full Linux-based desktop.

The TinyPi is a gaming handheld. And, believe it or not, it's even smaller than the Noodle Pi, with a tiny screen and tiny buttons. This is the sort of handheld game console you could put on a keychain.

Both of these are built on top of the (super-tiny and super-cheap) Raspberry Pi Zero. And, both are built by lone individuals with a heavy reliance on 3D printers.

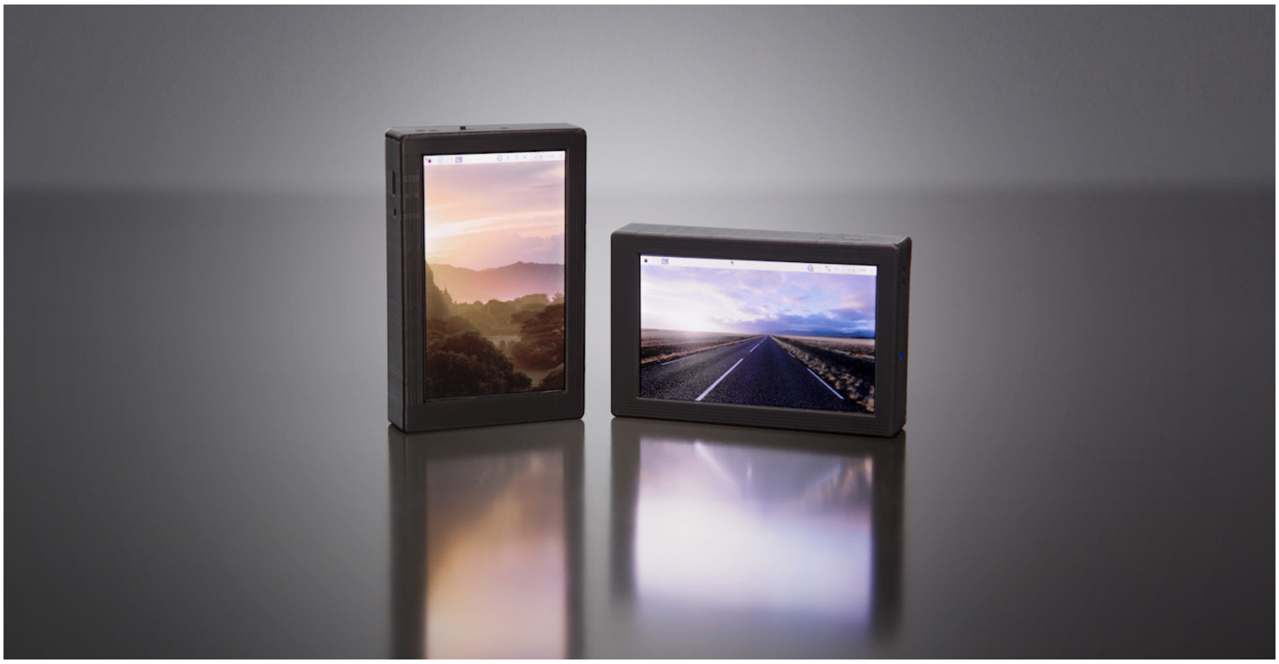


Figure 1. The Credit-Card-Sized, Pi Zero-Powered, Noodle Pi

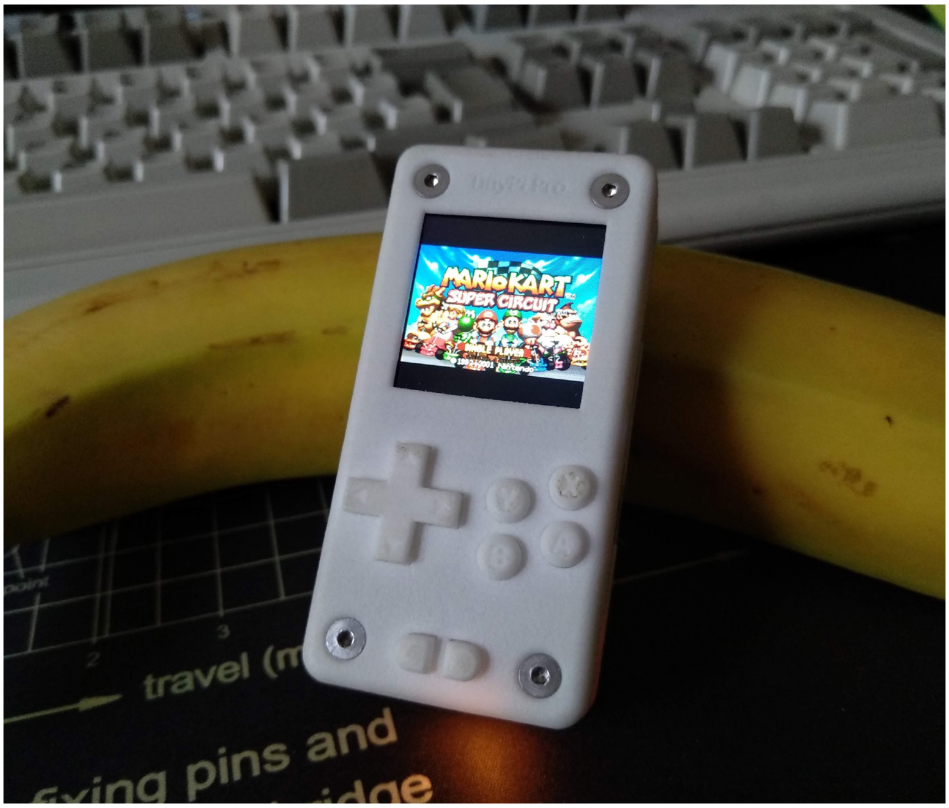


Figure 2.
The Impossibly
Small TinyPi
(Banana for Scale)

I wanted to know how they did it and how their experience was. What can we learn from these independent gadget makers? So, I reached out to both of them and asked them each the same questions (more or less).

Let's start with a chat with Pete Barker (aka "pi0cket"), maker of the TinyPi.

Interview with Pete Barker (pi0cket), TinyPi Maker

Bryan Lunduke: Could you give a quick overview of the TinyPi?

Pete Barker: TinyPi is (unofficially) the world's smallest pi-based gaming device. It started life as a bit of a joke—"how small can i make this?"—but it actually turned into something pretty good. The Pro version added more features and improvements, and a kickstarter was funded on December 30, 2018. Manufacturing is already underway, and the early-bird backers should start getting the kits in February 2019.

BL: When you first started working on the TinyPi, was it always envisioned to be based on a Raspberry Pi Zero?

PB: It actually was the release of the Zero that sparked the project. I already had been playing with the Raspberry Pi, but the reduced footprint was a game-changer.

BL: Any other boards that were considered back then?

PB: At the time, the zero was the best out there for the price and size. There were some "banana" and "orange" pis but they tended to be a little on the large size.

BL: Did the Pi Zero meet your needs? Was it...fast enough?

PB: We always want more speed. The Zero would be great if it had the power of a Pi3, but then it would eat the batteries. The Zero does a great job of emulating retrogames, and it also works well for other things.

BL: If you had it to do all over again, what sort of single-board computer would you use?

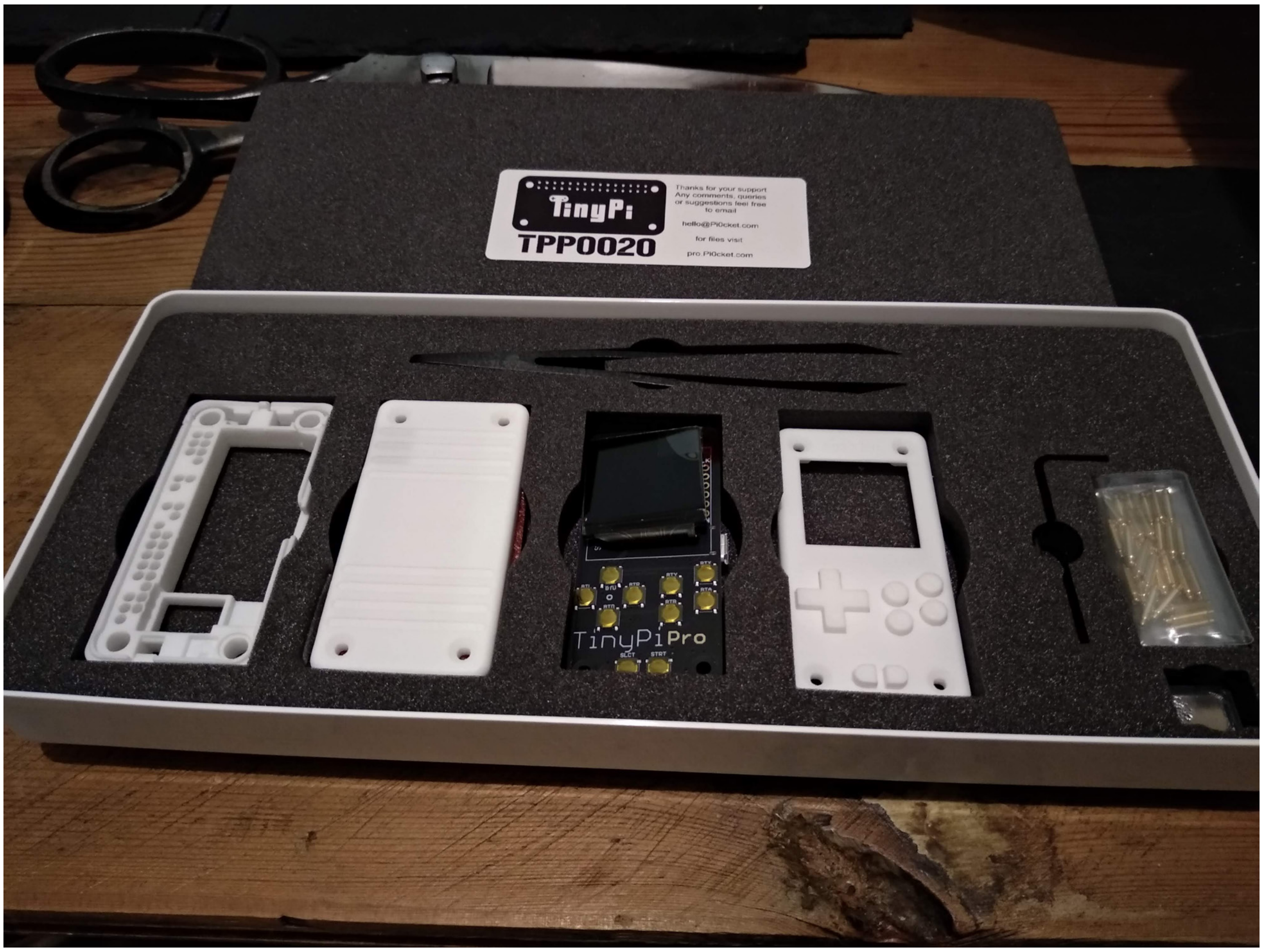


Figure 3. The parts of the TinyPi—the Smallest Handheld Game Console I Can Possibly Imagine

PB: I have been investigating alternatives. There are a few "NanoPi" boards that have quad-core processors in small footprints. There is also the BeagleBone on-chip, which lets you build your own SBC with very few components. The biggest draw to the Raspberry Pi boards is the software support. Because the RPis are super popular and well established, there's a fast range of software ready to roll.

BL: What part of building something around a single-board computer was the least enjoyable? Any headaches?

PB: There were no real headaches. It's all been a learning curve throughout the project. Two years ago, I had never designed a PCB or used any form of CAD.

BL: Do you use Pis (or other similar boards) in your own life, outside the TinyPi?

PB: I actually have a scary number of Raspberry Pis in my house. One is running DNS filtering and freeview decoding, and one is on each TV playing media from a NAS box. I have multiple handhelds filled with Pi, and I'm even working on a pocket-sized keyboard and screen combo called "clicker".

BL: If you could change one thing about the Pi Zero, what would it be?

PB: A Pi Zero with the power of the Pi3 would be great, but I don't think that will become a reality. And a smaller footprint with maybe castellated edges would mean that I could make something even smaller (something like this, but not $99).

And now let's pose those same questions to Ashish Gulhati, maker of the Noodle Pi.

Interview with Ashish Gulhati, Noodle Pi Maker

Bryan Lunduke: What is the Noodle Pi?

AG: Okay, so the Noodle Pi is a handheld PC based on the Raspberry Pi Zero, with a hi-res touchscreen, and integrated camera and battery. End users can assemble and disassemble it, with no tools or soldering required. A modular docking system enables you to pair it up with a variety of keyboards and gamepads, and you also can wear it on your wrist or in a holster clipped to a belt or pocket. Noodle Air is an air-gapped version of Noodle Pi, intended for high-security applications.

Noodle Pi was launched on Kickstarter in July 2017, and it has shipped to backers in 17 countries. Sadly, the HyperPixel display used in the Noodle Pi was discontinued less than a year after launch, so the original Noodle Pi is currently unavailable. An updated version is in the works and should be available soon.

I'm planning to launch a Kickstarter for a different Noodle Pi device very soon (not the updated version of the original Noodle Pi, which will be next).

BL: When you first started working on the Noodle Pi, did you always envision that it'd be based on a Raspberry Pi Zero?

AG: In July 2016, when I decided to make a handheld device as a deployment platform for Unsnoopable (my app for completely unsnoopable messaging), I envisioned that device as based on the Pi Zero right from the start.

But I've been trying to put together a practical, open and flexible wearable computer for almost two decades.

Around 2012, I hacked together a few prototype wearable Noodle computers for my own use. Some of them were based on the original Raspberry Pi Model B released in February 2012.

My first wearable Linux computer, back in 2001, was based on a Sony VAIO Picturebook (with a Transmeta Crusoe processor!), a Daeyang CyVisor head-mounted display and a Twiddler chorded keyboard.

BL: Did you consider any other boards back then?

AG: The Pi Zero was pretty much the only board small enough to fit into a reasonably small handheld, with the ability to run a full Linux system and for which some compatible touchscreen displays were readily available. I also wanted to include a camera in the device, and the Pi Zero v1.3, which included a camera connector, had just been released. So that was perfect.

I did later consider the Pi Compute Module 3 Lite, which has a faster processor than the Pi Zero. However, that would need a carrier board, and together they'd end up being too big.

The original Noodle Pi design in July 2016 also required a second board, or additional components soldered onto the Pi Zero's USB pads, to add WiFi and Bluetooth support, which made it very difficult to cram everything into the form factor of device I was aiming for.

So for a while, I procrastinated on the Noodle Pi and worked on some other prototypes based on the Jaguar Board. The procrastination strategy worked, and ideas from those prototypes helped with the design of the Noodle Pi.

Procrastinating on the Noodle Pi also paid off when the need for additional components for wireless functionality was neatly eliminated in February 2017 with the launch of the Pi Zero W, which added onboard WiFi and Bluetooth.

The touchscreen was the final sticking point as none of the small screens available in early 2017 offered a reasonably high resolution, and they required soldering and destructive modifications to fit into a reasonable device form factor. That was solved in June 2017 with the HyperPixel display.

BL: Did the Pi Zero meet your needs? Was it...fast enough?

AG: Yes, I think the Pi Zero is a fantastic little SBC, and it was very cool how it got the camera interface and the WiFi/Bluetooth upgrades just as I needed them for the Noodle Pi—and the HyperPixel display. With all these capabilities, I think it's easily the most versatile and feature-packed SBC of its size. In 2018, the Pi Zero WH version was launched, which includes a pre-soldered header. That simplifies the assembly of the Noodle Pi.

Speed definitely is a bit of a constraint on the Pi Zero, but then my main goal was to deploy fairly lightweight apps written in Perl, and the Pi Zero is more than capable of handling that, and of scanning QR codes using its camera, which is important for some of my apps.

Compared to the first computer I owned, an IBM PC XT with an 8088 @ 4.77 MHz and 640KB of RAM, the Pi Zero is a blazing speed demon. It's great for console apps, and it runs most GUI apps just fine too.

Of course, it isn't really suited for some applications, but the great thing is that you can just pull out your MicroSD card from a Pi Zero, pop it into a Pi 3, and you have the same system running on a much faster computer. I use this trick quite often.

You can use a Noodle Pi on the go, and at home/office, just pop the MicroSD into a Pi 3 connected to a big screen and full keyboard, and presto, it's a faster more powerful desktop machine!

BL: If you had it to do all over again, what sort of single-board computer would you use?

AG: For a small integrated handheld device, I'd stick with the Pi Zero. I'm working on the updated version of the Noodle Pi, which should be available in the near future.

I'm also working on a slightly different variation of the Noodle Pi, which I plan to launch first, probably in February 2019.

I picked up some NanoPi SBCs to play around with. There are some interesting possibilities there, but they don't have integrated WiFi/Bluetooth antennas like the Pi Zero W does, so the Zero still wins for wireless connectivity.

There's also the Banana Pi Zero, which could potentially be a nice drop-in upgrade for the Pi Zero, but I wasn't able to get it to boot up anything the last time I tried. And it doesn't have integrated wireless antennas either.

BL: What part of building something around a single-board computer was the least enjoyable? Any headaches?

AG: For the most part, it was and continues to be quite enjoyable. I love working with SBCs, and it's great that there's such a variety of them available now. One thing I don't particularly enjoy is soldering, which is why I wanted to make sure that the Noodle Pi could be assembled without any soldering required.

Back in the early 1990s, when I got started with digital electronics, I enjoyed making prototype circuit boards using wire-wrapping. That was great fun. I made a whole custom 68K-powered computer that way.

With the original Pi model B in 2012, the USB ports were quite flaky and that was a pain, but the Pi Zero's USB is fine.

The headaches were more on the production side of things, with endless tiny tweaks and prototyping required to get the model and print parameters just right, and with various malfunctions and maintenance issues with the 3D printers during production. I'm hoping we'll see some significant advances in consumer 3D printing tech soon, leading to big reliability and quality improvements.

BL: Do you use Pis (or other similar boards) in your own life, outside the Noodle Pi?

AG: Absolutely. In fact, I now rely on SBCs for nearly all of my computing (other than production servers). I have quite a variety of prototype Noodle devices based on Pis and other SBCs, which serve as my main workstations and mobile computers. I hope to launch some more of these as Noodle products over the next few months.

I also have a bunch of boards in clusters for testing the HashCash vault and some other projects.

I've used every conceivable type of mobile computer during the past 25 years, and I've never really found the perfect sweet spot of mobility, price and power with laptops and commercially available handheld devices, which is what led me to keep hacking up Noodle computers.

I think the modularity afforded by SBCs is a big win. With Noodle computers based on SBCs, I don't have to worry about a screen or battery or keyboard problem rendering my whole machine unusable. I can just quickly swap out parts and get back to work.

I've had work disrupted way too many times because of such issues with laptops, which could be repaired only by the authorized repair center. That's no good for me. I need computers that I can fix myself in five minutes.

This is why I haven't been tempted to pick up a GPD Pocket or the Gemini PDA, which I would have paid big bucks for five years ago. After switching mainly to Noodle computers based on SBCs, I find that the modularity and repair-ability are just too important to give up.

BL: If you could change one thing about the Pi Zero, what would it be?

AG: If a genie granted me one change to the Pi Zero, I'd say: make it completely open with no proprietary blob required to boot up.

Of course, it'd also be great to see it updated to a faster multi-core CPU. But for the most part, I think it's an awesome SBC that packs incredible capabilities into a really tiny package.

BL: Any other anecdotes or interesting stories you'd like to share about the process of building a Pi Zero-based handheld?

AG: This isn't specifically about building a Pi Zero handheld, but I find it fun and interesting how procrastinating at various points allowed the available tech time to catch up to what I was trying to do.

And given that my main goal with the Noodle Pi was just to be able to deploy my Perl apps on an open handheld device, had I procrastinated a bit longer, I probably could just have done that on the upcoming Purism Librem 5 and saved myself a lot of work!

On the other hand, it's nice to have had the Noodle Pi as a development platform during the past year and a half. Apps written for the Noodle Pi should be easy to deploy on the Librem 5 when it ships, hopefully with minimal changes to the code.

Conclusion

Big round of thanks to both Pete Barker and Ashish Gulhati for taking the time to share their experiences in building devices based on the Pi Zero. I love seeing independent and creative makers designing (and shipping) cool little niche Linux computers like this. I'm absolutely positive I'm not alone in hoping that the coming years sees this trend increasing. ■

Bryan Lunduke is a former Software Tester, former Programmer, former VP of Technology, former Linux Marketing Guy (tm), former openSUSE Board Member... and current Deputy Editor of *Linux Journal* as well as host of the popular *Lunduke Show*. More details: http://lunduke.com.

Send comments or feedback
via http://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Mycroft: a Privacy-Respecting Digital Assistant

How to build a Mycroft skill and then convert the Google AIY Voice Kit to run Mycroft instead of Google Assistant.

By Jan Newmarch

Mycroft is a digital assistant along the lines of Google Home, Amazon Alexa and the many others that are currently coming onto the market. Unlike most of these though, it has two major differences:

- The code is open source, and you can install it on most Linux systems.
- There is a focus on privacy.

Many assistants will listen all of the time, sending everything they hear into the "cloud". As you all know, many of these companies are vacuums for data, and their business models are built on using that data. Mycroft goes against that model by minimizing the amount of data collected and anonymizing the data when it has to interact with other systems.

Google processing is given in Table 1.

Table 1. Google Assistant Query Processing

Local Execution	Remote Execution
Recognise wakeup word	
Capture query	
	Speech to text (recorded in your Google account)
	Match to intent
	Call matching skill
	Remote service if required by skill
	Generate response
	Text to speech

Mycroft changes this around to what's shown in Table 2.

Table 2. Mycroft Query Processing

Local Execution	Remote execution
Recognise wakeup word	
Capture query	
	Speech to text (anonymous)
Match to intent	
Call matching skill	
	Remote service if required by skill
Generate response	
	Text to speech

All processing is local except for the speech to text and text to speech modules, and possibly an external web service. The "speech to text" service is currently Google's Speech to Text service, but there is an option of using the Mozilla DeepSpeech engine. The requests are anonymized to be from "MycroftAI" rather than from a particular user. The "Remote service if required by skill" will depend on what needs to be done to satisfy the query—for example, getting the local time won't need one, while getting the weather will need access to a weather service, such as OpenWeatherMap. The "Text to speech" module, by default, uses a mechanical-sounding voice produced by software called Mimic 2, based on the Tacotron architecture, but the voice is configurable. However, there is a noticeable lag of several seconds while a voice response is prepared, and choosing a more complex voice increases this delay.

The Google AIY Voice Kit

You can buy digital assistants as "black boxes". Google has created a do-it-yourself "white box" (Figure 1). It consists of the following:

Figure 1. Google AIY Voice Kit

- A cardboard enclosure (yes okay, it's brown, not white).
- A speaker.
- A microphone.
- A "voice hat" board to process audio.
- A big button to give some privacy—the kit won't listen until you press the button.

The kit is driven by a Raspberry Pi 3, which you supply yourself.

You can download an image for the Raspberry Pi that can drive the voice hat, and there's a lengthy set of instructions to turn it into a Google Assistant. This is non-trivial, requiring a Google account and using OAuth for identification purposes. But I'm not going that route here, as this is a nice little box for installing Mycroft.

Installing Mycroft

Mycroft requires a good quality microphone and reasonable speaker, which the Raspberry Pi doesn't have. The AIY kit has these as well as the little brown box in which to put everything. But, the AIY image doesn't include Mycroft, and the Raspberry Pi image for Mycroft—picroft—doesn't have drivers for the voice hat.

There are two ways if getting Mycroft onto the Raspberry Pi you use with the AIY kit: add Mycroft to the AIY image or install AIY support onto the picroft image.

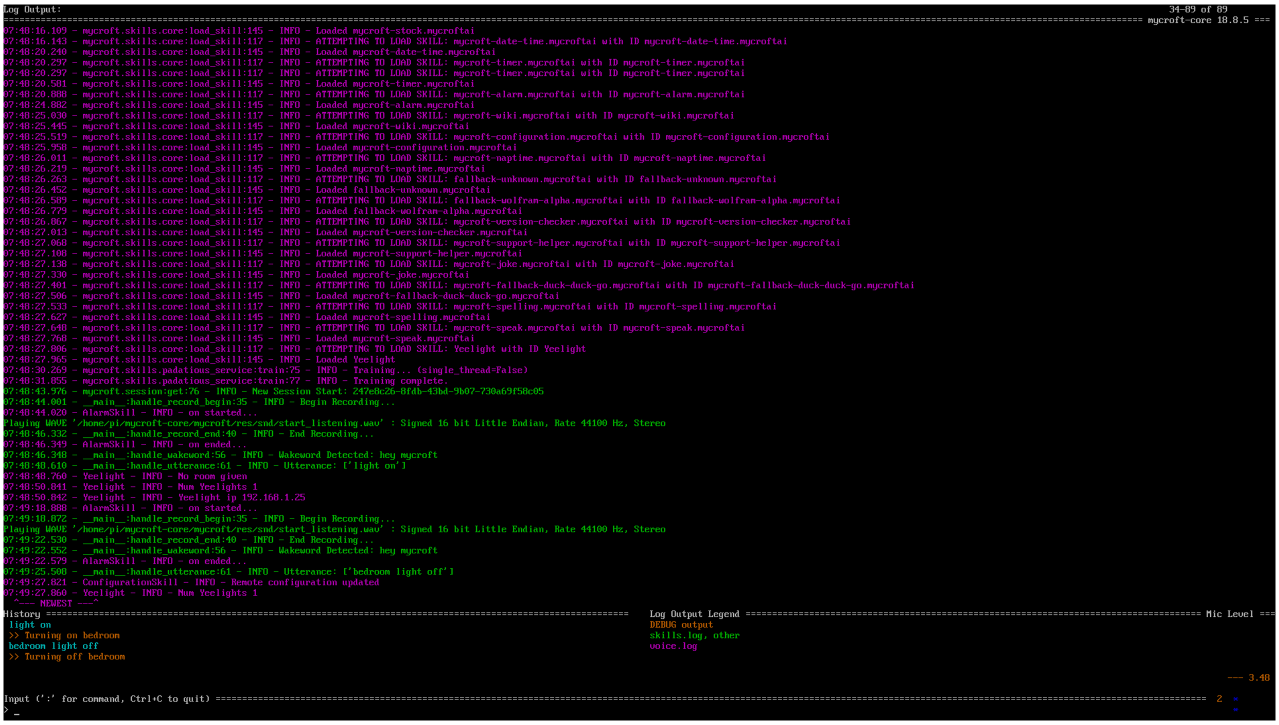
The first is straightforward but time-consuming: using the AIY image from here, download the Mycroft files from GitHub and build it on the Raspberry Pi. It's not hard.

The second is slightly less time-consuming: the Raspberry Pi image from September 2018 allows you to select the AIY devices during setup, and then downloads and builds all the relevant AIY files. At the time of this writing, Google has broken this by not having a particular Python "wheel" file available for the Raspberry Pi 3B or 3B+, but there is a workaround from andlo here. Hopefully this will be fixed soon—the AIY forums are very voluble on the subject!

To build this kit, follow the AIY instructions to connect everything together. But, don't fold them all into the cardboard enclosure yet, as that hides the USB ports and the HDMI port. Connect an external monitor, keyboard and mouse, so you can control the system while you build it. From then on, follow these steps:

- Using the AIY image: download and build Mycroft from GitHub.

- Using the picroft image: select the AIY option during setup, which should download and build lots of Python wheel files and libraries. While this is broken at the time of this writing (December 2018), hopefully Google will have fixed it by the time you read this.



```
Log Output:                                                                                                                34-89 of 89
============================================================================================================ mycroft-core 18.8.5 ===
07:48:16.109 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-stock.mycroftai
07:48:16.143 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-date-time.mycroftai with ID mycroft-date-time.mycroftai
07:48:20.240 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-date-time.mycroftai
07:48:20.297 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-timer.mycroftai with ID mycroft-timer.mycroftai
07:48:20.297 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-timer.mycroftai with ID mycroft-timer.mycroftai
07:48:20.581 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-timer.mycroftai
07:48:20.888 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-alarm.mycroftai with ID mycroft-alarm.mycroftai
07:48:24.882 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-alarm.mycroftai
07:48:25.030 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-wiki.mycroftai with ID mycroft-wiki.mycroftai
07:48:25.445 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-wiki.mycroftai
07:48:25.519 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-configuration.mycroftai with ID mycroft-configuration.mycroftai
07:48:25.958 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-configuration.mycroftai
07:48:26.011 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-naptime.mycroftai with ID mycroft-naptime.mycroftai
07:48:26.219 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-naptime.mycroftai
07:48:26.263 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: fallback-unknown.mycroftai with ID fallback-unknown.mycroftai
07:48:26.452 - mycroft.skills.core:load_skill:145 - INFO - Loaded fallback-unknown.mycroftai
07:48:26.589 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: fallback-wolfram-alpha.mycroftai with ID fallback-wolfram-alpha.mycroftai
07:48:26.779 - mycroft.skills.core:load_skill:145 - INFO - Loaded fallback-wolfram-alpha.mycroftai
07:48:26.867 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-version-checker.mycroftai with ID mycroft-version-checker.mycroftai
07:48:27.013 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-version-checker.mycroftai
07:48:27.068 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-support-helper.mycroftai with ID mycroft-support-helper.mycroftai
07:48:27.108 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-support-helper.mycroftai
07:48:27.138 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-joke.mycroftai with ID mycroft-joke.mycroftai
07:48:27.330 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-joke.mycroftai
07:48:27.401 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-fallback-duck-duck-go.mycroftai with ID mycroft-fallback-duck-duck-go.mycroftai
07:48:27.506 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-fallback-duck-duck-go.mycroftai
07:48:27.533 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-spelling.mycroftai with ID mycroft-spelling.mycroftai
07:48:27.627 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-spelling.mycroftai
07:48:27.648 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: mycroft-speak.mycroftai with ID mycroft-speak.mycroftai
07:48:27.768 - mycroft.skills.core:load_skill:145 - INFO - Loaded mycroft-speak.mycroftai
07:48:27.806 - mycroft.skills.core:load_skill:117 - INFO - ATTEMPTING TO LOAD SKILL: Yeelight with ID Yeelight
07:48:27.965 - mycroft.skills.core:load_skill:145 - INFO - Loaded Yeelight
07:48:30.269 - mycroft.skills.padatious_service:train:75 - INFO - Training... (single_thread=False)
07:48:31.855 - mycroft.skills.padatious_service:train:77 - INFO - Training complete.
07:48:43.976 - mycroft.session:get:76 - INFO - New Session Start: 247e8c26-8fdb-43bd-9b07-730a69f58c05
07:48:44.001 - __main__:handle_record_begin:35 - INFO - Begin Recording...
07:48:44.020 - AlarmSkill - INFO - on started...
Playing WAVE '/home/pi/mycroft-core/mycroft/res/snd/start_listening.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
07:48:46.332 - __main__:handle_record_end:40 - INFO - End Recording...
07:48:46.349 - AlarmSkill - INFO - on ended...
07:48:46.348 - __main__:handle_wakeword:56 - INFO - Wakeword Detected: hey mycroft
07:48:48.610 - __main__:handle_utterance:61 - INFO - Utterance: ['light on']
07:48:48.760 - Yeelight - INFO - No room given
07:48:50.841 - Yeelight - INFO - Num Yeelights 1
07:48:50.842 - Yeelight - INFO - Yeelight ip 192.168.1.25
07:49:18.888 - AlarmSkill - INFO - on started...
07:49:18.872 - __main__:handle_record_begin:35 - INFO - Begin Recording...
Playing WAVE '/home/pi/mycroft-core/mycroft/res/snd/start_listening.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
07:49:22.530 - __main__:handle_record_end:40 - INFO - End Recording...
07:49:22.552 - __main__:handle_wakeword:56 - INFO - Wakeword Detected: hey mycroft
07:49:22.579 - AlarmSkill - INFO - on ended...
07:49:25.508 - __main__:handle_utterance:61 - INFO - Utterance: ['bedroom light off']
07:49:27.821 - ConfigurationSkill - INFO - Remote configuration updated
07:49:27.860 - Yeelight - INFO - Num Yeelights 1
  ^--- NEWEST ---^
History ==========================================================   Log Output Legend ============================================ Mic Level ===
 light on                                                             DEBUG output
 >> Turning on bedroom                                                skills.log, other
 bedroom light off                                                    voice.log
 >> Turning off bedroom
                                                                                                                                   --- 3.48

Input (':' for command, Ctrl+C to quit) ====================================================================================== 2
> _
```

Figure 2. Screenshot of Mycroft in debug mode—this shows the skills loaded at the top and the interactions with the user at the bottom.

Running Mycroft

Once booted, you can run Mycroft from the directory mycroft-core using the `start-mycroft.sh` command. This can take parameters, and the most voluble way of seeing what's going on is to run it in debug mode:

```
start-mycroft.sh debug
```

This brings up the curses screen shown in Figure 2.

Building Your Own Skills

Mycroft has an ever-growing list of skills that come by default. These include Pandora, Spotify, OpenHAB and Wikipedia. But there's always room for more! I've got some smart bulbs in my house, and I'd like to control them using Mycroft.

Adding a skill means you need to know how to talk to the service you're requesting. Often this information isn't publicly available and has to be reverse-engineered. Such is the case with the Lanbon light switches, for example, but that's another story! One device that is okay is the Yeelight smart light. It has the advantage of having a documented "local" API to control it, so there's no need to tell the Yeelight server what time you go to bed, for example. The Hue and LIFX lights offer similar privacy-respecting APIs.

The Mycroft skills are in the /opt/mycroft/skills directory. There is one directory per skill, and the following sub-directories and files:

- A *vocab* directory per language, such as vocab/en-us for US English phrases that will trigger an intent.

- A *dialog* directory per language, such as dialog/de-de for German language responses.

- The Python file __init__.py with a class for the skill and methods for each of the intents.

- Some additional files for non-basic features.

The __init__.py file is where all the work of the skill is done. It needs to import required classes and define a class that inherits from `MycroftSkill`. This class should have methods for each intent, which is triggered by the keywords for that intent. This was formerly signaled by code in the constructor, but the preference now is for an adornment on each method. The method then does what is needed, and finally calls for a response to be sent.

The Yeelight Bulb Skill

I use the Yeelight bulbs here, as they are relatively simple, documented, and it's easy enough to show how to build a small skill—note that there are additional expectations of a real skill that would be distributed by Mycroft! Mycroft has a more detailed set of instructions at Developing a new Skill.

Yeelight makes a number of smart bulbs that you can control through an Android or iPhone app. Like many home IoT devices, it communicates with a (Yeelight) server, and to set it up, you have to go through the usual complex registration processes for smart-home devices. But, it also has a REST-like API through which you can control it with appropriate network calls. Typically, these would be restricted to the home LAN unless ports are explicitly opened on the home router (bad idea!). But if you just want to control the lights

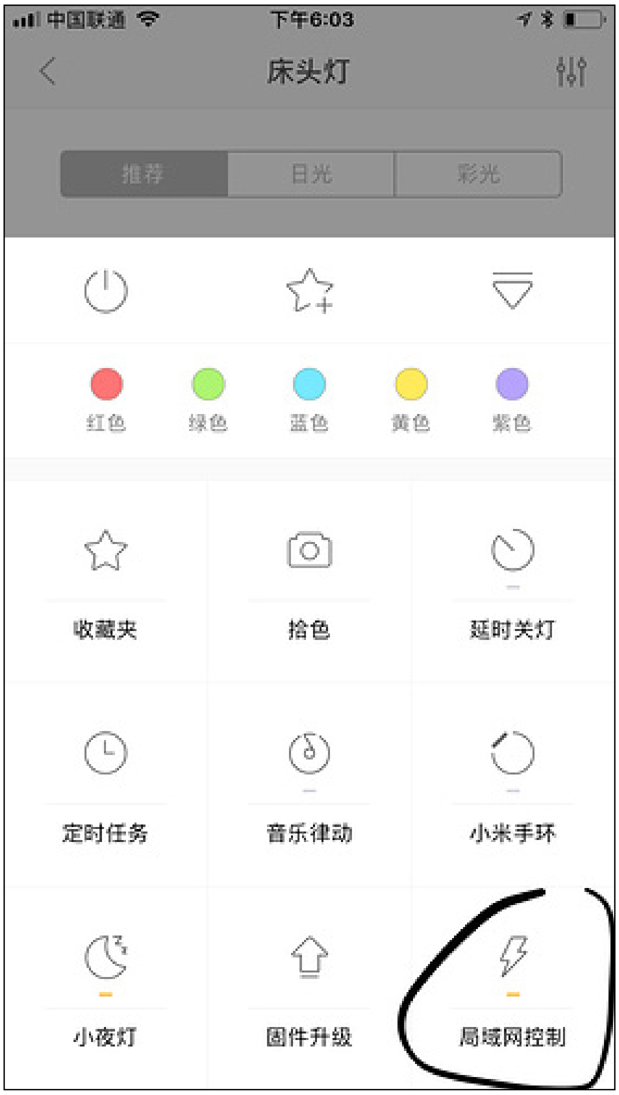


Figure 3. Enabling LAN API of the Yeelight

from inside the house, this is a good way. The "Yeelight WiFi Light Inter-Operation Specification" is described here.

The default is for LAN control to be disabled for each bulb. You have to enable the "Control LAN" mode, which is found by selecting a Yeelight device, scrolling down to the bottom of the screen and choosing the rightmost icon and then the Control LAN icon. The rather crude screenshot from Yeelight in Figure 3 shows this.

There are a number of different Yeelight bulbs. I look only at the common feature of on/off for each bulb in this article. The Mycroft files for such a skill are as follows:

- Yeelight/vocab/en-us/OnKeywords.voc
- Yeelight/vocab/en-us/OffKeywords.voc
- Yeelight/dialog/en-us/on.activity.dialog
- Yeelight/dialog/en-us/off.activity.dialog
- Yeelight/__init__.py
- Yeelight/requirements.txt

After enabling LAN control, the Yeelight bulb listens to multicast messages to 239.255.255.250 on port 1982. On receiving the correct message, each bulb responds with a packet containing information such as "Location: yeelight://192.168.1.25:55443". Several libraries have been developed to do this and other Yeelight commands. I use the library by Stavros Korokithakis available here. This can be installed by `pip`, but Mycroft will do it for you: the file requirements.txt contains a list of the packages needed to run a skill, which will be loaded by Mycroft, so here will contain the name of the Yeelight package:

```
yeelight
```

The package contains a method `discover_bulbs()`, which sends out the multicast message and collects the responses as a list, each element containing the IP address and port, and a list of the light's capabilities. A new `Bulb()` can then be created using the IP address. The simplest logic finds a list of bulbs and returns the first in the list, along with its name:

```
def select_light(location):
    yeelights = yeelight.discover_bulbs()
    if len(yeelights) >= 1:
        light_info = yeelights[0]
        ip = light_info['ip']
        name = light_info['capabilities']['name']
        return light, name
    return None, None
```

Once a light is found, commands can be sent to it. A command is in the form of a JSON string, which is constructed representing the command and its parameters and sent as a TCP packet to the bulb. These are encapsulated by the Yeelight package by methods including `turn_on()` and `turn_off()`.

Finally, you can give the "on" command, which combines the decorator to register the intent, the command itself and the response. The response contains the light's name, if it is available, to show that parameters can be passed to the speech dialog:

```
@intent_handler(IntentBuilder("OnIntent").\
                            require("OnKeywords").build())
def on_activity_intent(self, message):
    """Turn on Yeelight
    """
    light, name = select_light(location)
    if light != None:
        light.turn_on()
        if name == None:
            report = {"location": "unknown"}
        else:
            report = {"location": name}

        self.speak_dialog("on.activity", report)
```

The (US English) intent matcher for switching the light on is in the file Yeelight/vocab/en-us/OnKeywords.voc:

```
light on
```

And the (US English) response is in the file Yeelight/dialog/en-us/on.activity.dialog:

```
turning on {{location}}
```

If the light's name is, say, "bedroom", Mycroft will say "turning on bedroom". There is considerably more complexity that can be built in to the intent matcher and responses, these are just indicative.

Other Home Devices

Just considering lighting, many options exist now. Let's look at a few of them.

LIFX An API for both LAN and WAN management of devices is documented. The LAN interface uses UDP messages only, and the different devices are distinguished by their MAC address rather than IP address. The WAN API sends HTTP REST commands via the LIFX server. Several Python packages implement this, such as lifxlan for LAN control and pifx for WAN control. There is a Mycroft skill using the WAN API.

Hue Hue lights do not live on the IP network, but on a Zigbee network. The interaction with the LAN is via a Hue bridge: you talk IP to the bridge, it talks Zigbee to the devices. A REST API allows you to send requests to a light by a request to the bridge including the light's ID. Many Python packages implement this, and there already is a Mycroft skill for these lights.

IKEA Trådfr The IKEA lights also use Zigbee via a gateway. There is an API using CoAP over UDP, and it has been implemented in a Python package called pytradfri. Although there is no Mycroft skill for these lights, there is an OpenHAB binding and a Mycroft skill for OpenHAB.

Eufy Lumos This is an IP bulb, but the API does not appear to have been published. There have been several attempts to reverse-engineer the API, and there is currently a minimal Python package lakeside.

Note: although the functions performed by each smart light are similar, there is little consistency in their networking APIs, and there are a variety of Python and other language implementations. Unfortunately, many vendors have not published their APIs. This is reminiscent of the early days of device drivers for Linux, where some vendors were good, and others stupidly thought themselves clever for hiding specifications. A vendor of one product has told me they will give me the API if I purchase 2,000 units at about $100 each—not on my salary!

Security

There are multiple security issues to be considered in any IoT system, and several of them surface here.

First, there are security issues arising from switching the bulb to local access mode: it means that *anyone* on your local network (such as the disgruntled teenager you've just sent off early to bed) can program your lights to flash on and off just as you get to sleep. In the case of the Yeelight, that's a choice you get to make, as you have to change its mode explicitly. I'm assuming, of course, that your wireless network has security, such as WPA2 turned on, and any default router passwords have been changed—otherwise, you are vulnerable to any external attack!

You can reduce the security risks by running these lights on their own subnet, even on a different SSID with an appropriate firewall. This increases network configuration complexity as well as requiring a router that will pass multicast searches from the Mycroft subnet to the light's subnet. The IETF expects home networks to have multiple IP subnets in the future (at RFC 7368: IPv6 Home Networking Architecture Principles), so such issues will need to be addressed in user-friendly ways eventually.

Of course, you could say that this is all irrelevant, since anyone can just ask Mycroft to turn the lights on or off. Physical presence now clashes with network security! The Mycroft wake word recognizer cannot yet distinguish between different voices, but that's in the works.

What's Next?

Mycroft will continue to support the open-source version, and it's building up the skill sets. The company also makes its own hardware version, and version 2 will be released soon. This will have its own FPGA, which will allow more AI-style processing to be done onboard. So Linux hackers can play around to their hearts'

Resources

- Abraham Kang, "Understanding the Differences Between Alexa, API.ai, WIT.ai, and LUIS/Cortana"
- Fox News "Google is recording everything you say to a bot—right now"
- Steve Penrod, "Why We're moving to DeepSpeech on March 31 | Privacy, Speech to Text & Balance"
- Mycroft.ai, Mimic
- Mycroft.ai, Mycroft AI Open Source Voice Assistant
- Google AIY Voice Kit

content, and privacy-concerned individuals and companies can purchase an off-the-shelf version.

The main downside is the lack of specifications and stability for services and IoT devices. This is shown by Logitech in December 2018 turning off an undocumented LAN mode for the Harmony Hub due to security issues with its XMPP code. It caused an outcry from those relying on it for their home automation systems, and Logitech is now reconsidering its position—at least the company is listening; others are not! ■

Disclaimer: the author recently purchased shares in Mycroft.

Jan Newmarch has written several articles for the *Linux Journal* in the past, as well as more than 80 papers and six books. He is Professor of IoT at Box Hill Institute, and also Adjunct Professor at the University of Canberra and Adjunct Lecturer at Charles Sturt University. He expects to retire soon to a life of leisure, hacking and, of course, more Linux. His LinkedIn handle is https://www.linkedin.com/in/jannewmarch.

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

Oracle Linux on Btrfs for the Raspberry Pi

Enterprise comes to the micro server.

By Charles Fisher

Oracle Linux 7 has been released for the Raspberry Pi 3. The release packages Btrfs as the root filesystem on the UEK-branded Linux 4.14 Long Term Support

(LTS) kernel. A bootable disk image with a minimal install is provided along with a standard ISO installer.

CentOS appears to support only the "Mustang" Applied Micro X-Gene for AArch64, and it provides the older AArch32 environment for all models of the Raspberry Pi. Oracle Linux is a compelling option among RPM distributions in supporting AArch64 for the Pi Model 3.

This is not to say that Oracle AArch64 Linux is without flaw, as Oracle warns that this is "a preview release and for development purposes only; Oracle suggests these not be used in production." The non-functional WiFi device is missing firmware and documentation, which Oracle admits was overlooked. No X11 graphics are included in the image, although you can install them. The eponymous database client (and server) are absent. Oracle has provided a previous example of orphaned software with its Linux for SPARC project, which was abandoned after two minor releases. There's no guarantee that this ARM version will not suffer the same fate, although Oracle has responded that "our eventual target is server class platforms". One possible hardware target is the Fujitsu A64FX, a new server processor that bundles 48 addressable AArch64 cores and 32GB of RAM on one die, asserted to be the "fastest server processor" that exists.

AArch64 on the Pi

You'll need a Raspberry Pi Model 3 to run Oracle Linux. The 3B+ is the best available device, and you should choose that over the predecessor Model 3B and all other previous models. Both Model 3 boards retain the (constraining) 1GB of RAM—a SODIMM socket would be far more practical. The newer board has a CPU that is 200MHz faster and a Gigabit-compatible Ethernet port (that is limited to 300Mbit due to the USB2 linkage that connects it). A Model A also exists, but it lacks many of the ports on the 3B. More important, the Model 3 platform introduces a 64-bit CPU.

ARM was very tardy to 64-bit addressing capabilities compared to other well known microprocessor families, announcing this extension in 2011. Intel's first attempt to migrate the x86 market to the abortive Itanium 64-bit architecture shipped in 2001, ultimately yielding to AMD64, which debuted in 2003. MIPS and SPARC made this transition much, much earlier (1991 and 1995, respectively).

Despite the late arrival, the ARM AArch64 CPU architecture is now the dominant mobile platform. All supported iPhones are now required to run it, and most modern Android devices have migrated to it. ARM has guarded a competitive edge in power efficiency, as vast changes in its instruction set ideology and implementation have allowed ARM to maintain its market leadership in mobile.

The Acorn/Advanced RISC Machine (ARM) began with the retroactively named AArch32 assembly and machine language that was designed by Furber and Wilson for the Archimedes desktop computer, where tremendous power efficiency "was a complete accident". Desktop performance remained an architectural focus for a decade after the birth of ARM.

Apple's decision to base the (failed) Newton on ARM opened a new market of mobile device applications that prompted a new instruction set for ultra-compact assembly—Thumb 1 and 2. These are distinct from AArch32 and AArch64, and they focus on code density and minimal footprint for mobile devices. Thumb 2 is a 16-/32-bit variable length instruction set, which is dynamically translated to AArch32. Many other ARM extensions exist, but Thumb appears to have both great flexibility and persistence across ARM implementations.

When mobile devices neared the 4Gb RAM limit of 32-bit ARM, the designers decided to break from the past. The primary mistakes in AArch32 have long been known:

> Design errors, like having r15 as the program counter or making every instruction conditional, are problems for CPU architects rather than programmers, and it's no surprise that they disappeared in the 64-bit version of the ARM architecture. They must have made it awfully hard to implement superscalar, out-of-order execution.

The changes in AArch64 bring it much closer to the spirit of MIPS, the most notable of which are:

- Conditional Execution has been removed, facilitating out-of-order processing in multiple pipelines.

- The R15/PC register can now be manipulated only by a small number of jumping and branching instructions, vastly simplifying branch prediction.

These performance improvements, along with the increase of pointer sizes to 64-bits, come at the cost of code density—programs compiled for native AArch64 will be larger than the equivalent for AArch32. Despite these enhancements, the majority of Intel desktop processors of the last decade easily will beat the Pi in most benchmarks (but they will not do so with a 10-Watt power supply). I examine the code density impact of AArch64 in greater detail below.

Installation

Most Raspberry Pi users rely on flash memory, which comes in two grades. Multi-Level Cell (MLC) media is cheap and offers large amounts of storage, but it can decay very quickly (cells typically are destroyed after 5,000 write operations). Most retail flash media (SD cards, USB flash drives) are MLC, and will not have a long lifetime under high I/O usage, despite the "wear-leveling" electronics that attempt to distribute writes over the whole device evenly. Single-Level Cell (SLC) media is more expensive and offers smaller amounts of storage, but it drastically increases the number of write operations until cell failure (100,000). Both types of memory can be "rehabilitated" by heating them, but this is not feasible for memory in most plastic packaging. If you anticipate large amounts of I/O, plan to purchase the correct grade of flash.

One great benefit of the new Model 3 is the ability to boot from USB. A standard hard disk drive is now a boot option. SLC media is also more plentiful and inexpensive in the USB flash format than as microSD cards. Choose the format that fits your expected I/O usage.

Once you have selected and obtained your media, you're ready to download and uncompress the following file:

```
$ xz -dkv rpi3-ol7.6-image-20181116.img.xz
rpi3-ol7.6-image-20181116.img.xz (1/1)
  100 %   266.4 MiB / 5120.0 MiB = 0.052  55 MiB/s   1:32
```

The full size of the boot image is 5GB—your boot media must be at least this size:

```
$ ll rpi3-ol7.6-image-20181116.img*
-rw-r--r-- 1 root root 5368709120 Jan 13 18:52
rpi3-ol7.6-image-20181116.img
-rw-r--r-- 1 root root  279309592 Jan 13 18:52
rpi3-ol7.6-image-20181116.img.xz
```

Insert your boot media and ensure that it is detected, but not mounted:

```
# dmesg | tail -3
[  378.540649] mmc0: new high speed SDHC card at address 0002
[  378.544104] mmcblk0: mmc0:0002 00000 7.83 GiB
[  378.548395]  mmcblk0: p1
```

Finally, write the image to the raw media:

```
# dd if=rpi3-ol7.6-image-20181116.img of=/dev/mmcblk0 bs=4M
```

Assuming there are no write errors, the media is now prepared to boot the Raspberry Pi Model 3.

Operation

Load the media into the Pi, connect an HDMI cable to a monitor, and attach an ethernet cable. After the power supply is connected, the Pi will boot (there is no power switch).

The Pi might fail to boot with older USB peripherals. When a well-worn IBM 89P8800 USB 1.1 keyboard was attached, the firmware issued the message `Timeout poll on interrupt endpoint` and refused to boot. Trying a slightly newer Lenovo 41A5248 keyboard, the boot proceeded but was greatly delayed. Both keyboards worked without error once the OS was running. It might be wise to boot initially without any non-essential USB hardware.

Once the boot is complete, a `login:` prompt should be displayed. The user `root` with the password `oracle` will allow you to select a new root password, then drop to Bash.

The /proc/cpuinfo file will list four processor cores with the following descriptions:

```
[root@rpi3 ~]# cat /proc/cpuinfo
processor   : 0 ... 1 ... 2 ... 3
BogoMIPS    : 38.40
Features    : fp asimd evtstrm crc32 cpuid
CPU implementer   : 0x41
CPU architecture: 8
CPU variant : 0x0
CPU part    : 0xd03
CPU revision      : 4
```

The root filesystem is on Btrfs, which is a change from the XFS that you normally see on the x86_64 (AMD64) version of Oracle/Red Hat/CentOS/Scientific Linux 7. The /boot filesystem is on EXT4, likely due to bootloader considerations:

```
[root@rpi3 ~]# mount | egrep 'btrfs|ext4'
/dev/mmcblk0p4 on / type btrfs
(rw,noatime,ssd,space_cache,subvolid=5,subvol=/)
/dev/mmcblk0p2 on /boot type ext4 (rw,noatime,data=ordered)
```

Observe also the `ssd` mount option above. Btrfs detected this option automatically and was previously unsafe with a "a negative impact on usability and lifetime" of the flash media, but it's now appropriate in the 4.14 kernel. Observe that the autodetected SSD mount option was not specified in /etc/fstab (I've removed the UUIDs and LABELs):

```
[root@rpi3 ~]# sed -r 's/^(UUID|LABEL)[^ ]*[ ]*//' /etc/fstab
#Generated by RootFS Build Factory
/boot/efi vfat noatime 0 0
/boot ext4 noatime 0 0
swap swap noatime 0 0
/ btrfs noatime 0 0
tmpfs   /tmp       tmpfs   rw,nodev,nosuid,size=128M    0 0
```

The root filesystem is on the p4 partition of the SD card:

```
[root@rpi3 ~]# fdisk -l

Disk /dev/mmcblk0: 7948 MB, 7948206080 bytes, 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000164f6

        Device Boot    Start      End   Blocks   Id  System
/dev/mmcblk0p1          2048   526335   262144    c  W95 FAT32
 ↪(LBA)
/dev/mmcblk0p2        526336  1550335   512000   83  Linux
/dev/mmcblk0p3       1550336  2074623   262144   82  Linux
 ↪swap / Solaris
/dev/mmcblk0p4       2074624  10463231  4194304   83  Linux
```

Note above that I'm running on an 8GB SD card, but the last third of the card is unused, as it does not lie in a partition. You can add the unused space to the root filesystem by first expanding the partition:

```
[root@rpi3 ~]# growpart /dev/mmcblk0 4
CHANGED: partition=4 start=2074624 old: size=8388608
↪end=10463232 new: size=13449183,end=15523807
```

And then expanding the Btrfs filesystem into the newly allocated space:

```
[root@rpi3 ~]# btrfs filesystem resize max /
Resize '/' of 'max'
```

The root filesystem now occupies the rest of the flash device:

```
[root@rpi3 ~]# fdisk -l

Disk /dev/mmcblk0: 7948 MB, 7948206080 bytes, 15523840 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000164f6

        Device Boot  Start      End   Blocks   Id  System
/dev/mmcblk0p1        2048   526335   262144    c  W95 FAT32
 ↪(LBA)
/dev/mmcblk0p2      526336  1550335   512000   83  Linux
/dev/mmcblk0p3     1550336  2074623   262144   82  Linux
 ↪swap / Solaris
/dev/mmcblk0p4     2074624 15523806  6724591+  83  Linux
```

Btrfs is an extremely powerful filesystem, similar in capabilities to ZFS. It's capable of transparent compression, mirroring, error detection and it has self-healing capabilities. (Watch for a future article on in-depth Btrfs coverage.) Oracle Linux on the Raspberry Pi provides a useful learning environment for many new tools and features, and the addition of Btrfs is chief among them.

A number of missing utilities are present in minimal installs on x86_64. In no particular order of preference, some are `ethtool`, `less`, `man` and `nmtui`. Assuming internet connectivity to Oracle, a `yum install man` will bring in `less` as a dependency (and let you begin reading all the Btrfs manual pages). The `yum whatprovides` command is useful for searching the contents of uninstalled packages for a particular utility.

Busybox is an alternative to some of the native Oracle AArch64 packages. Those unfamiliar with Busybox might review my previous container article published in *Linux Journal* that details its use. The 1.28.1 release offers several ARM binaries of Busybox (listed below):

```
[root@rpi3 ~]# for x in busybox-arm*
               do ls -l $x; file $x; ./$x | head -1; done

-rwxr-xr-x 1 root root 1132724 Jan 10 17:23 busybox-armv5l
busybox-armv5l: ELF 32-bit LSB executable, ARM, version 1
 ↪(SYSV)...
BusyBox v1.28.1 (2018-02-15 14:34:02 CET) multi-call binary.
-rwxr-xr-x 1 root root  836560 Jan 10 17:23 busybox-armv7m
busybox-armv7m: ELF 32-bit LSB shared object, ARM, version 1
 ↪(SYSV)...
BusyBox v1.28.1 (2018-02-15 14:34:02 CET) multi-call binary.
-rwxr-xr-x 1 root root 1079156 Jan 10 17:23 busybox-armv7r
busybox-armv7r: ELF 32-bit LSB executable, ARM, version 1
 ↪(SYSV)...
BusyBox v1.28.1 (2018-02-15 14:34:02 CET) multi-call binary.
-rwxr-xr-x 1 root root 1078504 Jan 10 17:23 busybox-armv8l
busybox-armv8l: ELF 32-bit LSB executable, ARM, version 1
 ↪(SYSV)...
BusyBox v1.28.1 (2018-02-15 14:34:02 CET) multi-call binary.
```

Notice above that the busybox-armv7m binary is substantially smaller than all the rest. This binary is apparently composed of Thumb 2 code; the ARM 7 M and R architectures appear to exclude both AArch32 and AArch64: "ARMv7-M... No ARM instruction set support (Thumb only)." Thumb might explain the smaller size, but its use may come at some cost to performance.

Oracle includes an AArch64 compiler environment, but this isn't likely to be capable of emitting ARMv7-M code. Oracle doesn't provide a glibc.aarch32 or glibc.thumb2 package in the same way that it provides a glibc.i686 on AMD64, nor are there any 32-bit libraries in /usr/lib. ARM itself provides Thumb-capable GNU compilers, as do other sources. Using Thumb as a compiler target will conserve memory at the potential cost of performance. This might be a reasonable choice for standing dæmons that are not CPU-intensive.

One glaring lack in Oracle Linux on the Raspberry Pi is the missing WiFi device. The kernel dmesg has a clue to the problem:

```
brcmfmac: brcmf_fw_map_chip_to_name: using
        brcm/brcmfmac43455-sdio.bin for chip 0x004345(17221)
         ↪rev 0x000006
usbcore: registered new interface driver brcmfmac
brcmfmac mmc1:0001:1: Direct firmware load for
                         brcm/brcmfmac43455-sdio.bin failed
                          ↪with error -2
```

You can find one source of the missing firmware at this link, although you also can find it within Raspbian. Installing the firmware will cause a wlan0 device to appear, but all my attempts to configure it have failed. It doesn't appear to be functional in the current release, despite the brcmfmac kernel module:

```
[root@rpi3 ~]# cd /usr/lib/firmware/brcm/
[root@rpi3 ~]# ll brcmfmac43455*
```

```
-rw-r--r-- 1 root root 600487 Jan 14 19:33
 ↪brcmfmac43455-sdio.bin
-rw-r--r-- 1 root root  14036 Jan 14 19:49
 ↪brcmfmac43455-sdio.clm_blob
-rw-r--r-- 1 root root   2054 Jan 14 19:41
 ↪brcmfmac43455-sdio.txt
```

It appears that the WiFi and Bluetooth devices on the Raspberry Pi work through the SD card's SDIO interface. Once those files are in place, reboot, and the WiFi driver should appear in the `dmesg`. Note this will use a small amount of additional memory:

```
mmc1: new high speed SDIO card at address 0001
brcmfmac: brcmf_fw_map_chip_to_name: using
        brcm/brcmfmac43455-sdio.bin for chip 0x004345(17221)
         ↪rev 0x000006
brcmfmac: brcmf_c_preinit_dcmds: Firmware version = wl0:
      Feb 27 2018 03:15:32 version 7.45.154 (r684107 CY)
       ↪FWID 01-4fbe0b04
Bluetooth: Generic Bluetooth SDIO driver ver 0.1
```

There is no mention at all of the WiFi hardware on the Raspberry Pi within Oracle's documentation on the AArch64 release, which Oracle claims was an oversight. This also appears to be an issue in CentOS, where it is at least discussed at some length.

Code Density

As 1GB of RAM included on the Pi is constraining, you should have some idea of the penalty AArch64 imposes.

Below is a script that I've used to size all the ELF binaries in Raspbian Linux running on the original Raspberry Pi, storing this in the file a32.txt:

```
for x in /bin/*
do [ -f "$x" ] &&
   case "$(file "$x")" in
        *ELF*) stat -c %n\ %s "$x";;
   esac
done > a32.txt
```

Moving this file to Oracle Linux running on the Raspberry Pi Model 3 B+, I run the following to find the size differences:

```
while read p s
do [ -f "$p" ] &&
   case "$(file "$p")" in
        *ELF*) echo $p $s $(stat -c %s "$p");;
   esac
done < a32.txt | awk '
    {a+=$2; b+=$3; print $1,$2,$3,$3/$2}
END {print a,b,b/a}' > a64.txt
```

For this small sample of 66 files, I found the results shown in Table 1.

Table 1. Results of Size Differences of 66 Files

Program	Raspbian	OL7.6	% increase
/bin/bash	912712	971728	1.06466
/bin/cat	30560	70408	2.30393
/bin/chgrp	51084	70944	1.38877
/bin/chmod	46956	70840	1.50865
/bin/chown	51092	71000	1.38965
/bin/cp	104592	204296	1.95327
/bin/cpio	118460	141752	1.19662
/bin/date	83868	70368	0.839033

/bin/dd	63424	136456	2.15149
/bin/df	67876	137848	2.03088
/bin/dir	108804	138240	1.27054
/bin/dmesg	59484	78296	1.31625
/bin/echo	26404	69904	2.64748
/bin/false	22304	69880	3.13307
/bin/findmnt	52144	71992	1.38064
/bin/grep	173656	204048	1.17501
/bin/gzip	80476	137400	1.70734
/bin/hostname	13964	69048	4.94471
/bin/journalctl	63204	538448	8.51921
/bin/kill	22020	70432	3.19855
/bin/kmod	128560	203960	1.5865
/bin/less	151392	219472	1.44969
/bin/lessecho	9688	68752	7.09661
/bin/lesskey	14460	70320	4.86307
/bin/ln	46976	70848	1.50817
/bin/login	39112	70032	1.79055
/bin/loginctl	42732	538280	12.5966
/bin/ls	108804	138240	1.27054
/bin/lsblk	67756	138336	2.04168
/bin/mkdir	63472	137080	2.15969
/bin/mknod	55248	71272	1.29004
/bin/mktemp	34668	70288	2.02746
/bin/more	34708	69824	2.01176
/bin/mount	34872	68840	1.97408
/bin/mountpoint	9896	68944	6.96686
/bin/mv	100504	138480	1.37786
/bin/netstat	106676	211912	1.9865
/bin/ping	55720	70208	1.26001
/bin/ps	83624	137000	1.63829
/bin/pwd	26452	70056	2.64842

/bin/readlink	34628	70448	2.03442
/bin/rm	51076	71056	1.39118
/bin/rmdir	34628	70072	2.02356
/bin/sed	84100	71904	0.854982
/bin/sleep	26416	69984	2.6493
/bin/stty	59240	70240	1.18569
/bin/su	31016	69008	2.22492
/bin/sync	26424	69912	2.64578
/bin/systemctl	161680	738032	4.56477
/bin/systemd-ask-password	9948	70000	7.03659
/bin/systemd-escape	9936	69816	7.02657
/bin/systemd-hwdb	67520	136520	2.02192
/bin/systemd-inhibit	14040	337728	24.0547
/bin/systemd-machine-id-setup	18128	69912	3.85658
/bin/systemd-notify	9936	69728	7.01771
/bin/systemd-tmpfiles	50988	202912	3.9796
/bin/systemd-tty-ask-password-agent	26324	135920	5.16335
/bin/tailf	22288	69488	3.11773
/bin/tar	327644	350288	1.06911
/bin/touch	71584	70640	0.986813
/bin/true	22304	69880	3.13307
/bin/udevadm	395336	469248	1.18696
/bin/umount	22436	68856	3.069
/bin/uname	26416	69928	2.64718
/bin/vdir	108804	138240	1.27054
/bin/wdctl	26408	70256	2.66041
	5107652	9795488	1.91781

These programs take nearly twice the space in Oracle Linux as they do in Raspbian. This somewhat explains the CentOS decision to remain on AArch32 with smaller 32-bit binaries. Oracle's pursuit of AArch64 is likely due to similar platforms that it supports or may support in the future.

Should Oracle elect to provide a Thumb2 development environment in the same way that it supports 32-bit x86, then Oracle could produce even smaller binaries than are found in Raspbian while still running a 64-bit kernel, at some cost to performance. This assumes that all target platforms support Thumb2; by all appearances, the Fujitsu A64FX does not.

It might be useful to examine commonly run server dæmons, system libraries and extract the text/data/bss segment sizes within all of these programs to see greater detail on the AArch64 penalty paid here. Those with large ARM deployments are encouraged to do so.

Conclusion

It's refreshing to have a new Linux distribution where legacy support is slashed in a way that never would be tolerated within Intel/AMD64 environments. There is substantial complexity and inertia in the maintenance of the systems of decades past.

Still, the relative silence in the documentation on questions of (the overlooked WiFi) hardware support and the legacy Thumb and AArch32 instruction sets is unsettling. Operating system vendors should be clear on what their products can and cannot do with the target hardware. While there are issues with Oracle AArch64 Linux where that clarity is lacking, it must be conceded that this is a pre-production release, and the desired clarity and supported server-grade AArch64 target platforms may not yet exist. To reiterate, one possible hardware target is the Fujitsu A64FX, which designers are asserting as the fastest processor in the world. Amazon also recently began running ARM workloads in its EC2 cloud with its Graviton processor, but Gravitons are not expected to outperform the Fujitsu A64FX, and the relationship between Amazon and Oracle is not warm. Oracle also may be developing its own AArch64 specifically tuned for Oracle's database. Oracle previously has maintained SPARC in this capacity, and continues to dominate the TPC benchmark with it; the company also may decide to do this with its own ARM processor.

On the subject of the Oracle Database, the absence of any discussion or mention of it also is a cause for substantial concern on the longevity of the platform.

In any case, Oracle AArch64 Linux likely will be used by many pursuing low-power, large memory applications. The Raspberry Pi may be able to provide a development environment for those larger systems. It's encouraging to see ARM move into the enterprise space, and the prospect of a legacy-free computing environment without all the problems (Meltdown), scandals (ME/PSP) and firmware concerns of Intel is quite refreshing. ■

Charles Fisher has an electrical engineering degree from the University of Iowa and works as a systems and database administrator for a Fortune 500 mining and manufacturing corporation.

Resources

- Oracle Linux for ARM Downloads
- Raspberry Pi (Wikipedia)
- Btrfs—the Next Generation Filesystem for Linux
- CentOS and AArch64
- Oracle Linux for Sparc Downloads
- "Fujitsu's A64FX ARM Chip Waves The HPC Banner High" by Timothy Prickett Morgan

- R4000 Microprocessor from 1991 (Wikipedia)
- UltraSPARC Microprocessor from 1995 (Wikipedia)
- "ARM Creators Sophie Wilson and Steve Furber, Part Two: the accidental chip" by Chris Bidmead, *The Register*
- Acorn Archimedes (Wikipedia)
- Apple Newton (Wikipedia)
- Thumb and Thumb-2 (Wikipedia)
- "RISC instruction sets I have known and disliked" by Jack Whitham
- Conditional Execution (Wikipedia)
- R15/PC register (Wikipedia)
- Types of Flash Memory (Wikipedia)
- "Making flash memory more reliable with 800°C heat pulses" by Mark Tyson
- Installation of Raspberry Pi 3 Image—Release Notes for Oracle Linux 7 Update 6 (aarch64)
- Btrfs Gotchas
- "ZFS for Linux" by Charles Fisher, *LJ*, March 2018
- BusyBox
- "Infinite BusyBox with systemd" by Charles Fisher, *LJ*, March 2015
- BusyBox 1.28.1 Release
- What are the differences between ARMv7-A, ARMv7-R and ARMv7-M? (arm Developer)
- GNU ARM Embedded Toolchain Downloads (arm Developer)
- armbian firmware

- SDIO cards (Wikipedia)
- CentOS on ARMv7hl boards
- The Raspberry Pi 3 B+ in Fedora (nullr0ute's blog)
- Fujitsu A64FX Post-K Supercomputer: World's Fastest ARM Processor (YouTube)
- "Amazon's homegrown 2.3GHz 64-bit Graviton processor was very nearly an AMD Arm CPU" by Chris Williams, *The Register*
- SPARC SuperCluster with T3-4 Servers

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.

PiBox: an Embedded Systems Journey

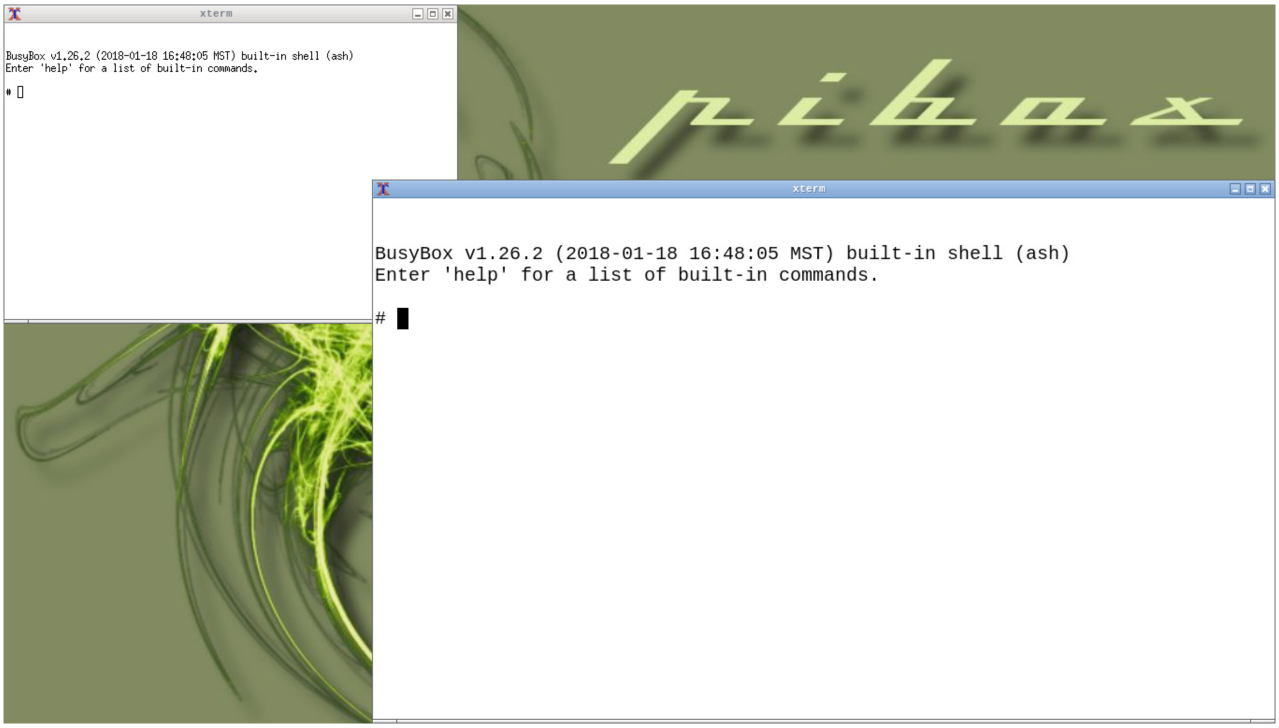
Raspberry Pi development with off-the-shelf software is easy by design. But, how would you use it to build a custom distribution with cross-compiled applications targeted for distributed media playback? Michael J. Hammel shares his experience in doing just that with his PiBox project.

By Michael J. Hammel

About 12 years ago, I started work on a system that was significantly different from the typical desktop/server systems I'd worked on previously. It was an embedded quad-core MIPS-based platform developed for general-purpose in-flight computational processing of radar data. That platform never saw the light of day, but it started me down a path that I'm still on today: building systems from scratch. And when I say *systems*, I mean the entire software stack from compiler toolchains through distributed end-user applications and into enclosures and production hardware.

That early work led to a project I started in early 2011 called BeagleBox. BeagleBox is a custom build system aimed at the BeagleBoard low-cost TI-based SoC board. The build system focus is on generating a cross-compiler toolchain that is used to compile bootloaders, the Linux kernel and an initial root filesystem. For a variety of non-technical reasons, the project switched in November 2012 to the Raspberry Pi, and the project is now known as PiBox.

PiBox is a generic name that currently encompasses three platforms. The PiBox Development Platform is the core distribution. It aims to be an easily extendable



```
BusyBox v1.26.2 (2018-01-18 16:48:05 MST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

```
BusyBox v1.26.2 (2018-01-18 16:48:05 MST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

Figure 1. The default UI for the PiBox developement platform is based on Blackbox.

platform for purpose-driven applications using opkg package management. The development platform boots into a standard X desktop using Blackbox. Beyond the UI and network configuration, there is little functionality. The development platform is just a starting point, but it easily can be extended using the PiBox build environment.

A collection of packages convert the Development Platform into a prototype consumer device called the PiBox Media Server. This device provides a simple UI for navigation to services such as video playback. However, the device also provides services to distributed PiBox Media Players. The Players are software-identical to the Server, but they are used specifically to play remote video sources. Long term, you'll see the Media Server provide a cental hub for home automation via the IronMan project and form the platform for a DIY phone based on the Pi Zero. Components of the the Media Server have been used to create a kiosk picture and video player running on a Pi Touchscreen.

There are three primary goals for PiBox:

1. Be a tool for learning not just the embedded development process but full systems development.

2. Be easily extensible for purpose-driven applications.

3. Be a prototype consumer device.

Ancillary goals now include learning how to design and produce consumer-grade enclosures, power management and other hardware-related issues, parts supply and costing and production issues.

PiBox met these goals with the current release, and the project continues to evolve by addressing what consumer devices really encompass. This includes support for intergration of mobile devices, voice command and control and distributed sensor networks. If you think of Linux as Linus' answer to freely available UNIX and Minix, then PiBox would be the answer to building distributed IoT infrastructure from the ground up.

Media Server and Media Player

The Media Server and Media Player are extensions of the Development Platform. The goal of the Media systems is to provide distributed media playback in a closed network, specifically in my family's travel trailer. We replaced the TV/DVD with the Media Server, plugging in USB sticks filled with videos. The server distributes the videos to mobile devices, such as our Android and iOS tablets, but also to a Media Player. The player holds a Raspberry Pi connected to a pico projector, all enclosed in a box with audio port exposed. We plug in an audio splitter to headphones, point the player at the side of our trailer and watch movies at night while camping.

The server, at the time of this writing, has limited support for Bluetooth devices, but it will eventually support an array of Bluetooth and BLE (Bluetouth low energy)

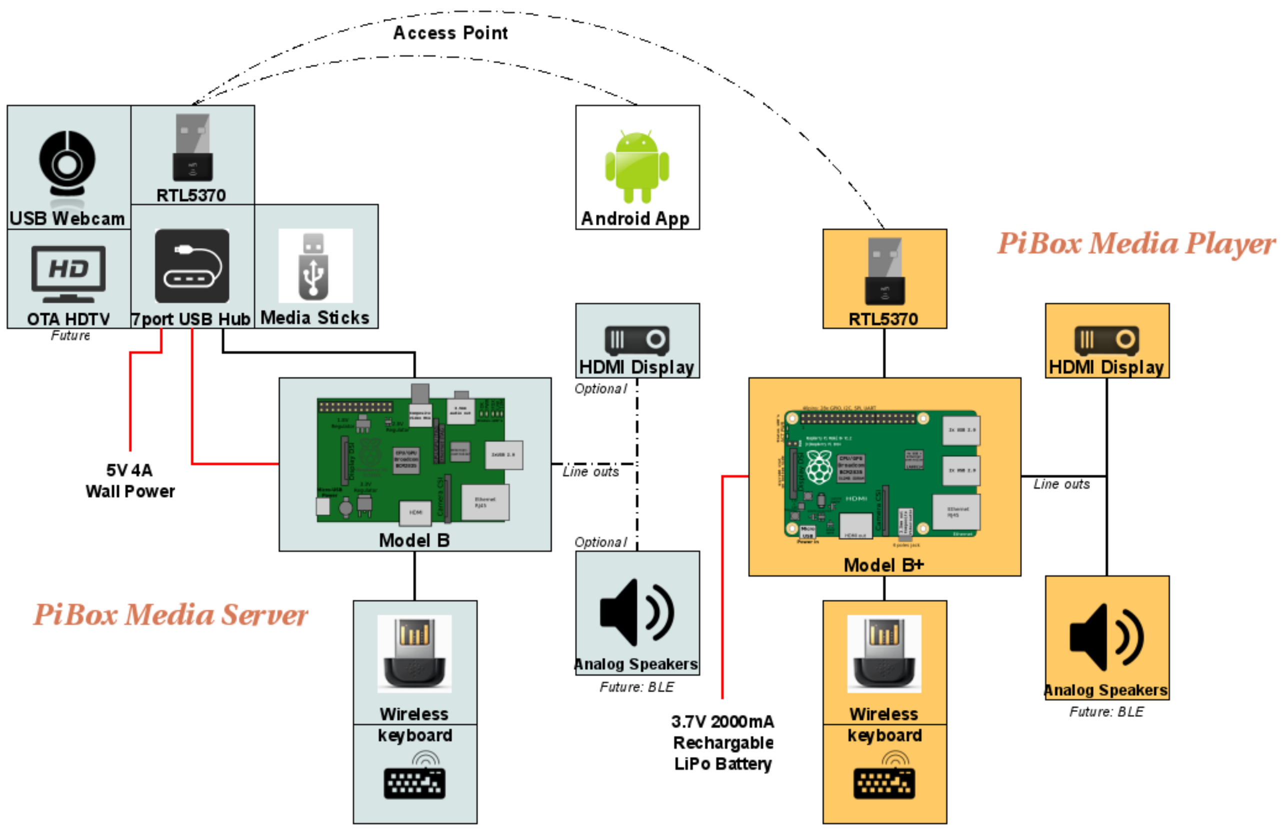


Figure 2. The Media Server and Media Player have specific hardware specifications.

devices for monitoring sensors. There are many other long term goals for the server, including voice control, remote management and better support for a wider array of video formats.

Hardware

The Media Player is a self-contained box intended to run on battery power only. A Raspberry Pi is connected via HDMI to a handheld projector, a power switch and a battery. The Media Server contains a Pi connected to a 7-port UUGear USB hub, each powered through a single power switch connected to an external power supply. The Server exposes both the HDMI and audio ports, but the HDMI defaults to audio over HDMI. The HDMI doesn't have to be connected for the server to function. The server USB ports are extended from the hub to the case to make them easy to access.

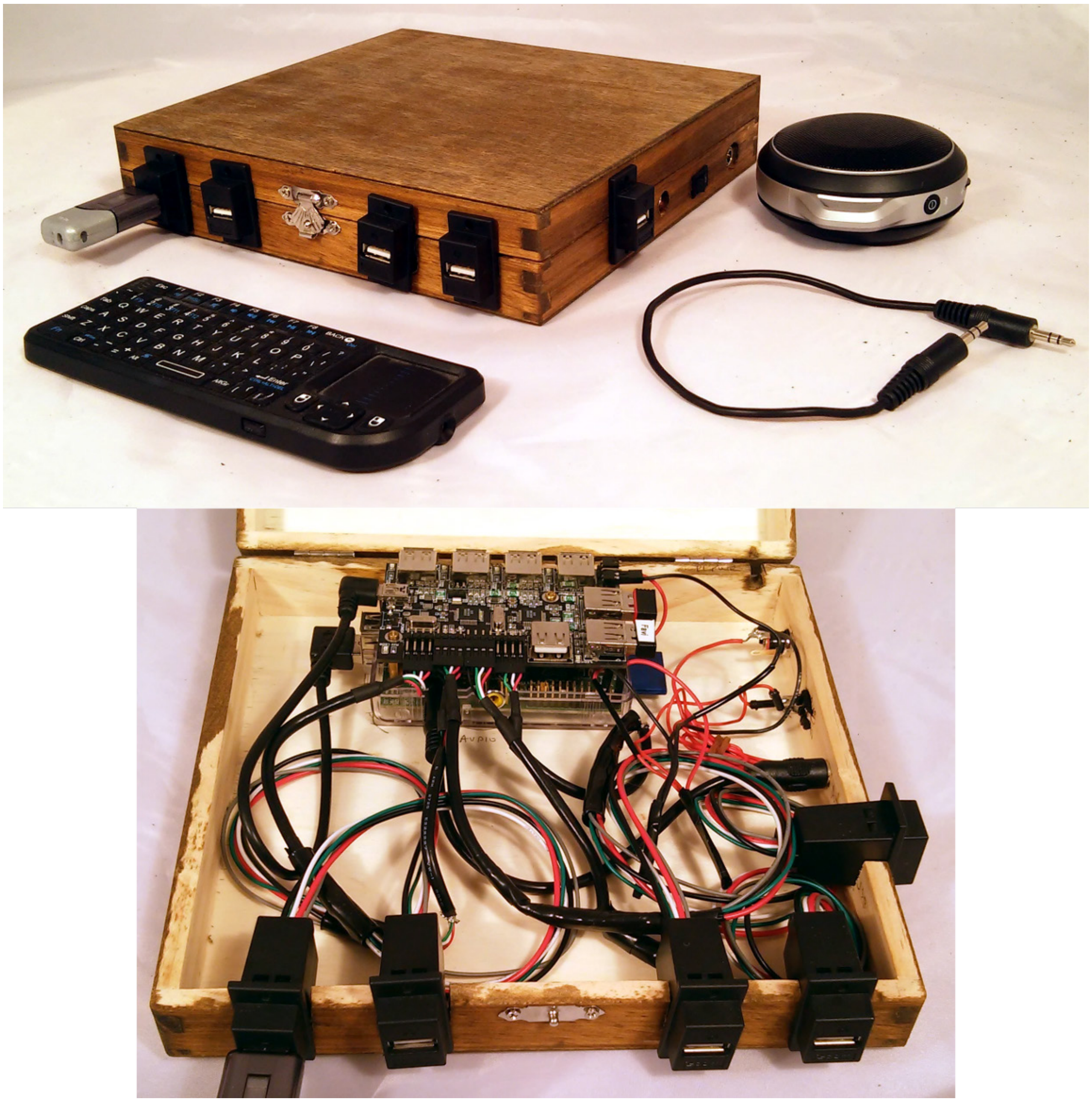


Figure 3. Media Server hardware includes enclosures with exposed USB ports.

The design for both Server and Player specifies a single power-on switch. The current Player prototype requires powering the Pi and the projector separately, since the projector is an off-the-shelf model that would require modification to make it integrate cleanly.

The Build System

At the root of PiBox is the core build system. There are many build systems for embedded systems. Many are based on bitbake, a Python-based metabuild system. Others embrace traditional GNU Make. Metabuilds are simply build systems that wrap other software builds. A metabuild system might include a toolchain, the Linux kernel and various pieces of a root filesystem.

PiBox is a metabuild system that wraps four primary components: a cross toolchain for a chosen hardware platform, a bootloader, the Linux kernel and a root filesystem. The design goal of the metabuild system is to permit easy modification of the primary components, extend the primary components or change them to use different tools and provide a consistent set of targets for building them.

The cross toolchain is built based on Crosstool-NG. For the Raspberry Pi, this includes the use of a custom glibc from Linaro, the ARM division devoted to improving software support for ARM devices. It also includes a custom GCC built with hard float support, allowing applications to take full advantage of the Broadcom processors they support.

PiBox wraps Buildroot and BusyBox to generate a root filesystem. A small number of custom packages are added, including a standalone network configuration tool (PiBox Network Config and its associated libpnc library). The root filesystem is designed to work with both wired Ethernet and WiFi out of the box. The core build system allows for easy rebuilds when changing the Buildroot configuration or testing Buildroot packages.

The design of the build system is flexible enough to support the use of a bootloader, such as Das U-Boot. However, the Raspberry Pi is used with a binary bootloader, also referred to in this case as firmware, provided through a GitHub repository. PiBox downloads and unpacks this firmware during the bootloader phase of the build. However, it easily could be modified to use a bootloader that required compilation.

The top-level Makefile includes files suffixed with .cfg and .mk from the config directory. The .cfg files provide configuration options that allow changing upstream repositories and build options without having to modify build targets in the .mk files. The goal of the .mk files is to provide a consistent interface, so that all components are built using the same set of targets: -get, -unpack, -patch, -config, -pkg and so forth. Most important, all core components of the build offer `-menuconfig` targets that allow changing the underlying tool configuration. All builds are done out of tree to keep build artifacts seperate from the build system. A `-saveconfig` target allows saving the changes to any component into the PiBox source tree. This makes it easier to commit to a local or upstream repository. A built-in help target explains all available component targets. Editing and saving a modified Buildroot configuration looks like this:

```
make buildroot-menuconfig
make buildroot-saveconfig
make buildroot-clobber
make buildroot
```

PiBox does more than just build each component. The cross toolchain is relocatable. It can be packaged as an RPM that installs under /opt/rpiTC. The target components (bootloader, kernel and root filesystem) also are packaged along with scripts for formatting and populating an SD card. The SD card is used to boot the Raspberry Pi.

The key advantage to the packaging is that the combination of the toolchain and a staging archive allow cross compiling of applications without having to build the core development platform. A project with multiple developers deploys the cross build environment to developer systems. Developers focus on app development instead of building the core development platform. Template build systems are available for autoconf projects and for downloading and compiling/packaging third-party applications to extend the core root filesystem.

Like many Raspberry Pi distributions, PiBox comes with a set of scripts to ease installation to an SD card. The public releases of the binary distribution includes these scripts and all files needed by end users wanting to try PiBox without compiling anything.

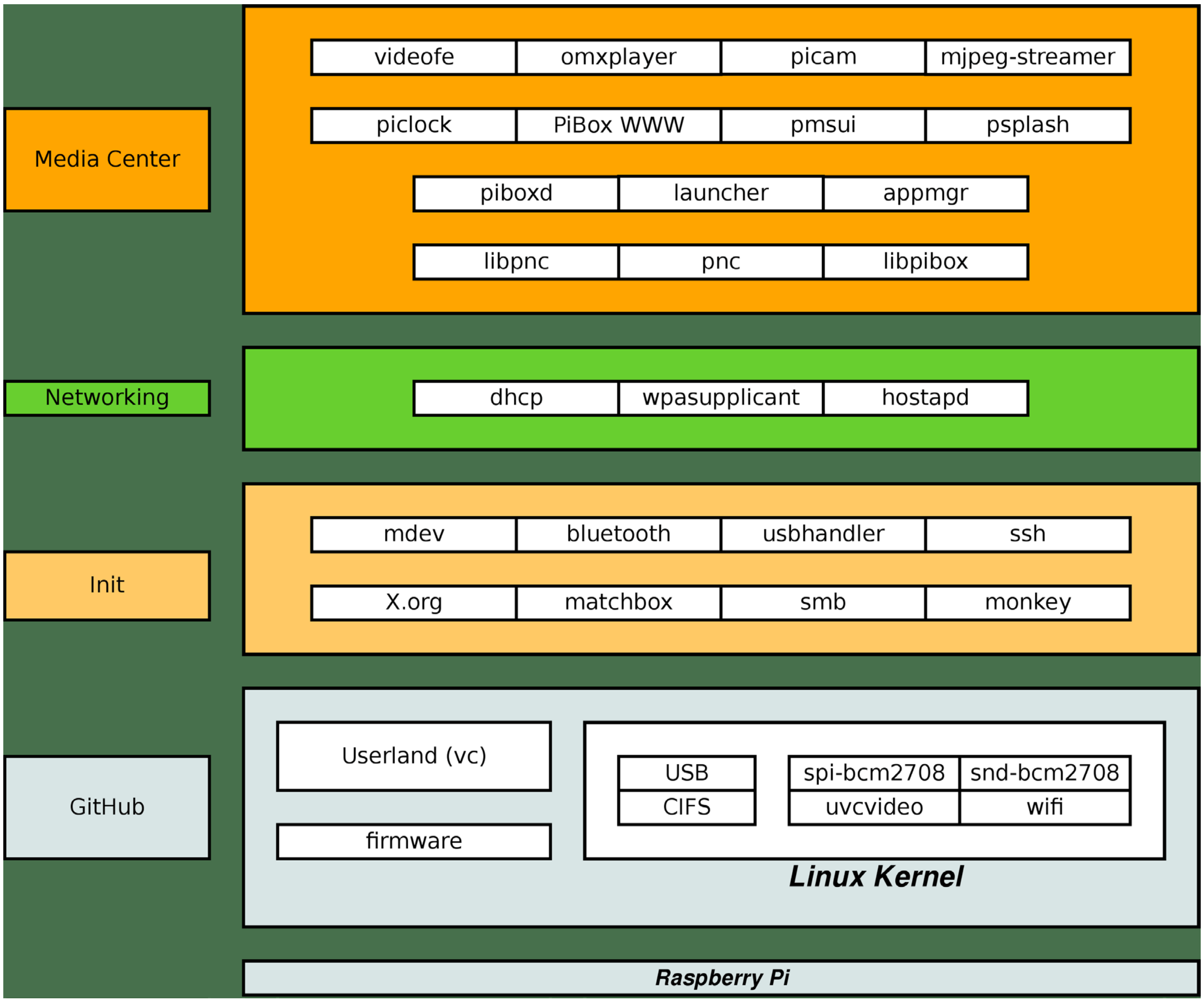


Figure 4. A small but growing set of library APIs was recently added to the software stack.

Software Stack

The PiBox Development Platform is a base on which purpose-driven systems can be built. For the PiBox Media Server and PiBox Media Player, I've developed a UI based on Matchbox, a stacked window manager running on X.org over the framebuffer. Apps are managed by a dæmon (`appmgr`) that handles communications between them and a back-end set of dæmons (`piboxd`, for example) for system services. A newly developed API layer (`libpibox` and `libpnc`) provides custom interfaces for handling common tasks, such as network configuration and media sharing.

The Media systems are created from the Development Platform using a collection of opkg-formatted packages. The packages are built from source repositories using a cross-compiler environment that relies on the cross toolchain and staging tree from the Development Platform.

PiBox applications are written in C using GTK+ 2.x and Cairo. They install a configuration file announcing their presence to the app manager. The app manager chooses which apps to show and in what order, then displays them on the navigation page scaled to fit the display. Current apps include a themed analog clock, video playback, webcam, XMRadio and network configuration, among others.

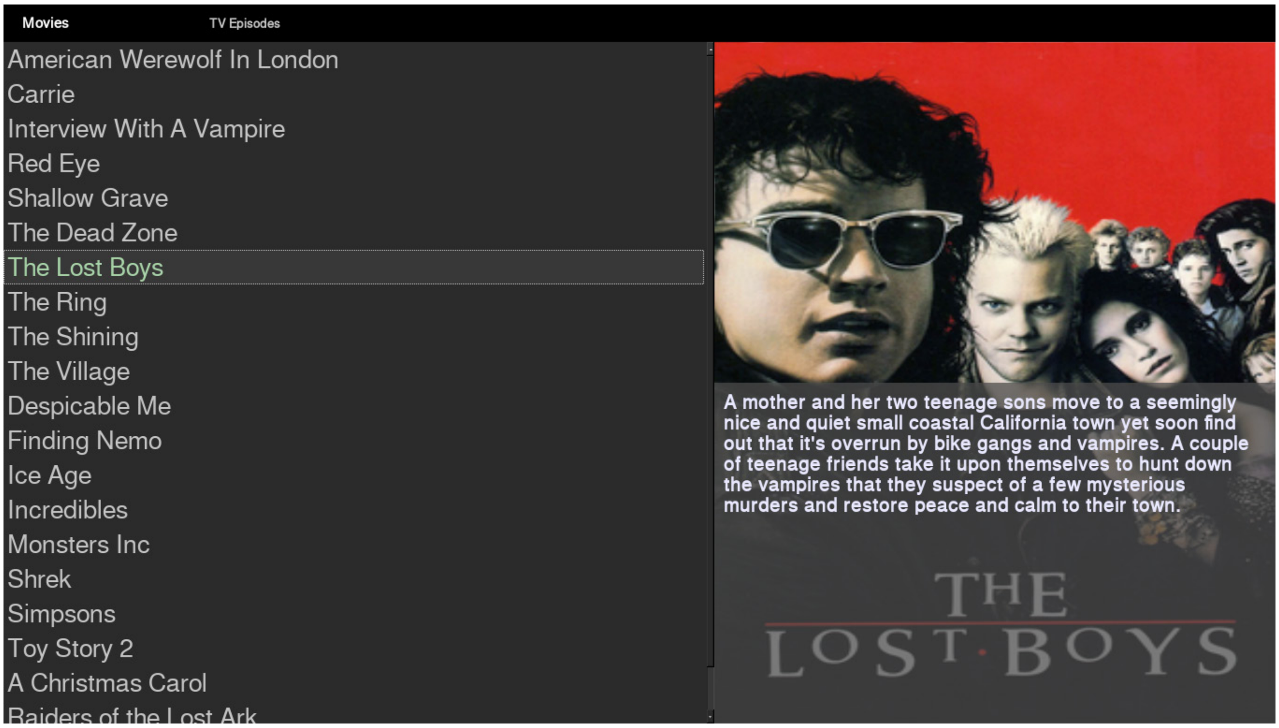
The Media systems are designed to be used from the primary console. However, additional functionality is available from a web UI, including a webcam stream and complete network configuration. The web UI provides user authentication and via client-side JavaScript mixed with a PHP back end that speaks a custom network-oriented protocol to PiBox dæmons. This UI recently received the initial stubs of a RESTful interface allowing a custom-designed Android keyboard to be used to control the console UI. Android support is still in early development as is a migration to a node.js REST API for IoT control and home automation.

All repositories are developed using `cdtools`, a simple bash configuration tool that allows easy context switching between projects while encouraging a common and consistent structure for all PiBox-related build systems.

Video Playback

The primary purpose of the Media systems is for server-provided video content distributed to players. "Players" includes both mobile devices and the custom PiBox Media Player.

Media sources are intended to be found on USB Flash media sticks. The Raspberry Pi has its own hardware-accelerated media player, omxplayer. PiBox integrated this



A mother and her two teenage sons move to a seemingly nice and quiet small coastal California town yet soon find out that it's overrun by bike gangs and vampires. A couple of teenage friends take it upon themselves to hunt down the vampires that they suspect of a few mysterious murders and restore peace and calm to their town.

Figure 5. VideoFE's video list, which is a sorted collection from multiple sources, is also searchable.

as a back-end player to a custom front-end UI component known simply as VideoFE. VideoFE is configurable to use other players, and it's been used on development desktops running xine.

VideoFE presents a list of videos and displays the poster of the current selection. Video data comes from The Movie DB via its JASON API. The data is retrieved using a custom Java application called VideoLib. The application runs on a desktop. It scans a directory tree and does a lookup by filename. The lookup can be customized if the filename isn't a good match, and The Movie DB provides alternative options that also can be selected. Once the directory tree is scanned correctly, the database is saved to the top of the tree, which should be the top of the media stick.

The UI has a customizable keymap file that provides a consistent interface for all apps. By default, the Esc key exits an application and returns the user to the navigation page. The arrow keys move around the navigation page. The design is intended to require as few keys as possible. The actual implementation of the Media systems makes use of a very small FAVI keyboard that is ideal for a variety of PiBox use cases.

Lessons Learned

As a learning tool, PiBox opened my eyes to a wide variety of problems. The first was the problem of switching hardware platforms easily. Crosstool-NG makes this easy, but wrapping this inside the PiBox build system made it easy to experiment quickly with different configurations. This allowed me to switch originally from the BeagleBone to the Raspberry Pi and recently aided in a quick migration to supporting the Raspberry Pi 2, which has a slightly different processor configuration. Early versions of PiBox toolchains also have been used for PowerPC and Intel targets.

During early development, I had to deal with device handling during boot. To reduce overhead and speed boot times, many embedded systems use mdev instead of udev for device handling. Decoding mdev events took some work and led to

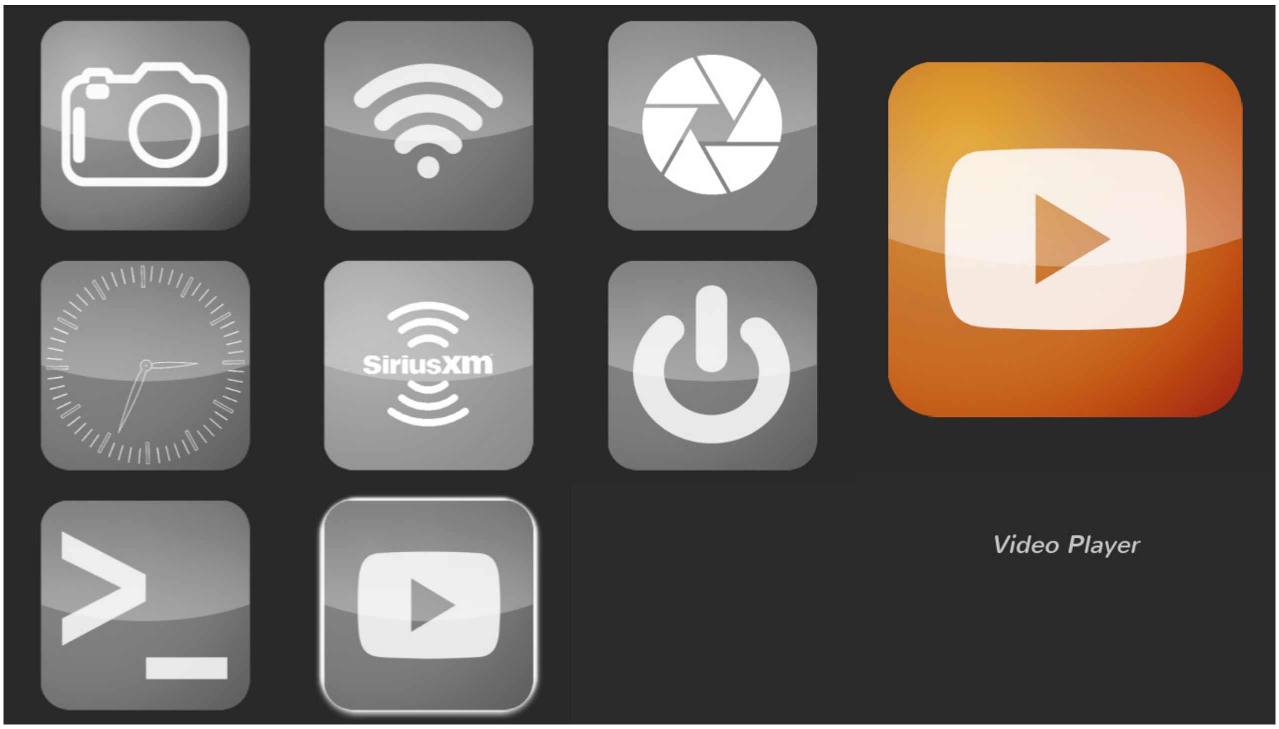
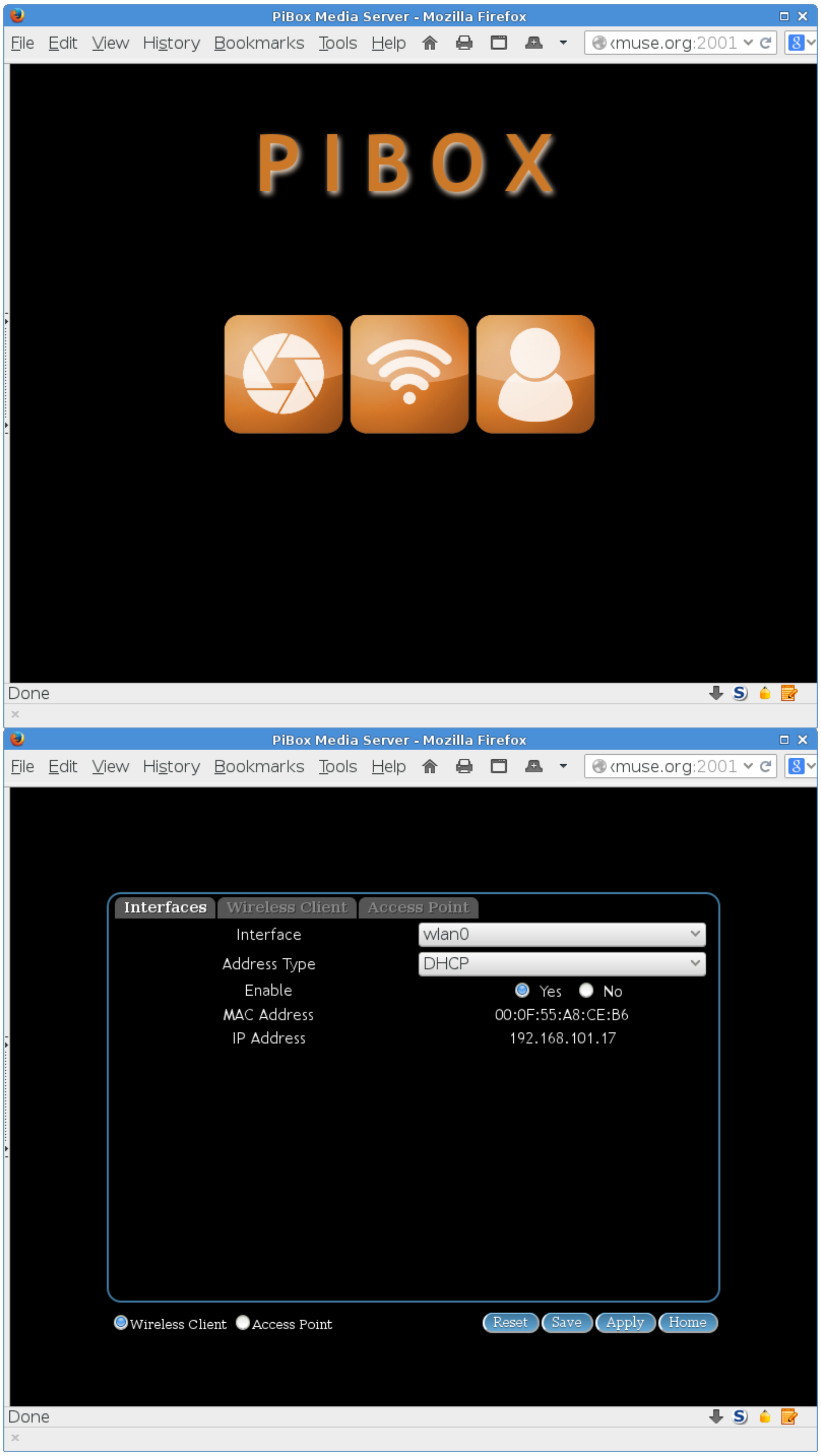


Figure 6. The console UI is consumer-oriented, with few keystrokes required for navigation.


PiBox Media Server - Mozilla Firefox
File Edit View History Bookmarks Tools Help
‹muse.org:2001
PIBOX
Done
Interfaces
Wireless Client
Access Point
Reset
Save
Apply
Home

Interface: wlan0
Address Type: DHCP
Enable: [✓ Yes] No
MAC Address: 00:0F:55:A8:CE:B6
IP Address: 192.168.101.17
[✓ Wireless Client] Access Point

Figure 7.
The WebUI supports user authentication and provides access to the webcam.

custom scripting to deal with USB device hot plug events, which were critical to the Media systems.

A related issue invovled removing early boot messages while speeding boot times. Boot times are still on the order of a minute or more (mostly due to network setup), but boot messages now are handled by the use of pslplash to show boot

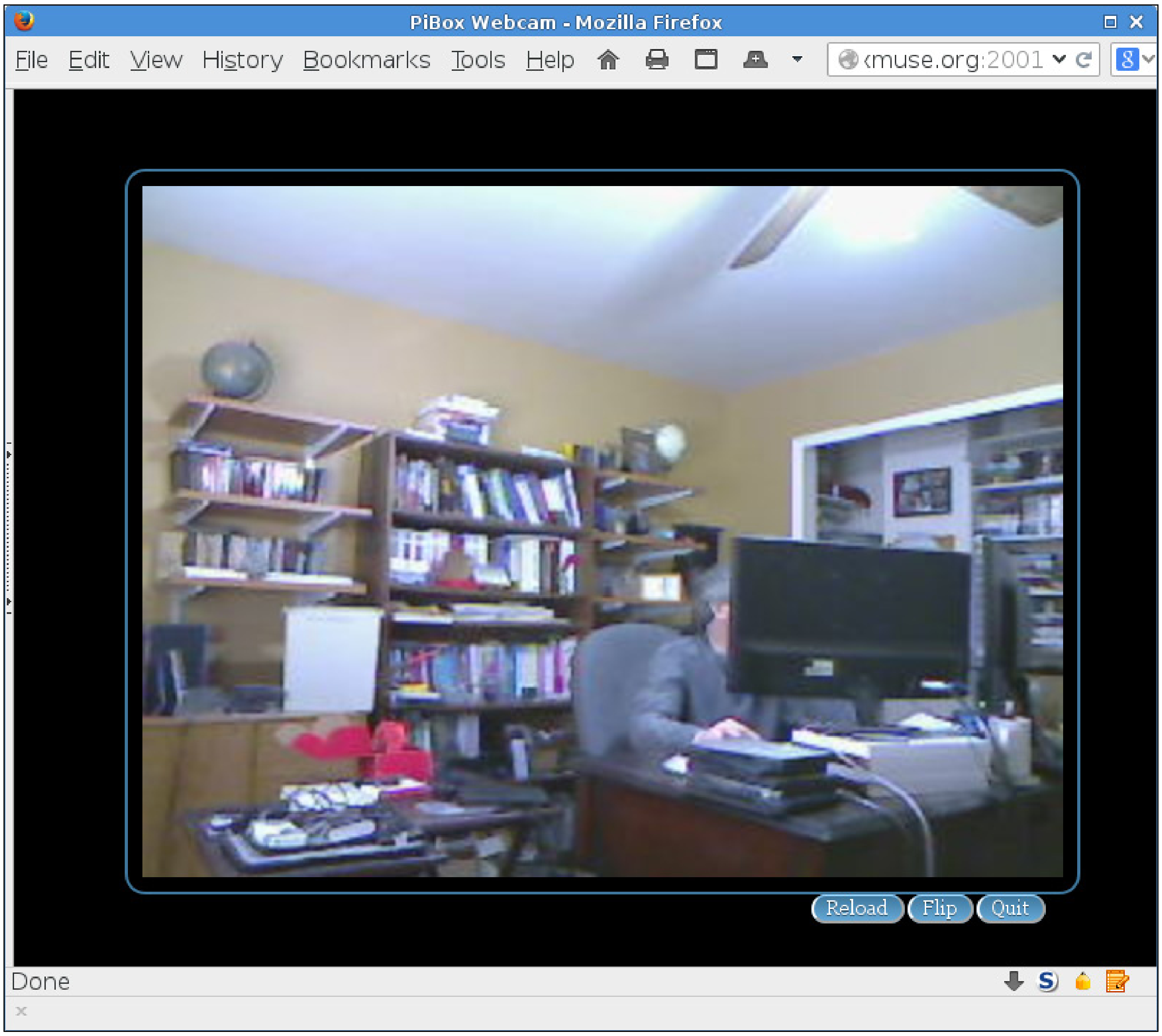

PiBox Webcam - Mozilla Firefox
File Edit View History Bookmarks Tools Help
‹muse.org:2001
Reload
Flip
Quit
Done

Figure 8. The webcam image can be flipped so the camera can be mounted from above or below a window.

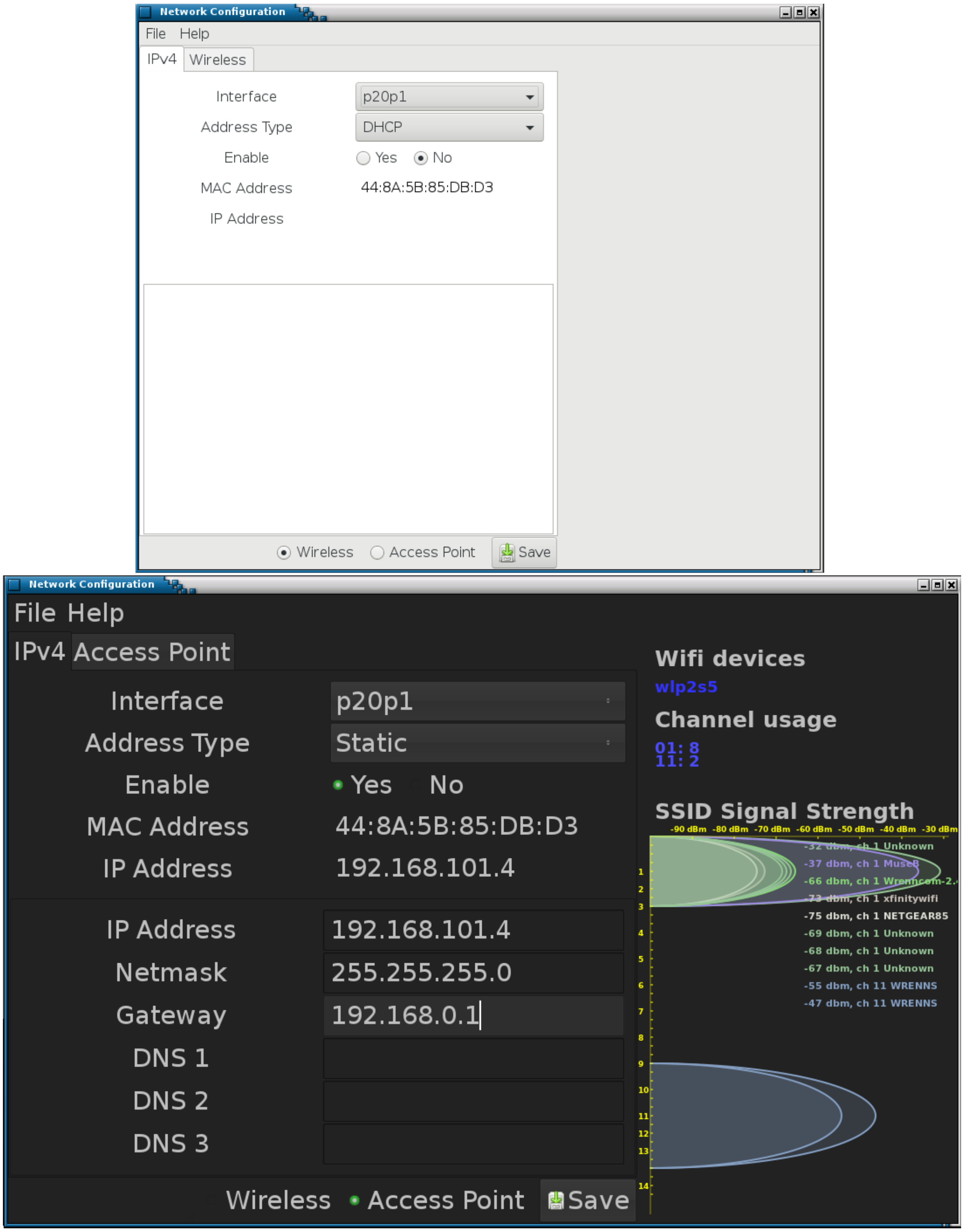
processing. I also had to start X.org very early without starting any clients. This is due to timing issues that prevent keyboards from operating properly. X is later stopped and restarted. This circumvents the non-functional keyboard issues without the user noticing the workaround.

PiBox offers both a console UI-based on GTK+/Cairo and a JavaScript-based interface based on the Monkey web server and a PHP back end. I originally looked at building an HTML5 interface (and would still like to in the long run), but WebKit wouldn't compile properly, and I didn't want to waste too much time trying to address that issue. So I fell back to what I already knew how to do: build a GTK+-based UI.

Cairo presented an estoric problem that proved difficult to isolate. Cairo is used by GTK+ to paint icons in the navigation page. This code originally was tested on my Intel-based desktop and worked fine. On the target board, the icons failed to be displayed. After much trial and error, I discovered that Cairo was unhappy within non-square images on the ARM board. Making them square fixed the problem. This adds an odd requirement for app developers when creating the navigation icons for their apps.

The web UI required me to learn more about JavaScript, a language I enjoy more now that I understand it a better. Its interaction with back-end PHP is clean and easy to understand, but creating UIs can require carrying client libraries around from application to application, much like Java. It does offer the option of setting up direct connections from the client browser to non-web-based back ends, and that's something I find very encouraging.

What I found less encouraging was the use of web-based media through proxies. The only format that moves natively through http/https is motion jpeg (mjpeg), and on the Pi, this works only at a relatively low frame rate. Note that this is a USB webcam, not the onboard camera. I chose to use a webcam due to the relatively short ribbon cables provided for the onboard camera. The use case for the webcam is as a backup camera for our trailer. The idea is to allow the camera to be physically distant from the Pi.



Interface: p20p1
Address Type: DHCP
Enable: Yes [✓ No]
MAC Address: 44:8A:5B:85:DB:D3
IP Address

[✓ Wireless] Access Point

Interface: p20p1
Address Type: Static
Enable: [✓ Yes] No
MAC Address: 44:8A:5B:85:DB:D3
IP Address: 192.168.101.4

IP Address: 192.168.101.4
Netmask: 255.255.255.0
Gateway: 192.168.0.1
DNS 1:
DNS 2:
DNS 3:

Wireless [✓ Access Point]

Wifi devices
wlp2s5

Channel usage
01: 8
11: 2

SSID Signal Strength

```
-32 dbm, ch 1 Unknown
-37 dbm, ch 1 MuseB
-66 dbm, ch 1 Wrenncom-2.4
-73 dbm, ch 1 xfinitywifi
-75 dbm, ch 1 NETGEAR85
-69 dbm, ch 1 Unknown
-68 dbm, ch 1 Unknown
-67 dbm, ch 1 Unknown
-55 dbm, ch 11 WRENNS
-47 dbm, ch 11 WRENNS
```

Figure 9. pibox-network-config is available in the development platform (above) but takes on a different appearance in the Media systems (below).

PiBox Media systems are designed to reduce user-managed wires. This includes networking, where WiFi rules. Unfortunately, not all WiFi dongles are created equal with all versions of the Linux kernel. After many months of trial and error, I managed to find that the Ralink RT5370 chip provided the most stable experience. However, support was left in place for the variety of chips I tested. Handling USB WiFi dongles is done manually, outside of mdev, with the help of USB device IDs and a custom init script.

While researching WiFi dongles, I also improved the network configuration utility I wrote from scratch. This GTK+ application provides support for configuring WiFi or wired static and DHCP connections and also allows configuring the Media Server as a wireless access point for the closed network in our travel trailer. This utility is available in the Development Platform, but with GTK+ themes, inherits the look of the custom UI for the Media systems. The utility comes with a WiFi scanner and also provides the `libpiboxnet` library. The library is used by `piboxd`, a dæmon that handles many functions including inbound requests from the web UI.

Summary

There are many other issues I covered over the years of PiBox development, including learning enough Blender to generate a 3D design for a sample enclosure—an experiment still in need of much refinement. But the design of the build system for the Development Platform has proven flexible enough for a variety of platforms. Cross-compilation turns out to be much simpler than I expected once you get a method identified. I use a common cross.sh script in many of the packages I build to simplify this process. I also found that the use of `cdtools` made it possible to create an all-encompassing metabuild that automates the download and build process for all packages at release time.

Future development plans include integration with remote sensors. I hope to create my own Iron Man home, with voice commands handled by my Jarvis project. Jarvis is a proof-of-concept voice-to-text-to-action-to-speech Java project. But the hope is to have it integrate with a back-end sensor management system, connected through a PiBox Media Server, to handle control of devices around the home.

This entire project has been a one-man job, and I'm very much interested in others joining in, even if all they do is fork the project. I'd like to hear how I can improve the build systems, the Media systems and their UIs and apps. I've created as much documentation as I can on the wiki, and interested developers can follow progress on my Redmine issue tracker. All code is released as open source and is available from GitLab.com. ■

Michael J. Hammel is a Software Engineer for NetApp living with his wife Brinda and two Golden Retrievers in Broomfield, Colorado. When he isn't working on embedded systems or other geekery, he likes to camp, walk his dogs around the park, and drink tea with his wife and revel in the joy of his daughter's success. He has written more than 100 articles for numerous online and print magazines, and he's the author of four books on GIMP, the GNU Image Manipulation Program.

Resources

- The Graphics Muse Development Wiki
- PiBox
- Cdtools
- PiBox on GitLab
- xjarvis
- Michael J. Hammel on GitLab
- "Building a Voice-Controlled Front End to IoT Devices" by Michael J. Hammel, *LJ* June 2018

Text Processing in Rust

Create handy command-line utilities in Rust.

By Mihalis Tsoukalos

This article is about text processing in Rust, but it also contains a quick introduction to pattern matching, which can be very handy when working with text.

Strings are a huge subject in Rust, which can be easily realized by the fact that Rust has two data types for representing strings as well as support for macros for formatting strings. However, all of this also proves how powerful Rust is in string and text processing.

Apart from covering some theoretical topics, this article shows how to develop some handy yet easy-to-implement command-line utilities that let you work with plain-text files. If you have the time, it'd be great to experiment with the Rust code presented here, and maybe develop your own utilities.

Rust and Text

Rust supports two data types for working with strings: `String` and `str`. The `String` type is for working with mutable strings that belong to you, and it has length and a capacity property. On the other hand, the `str` type is for working with immutable strings that you want to pass around. You most likely will see an `str` variable be used as `&str`. Put simply, an `str` variable is accessed as a reference to some UTF-8 data. An `str` variable is usually called a "string slice" or, even simpler, a "slice". Due to its nature, you can't add and remove any data from an existing `str` variable. Moreover, if you try to call the `capacity()` function on an `&str` variable, you'll get an error message similar to the following:

```
error[E0599]: no method named 'capacity' found for type
 ↪'&str' in the current scope
```

Generally speaking, you'll want to use an `str` when you want to pass a string as a function parameter or when you want to have a read-only version of a string, and then use a `String` variable when you want to have a mutable string that you want to own.

The good thing is that a function that accepts `&str` parameters can also accept `String` parameters. (You'll see such an example in the `basicOps.rs` program presented later in this article.) Additionally, Rust supports the `char` type, which is for representing single Unicode characters, as well as string literals, which are strings that begin and end with double quotes.

Finally, Rust supports what is called a `byte` string. You can define a new `byte` string as follows:

```
let a_byte_string = b"Linux Journal";
```

unwrap()

You almost certainly cannot write a Rust program without using the `unwrap()` function, so let's take a look at that here. Rust does not have support for `null`, `nil` or `Null`, and it uses the `Option` type for representing a value that may or may not exist. If you're sure that some `Option` or `Result` variable that you want to use has a value, you can use `unwrap()` and get that value from the variable.

However, if that value doesn't exist, your program will panic. Take a look at the following Rust program, which is saved as unwrap.rs:

```
use std::net::IpAddr;

fn main() {
    let my_ip = "127.0.0.1";
    let parsed_ip: IpAddr = my_ip.parse().unwrap();
    println!("{}", parsed_ip);

    let invalid_ip = "727.0.0.1";
    let try_parsed_ip: IpAddr = invalid_ip.parse().unwrap();
    println!("{}", try_parsed_ip);
}
```

Two main things are happening here. First, as `my_ip` is a valid IPv4 address, `parse().unwrap()` will be successful, and `parsed_ip` will have a valid value after the call to `unwrap()`.

However, as `invalid_ip` is not a valid IPv4 address, the second attempt to call `parse().unwrap()` will fail, the program will panic and the second `println!()` macro will not be executed. Executing `unwrap.rs` will verify all these:

```
$ ./unwrap
127.0.0.1
thread 'main' panicked at 'called 'Result::unwrap()'
 ↪on an 'Err'
value: AddrParseError(())', libcore/result.rs:945:5
note: Run with 'RUST_BACKTRACE=1' for a backtrace.
```

This means you should be extra careful when using `unwrap()` in your Rust programs. Unfortunately, going into more depth on `unwrap()` and how to avoid panic situations is beyond the scope of this article.

The `println!` and `format!` Macros

Rust supports macros, including `println!` and `format!` that are related to strings.

A Rust macro lets you write code that writes other code, which is also known as metaprogramming. Although macros look a lot like Rust functions, they have a fundamental difference from Rust functions: macros can have a variable number of parameters, whereas the signature of a Rust function must declare its parameters and define the exact type of each one of those function parameters.

As you might already know, the `println!` macro is used for printing output to the UNIX standard output, whereas the `format!` macro, which works in the same way as `println!`, returns a new `String` instead of writing any text to standard output.

The Rust code of `macros.rs` will try to clarify things:

```
macro_rules! hello_world{
    () => {
        println!("Hello World!")
    };
}

fn double(a: i32) -> i32 {
    return a + a
}

fn main() {
    // Using the format!() macro
    let my_name = "Mihalis";
    let salute = format!("Hello {}!", my_name);
    println!("{}", salute);

    // Using hello_world
    hello_world!();

    // Using the assert_eq! macro
    assert_eq!(double(12), 24);
    assert_eq!(double(12), 26);
}
```

What knowledge do you get from `macros.rs`? First, that macro definitions begin with `macro_rules!` and can contain other macros in their implementation. Note that this is a very naïve macro that does nothing really useful. Second, you can see that `format!` can be very handy when you want to create your own strings using your own format. Third, the `hello_world` macro created earlier should be called as `hello_world!()`. And finally, this shows that the `assert_eq!()` macro can help you test the correctness of your code.

Compiling and running `macros.rs` produces the following output:

```
$ ./macros
Hello Mihalis!
Hello World!
thread 'main' panicked at 'assertion failed: '(left == right)'
  left: '24',
 right: '26'', macros.rs:22:5
note: Run with 'RUST_BACKTRACE=1' for a backtrace.
```

Additionally, you can see an advantage of the `assert_eq!` macro here: when an `assert_eq!` macro fails, it also prints the line number and the filename of the assertion, which can't be done using a function.

Working with Strings

Now let's look at how to perform basic text operations in Rust. The Rust code for this example is saved in `basicOp.rs` and is the following:

```
fn accept(s: &str) {
    println!("{}", s);
}

fn main() {
    // Define a str
    let l_j: &str= "Linux Journal";
    // Or
```

```
let magazine: &'static str = "magazine";
// Use format! to create a String
let my_str = format!("Hello {} {}!", l_j, magazine);
println!("my_str L:{} C:{}", my_str.len(),
 ↪my_str.capacity());

// String character by character
for c in my_str.chars() {
    print!("{} ", c);
}
println!();

for (i, c) in my_str.chars().enumerate() {
    print!("{}:{} ", c, i);
}
println!();

// Convert string to number
let n: &str = "10";
match n.parse::<i32>() {
  Ok(n) => println!("{} is a number!", n),
  Err(e) => println!("{} is NOT a number!", e),
}

let n1: &str = "10.2";
match n1.parse::<i32>() {
  Ok(n1) => println!("{} is a number!", n1),
  Err(e) => println!("{}: {}", n1, e),
}

// accept() works with both str and String
let my_str = "This is str!";
let mut my_string = String::from("This is string!");
```

```
    accept(&my_str);
    accept(&my_string);

    // my_string has capacity
    println!("my_string L:{} C:{}", my_string.len(),
     ↪my_string.capacity());
    my_string.push_str("OK?");
    println!("my_string L:{} C:{}", my_string.len(),
     ↪my_string.capacity());

    // Convert String to str
    let s_str: &str = &my_string[..];
    // Convert str to String
    let s_string: String = s_str.to_owned();
    println!("s_string: L:{} C:{}", s_string.len(),
     ↪s_string.capacity());
}
```

So, first you can see two ways for defining `str` variables and creating a `String` variable using the `format!` macro. Then, you can see two techniques for iterating over a string character by character. The second technique also returns an index to the string that you process. After that, this example shows how to convert a string into an integer, if it's possible, with the help of `parse::<i32>()`. Next, you can see that the `accept()` function accepts both an `&str` and a `String` parameter even though its definition mentions an `&str` parameter. Following that, this shows the capacity and the length properties of a `String` variable, which are two different things. The length of a `String` is the size of the `String`, whereas the capacity of a `String` is the room that is currently allocated for that `String`. Finally, you can see how to convert a `String` to `str` and vice versa. Other ways for getting a `String` from an `&str` variable include the use of `.to_string(), String::from(), String::push_str(), format!() and .into().`

Executing `basicOp.rs` generates the following output:

```
$ ./basicOp
my_str L:29 C:32
H e l l o   L i n u x   J o u r n a l   m a g a z i n e !
H:0 e:1 l:2 l:3 o:4  :5 L:6 i:7 n:8 u:9 x:10  :11 J:12 o:13
 ↪u:14 r:15
n:16 a:17 l:18  :19 m:20 a:21 g:22 a:23 z:24 i:25 n:26 e:27
 ↪!:28
10 is a number!
10.2: invalid digit found in string
This is str!
This is string!
my_string L:15 C:15
my_string L:18 C:30
s_string: L:18 C:18
```

Finding Palindrome Strings

Now, let's look at a small utility that checks whether a string is a palindrome. The string is given as a command-line argument to the program. The logic of `palindrome.rs` is found in the implementation of the `check_palindrome()` function, which is implemented as follows:

```
pub fn check_palindrome(input: &str) -> bool {
    if input.len() == 0 {
        return true;
    }
    let mut last = input.len() - 1;
    let mut first = 0;

    let my_vec = input.as_bytes().to_owned();

    while first < last {
        if my_vec[first] != my_vec[last] {
            return false;
        }

        first +=1;
        last -=1;
    }
    return true;
}
```

The key point here is that you convert the string to a vector using a call to `as_bytes().to_owned()` in order to be able to access it as an array. After that, you keep processing the input string from both its left and its right side, one character from each side for as long as both characters are the same or until you pass the middle of the string. In that case, you are dealing with a palindrome, so the function returns "true"; otherwise, the function returns "false".

Executing `palindrome.rs` with various types of input generates the following kind of output:

```
$ ./palindrome 1
1 is a palindrome!
$ ./palindrome
Usage: ./palindrome string
$ ./palindrome abccba
abccba is a palindrome!
$ ./palindrome abcba
abcba is a palindrome!
$ ./palindrome acba
acba is not a palindrome!
```

Pattern Matching

Pattern matching can be very handy, but you should use it with caution, because it can create nasty bugs in your software. Pattern matching in Rust happens with the help of the `match` keyword. A match statement must catch all the possible values of the used variable, so having a default branch at the end of the block is a very common practice. The default branch is defined with the help of the underscore character, which is a synonym for "catch all". In some rare situations, such as when you examine a condition that can be either true or false, a default branch is not needed. A pattern-matching block can look like the following:

```
let salute = match a_name
{
"John" => "Hello John!",
"Jim" => "Hello Boss!",
"Jill" => "Hello Jill!",
_ => "Hello stranger!"
};
```

What does that block do? It matches one of the three distinct cases, if there is match, or it goes to the match all cases, which is last. If you want to perform more complex tasks that require the use of regular expressions, the regex crate might be more appropriate.

A Version of `wc` in Rust

Now let's look at the implementation of a simplified version of the `wc(1)` command-line utility. The Rust version of the utility will be saved as wc.rs, will not support any command-line flags, will consider every command-line argument as a file, and it can process multiple text files. The Rust version of wc.rs is the following:

```
use std::env;
use std::io::{BufReader, BufRead};
use std::fs::File;

fn main() {
    let mut lines = 0;
    let mut words = 0;
    let mut chars = 0;
}



    let args: Vec<_> = env::args().collect();
    if args.len() == 1 {
        println!("Usage: {} text_file(s)", args[0]);
        return;
    }

    let n_args = args.len();
    for x in 1..n_args {
        let mut total_lines = 0;
        let mut total_words = 0;
        let mut total_chars = 0;

        let input_path = ::std::env::args().nth(x).unwrap();
        let file = BufReader::new(File::open(&input_path)
↪.unwrap());
        for line in file.lines() {
            let my_line = line.unwrap();
            total_lines = total_lines + 1;
            total_words += my_line.split_whitespace().count();
            total_chars = total_chars + my_line.len() + 1;
        }

        println!("\t{}\t{}\t{}\t{}", total_lines, total_words,
 ↪total_chars, input_path);
        lines += total_lines;
        words += total_words;
        chars += total_chars;
    }

    if n_args-1 != 1 {
        println!("\t{}\t{}\t{}\ttotal", lines, words, chars);
    }
}
```

First, you should know that `wc.rs` is using buffered input for processing its text files. Apart from that, the logic of the program is found in the inner `for` loop that reads each input file line by line. For each line it reads, it counts the characters and words. Counting the characters of a line is as simple as calling the `len()` function. Counting the words of a line requires splitting the line using `split_whitespace()` and counting the number of elements in the generated iterator.

The other thing you should think about is resetting the `total_lines`, `total_words` and `total_chars` counters after processing a file. The `lines`, `words` and `chars` variables hold the total number of lines, words and characters read from all processed text files.

Executing `wc.rs` generates the following kind of output:

```
$ rustc wc.rs
$ ./wc
Usage: ./wc text_file(s)
$ ./wc wc.rs
    40      124     1114    wc.rs
$ ./wc wc.rs palindrome.rs
    40      124     1114    wc.rs
    39      104     854     palindrome.rs
    79      228     1968    total
$ wc wc.rs palindrome.rs
      40     124    1114 wc.rs
      39     104     854 palindrome.rs
      79     228    1968 total
```

The last command executed `wc(1)` in order to verify the correctness of the output of `wc.rs`.

As an exercise, you might try creating a separate function for counting the lines, words and characters of a text file.

Matching Lines That Contain a Given String

In this section, you'll see how to show the lines of a text file that match a given string—both the filename and the string will be given as command-line arguments to the utility, which is named `match.rs`. Here's the Rust code for match.rs:

```
use std::env;
use std::io::{BufReader,BufRead};
use std::fs::File;

fn main() {
let mut total_lines = 0;
    let mut matched_lines = 0;
    let args: Vec<_> = env::args().collect();

if args.len() != 3 {
  println!("{} filename string", args[0]);
        return;
  }

let input_path = ::std::env::args().nth(1).unwrap();
let string_to_match = ::std::env::args().nth(2).unwrap();
let file = BufReader::new(File::open(&input_path).unwrap());
for line in file.lines() {
total_lines += 1;
let my_line = line.unwrap();
if my_line.contains(&string_to_match) {
println!("{}", my_line);
            matched_lines += 1;
}
}
println!("Lines processed: {}", total_lines);
println!("Lines matched: {}", matched_lines);
 }
```

All the dirty work is done by the `contains()` function that checks whether the line that is currently being processed contains the desired string. Apart from that, the rest of the Rust code is pretty trivial.

Building and executing `match.rs` generates output like this:

```
$ ./match tabSpace.rs t2s
fn t2s(input: &str, n: i32) {
        t2s(&input_path, n_space);
Lines processed: 56
Lines matched: 2
$ ./match tabSpace.rs doesNotExist
Lines processed: 56
Lines matched: 0
```

Converting between Tabs and Spaces

Next, let's develop a command-line utility that can convert tabs to spaces in a text file and vice versa. Each tab is replaced with four space characters and vice versa.

This utility requires at least two command-line parameters: the first one should indicate whether you want to replace tabs with spaces or the other way around. After that, you should give the path of at least one text file. The utility will process as many text files as you want, just like the `wc.rs` utility presented earlier in this article.

You can find `tabSpace.rs`'s logic in the following two Rust functions:

```
fn t2s(input: &str) {
let file = BufReader::new(File::open(&input).unwrap());
for line in file.lines() {
        let my_line = line.unwrap();
        let new_line = my_line.replace("\t", "    ");
        println!("{}", new_line);
    }
}

fn s2t(input: &str) {
let file = BufReader::new(File::open(&input).unwrap());
for line in file.lines() {
        let my_line = line.unwrap();
        let new_line = my_line.replace("    ", "\t");
        println!("{}", new_line);
    }
}
```

All the work is done by `replace()`, which replaces every occurrence of the first pattern with the second one. The return value of the `replace()` function is the altered version of the input string, which is what's printed on your screen.

Executing `tabSpace.rs` creates output like the following:

```
$ ./tabSpace -t basicOp.rs > spaces.rs
Processing basicOp.rs
$ mv spaces.rs basicOp.rs
$ ./tabSpace -s basicOp.rs > tabs.rs
Processing basicOp.rs
$ ./tabSpace -t tabs.rs > spaces.rs
Processing tabs.rs
$ diff spaces.rs basicOp.rs
```

The previous command verifies the correctness of `tabSpace.rs`. First, any tabs in `basicOp.rs` are converted into spaces and saved as `spaces.rs`, which afterward becomes the new `basicOps.rs`. Then, the spaces of basicOps.rs are converted into tabs and saved in tabs.rs. Finally, the tabs.rs file is processed, and all of its tabs are converted into spaces (spaces.rs). The last version of spaces.rs should be exactly the same as basicOps.rs.

It would be a very interesting exercise to add support for tabs of variable size in `tabSpace.rs`. Put simply, the number of spaces of a tab should be a variable that will be given as a command-line parameter to the utility.

Conclusion

So, is Rust good at text processing and working with text in general? Yes it is! Additionally, it should be clear that text processing is closely related to file I/O and (sometimes) to pattern matching and regular expressions.

The only rational way to learn more about text processing in Rust is to experiment on your own, so don't waste any more time, and give it a whirl. ■

Mihalis Tsoukalos is a UNIX administrator and developer, a DBA and mathematician who enjoys technical writing. He is the author of *Go Systems Programming* and *Mastering Go*. You can reach him at http://www.mtsoukalos.eu and @mactsouk.

Resources

- The Rust Programming Language
- Rust Documentation

By Jupyter—Is This the Future of Open Science?

Taking the scientific paper to the next level.

By Glyn Moody

Glyn Moody has been writing about the internet since 1994, and about free software since 1995. In 1997, he wrote the first mainstream feature about GNU/Linux and free software, which appeared in *Wired*. In 2001, his book *Rebel Code: Linux And The Open Source Revolution* was published. Since then, he has written widely about free software and digital rights. He has a blog, and he is active on social media: @glynmoody on Twitter or identi.ca, and +glynmoody on Google+.

In a recent article, I explained why open source is a vital part of open science. As I pointed out, alongside a massive failure on the part of funding bodies to make open source a key aspect of their strategies, there's also a similar lack of open-source engagement with the needs and challenges of open science. There's not much that the Free Software world can do to change the priorities of funders. But, a lot can be done on the other side of things by writing good open-source code that supports and enhances open science.

People working in science potentially can benefit from every piece of free software code—the operating systems and apps, and the tools and libraries—so the better those become, the more useful they are for scientists. But there's one open-source project in particular that already has had a significant impact on how scientists work—Project Jupyter:

> Project Jupyter is a set of open-source software projects that form the building blocks for interactive and exploratory computing that is reproducible and multi-language. The main application offered by Jupyter is the Jupyter Notebook, a

web-based interactive computing platform that allows users to author documents that combine live code, equations, narrative text, interactive dashboard and other rich media.

Project Jupyter was spun-off from IPython in 2014 by Fernando Pérez. Although it began as an environment for programming Python, its ambitions have grown considerably. Today, dozens of Jupyter kernels exist that allow other languages to be used. Indeed, the project itself speaks of supporting "interactive data science and scientific computing across all programming languages". As well as this broad-based support for programming languages, Jupyter is noteworthy for its power. It enables users to create and share documents that contain live code, equations, visualizations and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization and machine learning.

In a way, Project Jupyter is the ultimate scientific tool, since it can be used in any discipline and for multiple purposes. As an article in the *Atlantic* rightly put it, it also can be thought of as the scientific paper taken to the next level by exploiting the possibilities of digital technology. A key aspect is that it's interactive—readers can use the embedded code to explore the data and carry out limitless "what ifs". It's such an obvious idea, you may wonder why it hasn't been done before. And the answer is that it has, notably in the form of Mathematica from Wolfram Research.

Mathematica is an innovative and powerful program with one huge flaw: it's proprietary. As such, it suffers all the usual downsides, one of which more or less disqualifies it for science: you can't check the code. That means you don't really know why it produces the results it does; you just have to take it on trust. That's not science; that's voodoo.

Its closed-source nature means that Mathematica can't tap into the community of users in the same way open-source projects can. Whatever advantages Mathematica once had, it's only a matter of time before open-source alternatives like Jupyter surpass it. Indeed, it's interesting that the Google Trends comparison of searches for Mathematica and searches for Jupyter show that interest in the latter is rising, while Google searches for the former are falling. It's an inexact metric, of course, but the overall trends are clear: Mathematica, like Microsoft Windows, is the past, and Jupyter, like GNU/Linux, is the future.

It's not just about the familiar dynamics of open-source development. There's a key reason why Jupyter has beaten Mathematica, as the academic Paul Romer explained in a perceptive post:

> Mathematica failed, despite technical accomplishments, because the norms of its developers clashed so obviously with the norms of its intended users. Jupyter is succeeding because the norms of the community that is developing it are aligned with the norms of its users.

As well as its culture, there's another aspect of Jupyter that makes it a perfect fit for open science. On the page listing dozens of Jupyter notebooks—all freely accessible—there's a section titled "Reproducible academic publications":

> This section contains academic papers that have been published in the peer-reviewed literature or pre-print sites such as the ArXiv that include one or more notebooks that enable (even if only partially) readers to reproduce the results of the publication.

Coupled with the transparency of the underlying code, this ability for anybody to check the logic and results of a publication is a real breakthrough in open science. At the moment, most academic papers can be read only superficially. In theory, anyone could set about reproducing the final conclusions—at least, provided the relevant datasets are freely available. Few will take the trouble to do so though, because there are no academic incentives to expend all that time and energy. With papers published not as static documents, but as dynamic Jupyter notebooks with full open datasets, it is possible to check the results properly, as well as to plug in other datasets or tweak the underlying assumptions. In this way, Jupyter notebooks are the perfect marriage of open source, open access and open data. This is exactly how open science should work, but until now almost never does.

The power and flexibility of the Jupyter environment make it a strong foundation for open-ended experimentation of the kind the Free Software community relishes. Moreover, coding in this domain could have a major impact on scientists using the notebook format and on the science they produce. That combination of satisfying intellectual challenge with real-world practical benefits makes it a perfect candidate for open-source coders looking for new and meaningful challenges. ■

Send comments or feedback
via https://www.linuxjournal.com/contact
or email ljeditor@linuxjournal.com.