

Testing Code with
Python's pytest

Time for Net
Giants to Pay Up

Game Review:
Lamplight City

LINUX JOURNAL

Since 1994: The original magazine of the Linux community

MONITORING

Why Your Server Monitoring Sucks • The Evils of CloudWatch
How-To: Resource-Friendly Monitoring Solution



84 *DEEP DIVE: Monitoring*

85 **Why Your Server Monitoring (Still) Sucks**

by Mike Julian

Five observations about why your server monitoring still stinks by a monitoring specialist-turned-consultant.

96 **CloudWatch Is of the Devil, but I Must Use It**

by Corey Quinn

Let's talk about Amazon CloudWatch.

104 **Bare-Bones Monitoring with Monit and RRDtool**

by Andy Carlson

How to provide robust monitoring to low-end systems.

114 **How-To: Implementing a Real-Time Syslog Shipper for Your Terminal**

by Fabien Wernli

Ever wondered how to tail -F /var/log/messages from multiple servers at once? Read on.

130 **Taking System Monitoring to the Next Level: an Interview with Scalyr CEO Steve Newman**

by Petros Koutoupis

As computing ecosystems become more complex, monitoring and analyzing those often disconnected moving parts becomes increasingly challenging.

6 The Monitoring Issue

by Bryan Lunduke

10 From the Editor—Doc Searls

An Immodest Proposal for the Music Industry

19 Letters

UPFRONT

27 What's Your System's Uptime

by Ricardo Fraile

33 Patreon and *Linux Journal*

34 Getting Started with Scilab

by Joey Bernard

43 FOSS Project Spotlight: BlueK8s

by Tom Phelan

48 Lessons in Vendor Lock-in: Shaving

by Kyle Rankin

51 Reality 2.0: a *Linux Journal* Podcast

52 News Briefs

COLUMNS

55 Kyle Rankin's Hack and /

Schedule One-Time Commands with the UNIX at Tool

59 Reuven M. Lerner's At the Forge

Testing Your Code with Python's pytest

68 Dave Taylor's Work the Shell

Roman Numerals and Bash

74 Zack Brown's diff -u

What's New in Kernel Development

177 Glyn Moody's Open Sauce

Time for Net Giants to Pay Fairly for the Open Source on Which They Depend

ARTICLES

138 **Review: the Dell XPS 13 Developer Edition Laptop**

by *Petros Koutoupis*

A look at Dell's thin and sleek XPS 13 Developer Edition laptop that now ships with Ubuntu 18.04 LTS pre-installed.

151 **Chrome OS Stable Channel Gets Linux Apps**

by *Philip Raymond*

How to get started with Linux Apps for Chromebooks.

158 **About ncurses Colors**

by *Jim Hall*

Why does ncurses support only eight colors?

170 **Game Review: *Lamplight City***

by *Patrick Whelan*

A well lit look into Grundislav Game's latest release.

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: <http://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 <http://www.linuxjournal.com/digital>.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at <http://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: <http://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 <http://www.linuxjournal.com>. Subscribe for free today: <http://www.linuxjournal.com/newsletters>.

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

COMMUNITY ADVISORY BOARD

John Abreau, Boston Linux & UNIX Group; John Alexander, Shropshire Linux User Group;
Robert Belnap, Classic Hackers UGA Users Group; Aaron Chantrill, Bellingham Linux Users Group;
Lawrence D'Oliveiro, Waikato Linux Users Group; Chris Ebenezer, Silicon Corridor Linux User Group;
David Egts, Akron Linux Users Group; Michael Fox, Peterborough Linux User Group;
Braddock Gaskill, San Gabriel Valley Linux Users' Group; Roy Lindauer, Reno Linux Users Group;
Scott Murphy, Ottawa Canada Linux Users Group; Andrew Pam, Linux Users of Victoria;
Bob Proulx, Northern Colorado Linux User's Group; Ian Sacklow, Capital District Linux Users Group;
Ron Singh, Kitchener-Waterloo Linux User Group; Jeff Smith, Kitchener-Waterloo Linux User Group;
Matt Smith, North Bay Linux Users' Group; James Snyder, Kent Linux User Group;
Paul Tansom, Portsmouth and South East Hampshire Linux User Group;
Gary Turner, Dayton Linux Users Group; Sam Williams, Rock River Linux Users Group;
Stephen Worley, Linux Users' Group at North Carolina State University;
Lukas Yoder, Linux Users Group at Georgia Tech

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@linuxjournal.com

URL: www.linuxjournal.com/subscribe

Mail: 9597 Jones Rd, #331, Houston, TX 77065

SPONSORSHIPS

E-MAIL: sponsorship@linuxjournal.com

Contact: Publisher Carlie Fairchild

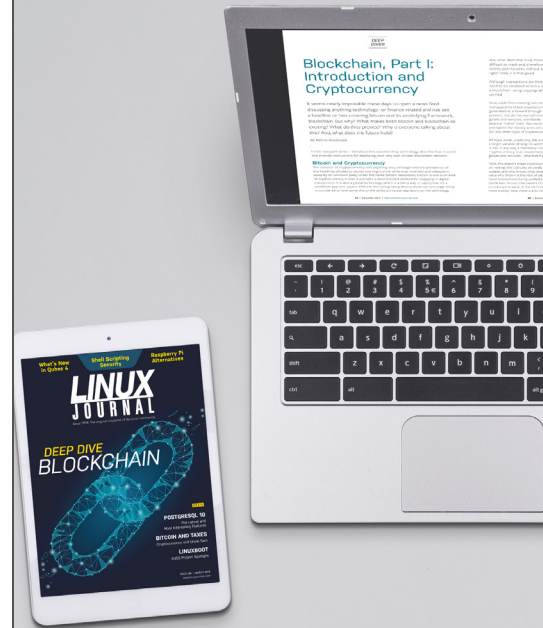
Phone: +1-281-944-5188

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 MONITORING ISSUE

In 1935, Austrian physicist, Erwin Schrödinger, still flying high after his Nobel Prize win from two years earlier, created a simple thought experiment.

It ran something like this:

If you have a file server, you cannot know if that server is up or down...until you check on it. Thus, until you use it, a file server is—in a sense—both up and down. At the same time.

This little brain teaser became known as Schrödinger's File Server, and it's regarded as the first known critical research on the intersection of Systems Administration and Quantum Superposition. (Though, why Erwin chose, specifically, to use a "file server" as an example remains a bit of a mystery—as the experiment works equally well with any type of server. It's like, we get it, Erwin. You have a nice NAS. Get over it.)

...

Okay, perhaps it didn't go exactly like that. But I'm confident it would have...you know...had good old Erwin had a nice Network Attached Storage server instead of a cat.



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*.

THE MONITORING ISSUE

Regardless, the lessons from that experiment certainly hold true for servers. If you haven't checked on your server recently, how can you be truly sure it's running properly? Heck, it might not even be running at all!

Monitoring a server—to be notified when problems occur or, even better, when problems look like they are about to occur—seems, at first blush, to be a simple task. Write a script to ping a server, then email me when the ping times out. Run that script every few minutes and, shazam, we've got a server monitoring solution! Easy-peasy, time for lunch!

Whoah, there! Not so fast!

That server monitoring solution right there? It stinks. It's fragile. It gives you very little information (other than the results of a ping). Even for administering your own home server, that's barely enough information and monitoring to keep things running smoothly.

Even if you have a more robust solution in place, odds are there are significant shortcomings and problems with it. Luckily, *Linux Journal* has your back—this issue is chock full of advice, tips and tricks for how to keep your servers effectively monitored.

You know, so you're not just guessing if the cat is still alive in there.

Mike Julian (author of O'Reilly's *Practical Monitoring*) goes into detail on a bunch of the ways your monitoring solution needs serious work in his adorably titled “Why Your Server Monitoring (Still) Sucks” article.

We continue “telling it like it is” with Corey Quinn's treatise on Amazon's CloudWatch, “CloudWatch Is of the Devil, but I Must Use It”. Seriously, Corey, tell us how you really feel.

With our cathartic, venting session behind us, we've got a detailed, hands-on walk-through of how to use Monit (an open-source process supervisor for Linux) coupled with RRDtool (a GPL'd tool for capturing data over long periods of time,

THE MONITORING ISSUE

such as from shell scripts, and graphing it) to monitor your server in a fairly simple, and very open-source, way.

Then, get this, we've got a sysadmin from the Computing Centre of the National Institute of Nuclear Physics and Particle Physics in France (Fabien Wernli) — seriously, how cool is that?—walking us through how to create a site-wide, low-latency (we're talking sub-millisecond here) log infrastructure.

Round that out with an interview with Steve Newman (one of the folks who created Writely, which you might know as Google Docs, following Google's acquisition in 2006) on his company, Scalyr, which handles server monitoring and log management—and you've got more server monitoring information than you can shake a stick at.

Or, you can go back to guessing if the cat is still alive. That's fun too. ■

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

Thanks to Sponsors **Linode** and **Pulseway**
for Supporting *Linux Journal*



linode

Cloud Hosting for You.

High performance SSD Linux servers for all of your infrastructure needs.

www.linode.com



System Management at Your Fingertips.

www.pulseway.com

Want to see your company's logo here?
Find out more, <https://www.linuxjournal.com/sponsors>.



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.

An Immodest Proposal for the Music Industry

How music listeners can fill the industry's “value gap”.

From the 1940s to the 1960s, countless millions of people would put a dime in a **jukebox** to have a single piece of music played for them, one time. If they wanted to hear it again, or to play another song, they'd put in another dime.

In today's music business, companies such as Spotify, Apple and Pandora pay fractions of a penny to stream songs to

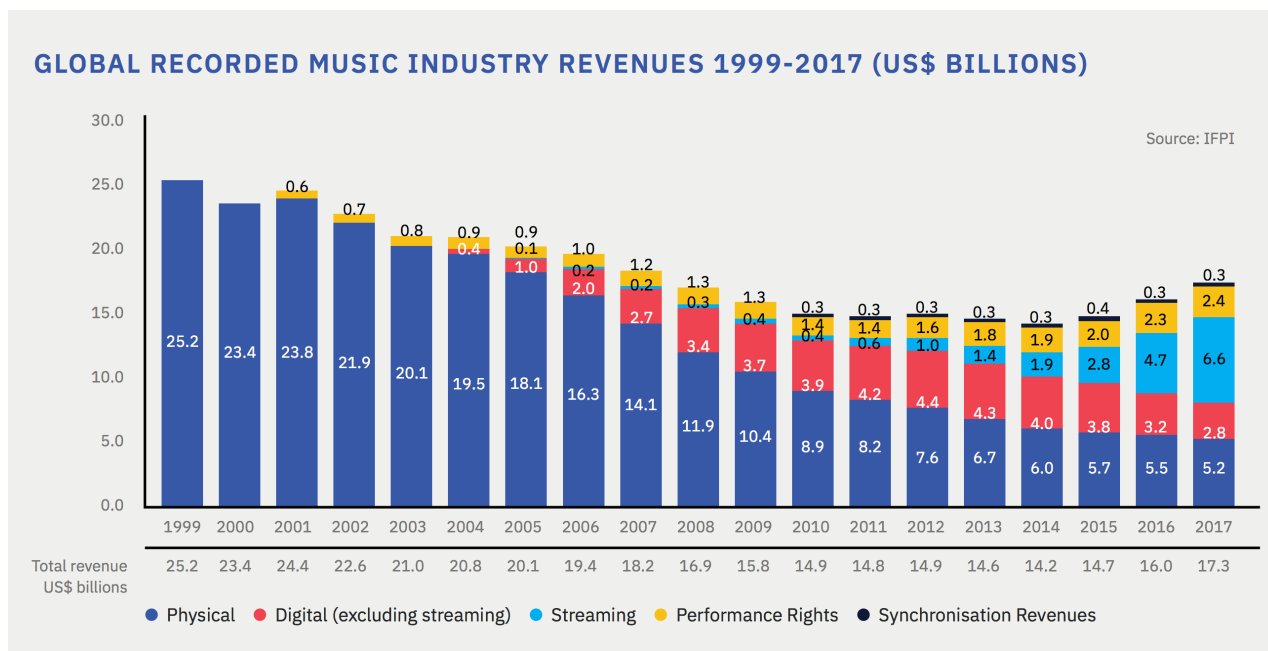


Figure 1. Global Music Report 2018

listeners. While this is a big business that continues to become bigger, it fails to cover what the music industry calls a “value gap”.

They have an idea for filling that gap. So do I. The difference is that mine can make them more money, with a strong hint from the old jukebox business.

For background, let’s start with the graph shown in Figure 1 from the IFPI’s [Global Music Report 2018](#).

You can see why [IFPI](#) no longer gives its full name: *International Federation of the Phonographic Industry*. That phonographic stuff is what they now call “physical”. And you see where that’s going (or mostly gone). You also can see that what once threatened the industry—“digital”—now accounts for most of its rebound (Figure 2).

The graphic shown in Figure 2 is also a call-out from the first. Beside it is this text: “Before seeing a return to growth in 2015, the global recording industry lost nearly 40% in revenues from 1999 to 2014.”

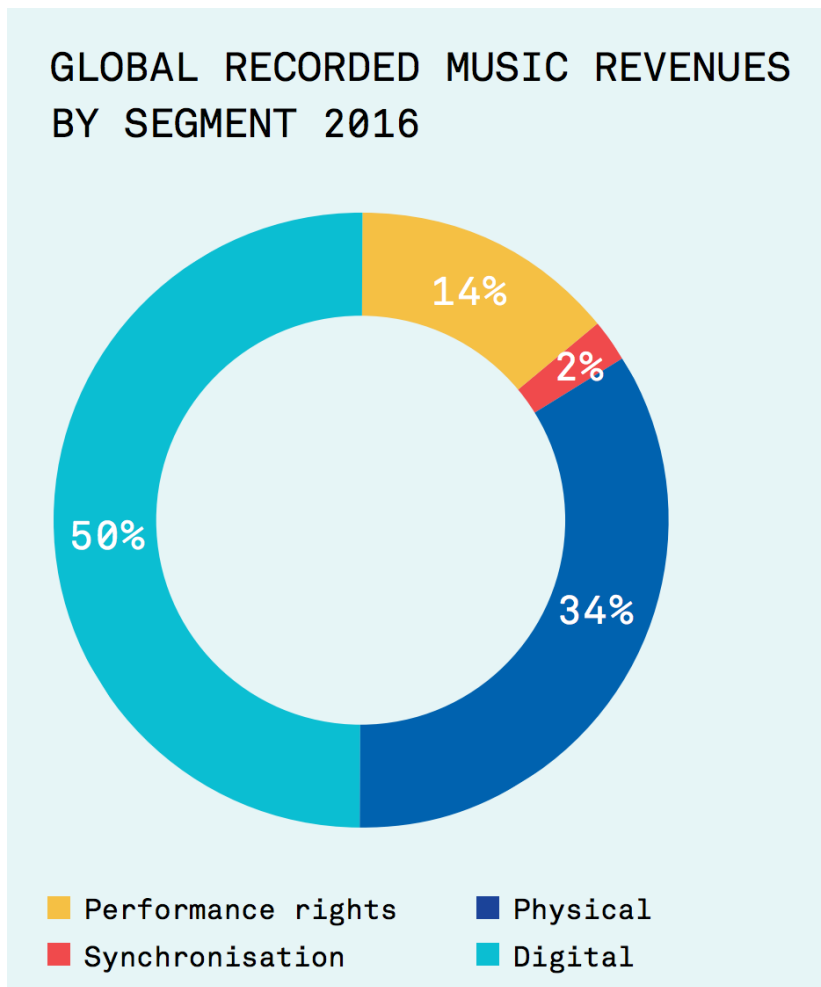


Figure 2. Global Recorded Music Revenues by Segment (2016)

Later, the report says:

However, significant challenges need to be overcome if the industry is going to move to sustainable growth. The whole music sector has united in its effort to fix the fundamental flaw in today’s music market, known as the “value gap”, where fair revenues are not being returned to those who are creating and investing in music.

They want to solve this by lobbying: “The value gap is now the industry’s single highest legislative priority as it seeks to create a level playing field for the digital market and secure the future of the industry.” This has worked before. Revenues from streaming and performance rights owe a lot to royalty and copyright rates and regulations guided by the industry. (In the early 2000s, I covered this like a rug in *Linux Journal*. See [here](#).)

FROM THE EDITOR

But, there's another way to fill that gap: on the *listening* side. You can see a hint in that direction from growth in live performance revenues. According to [Statista](#), live music industry revenue:

...will grow from 9.28 billion U.S. dollars in 2015 to 11.99 billion in 2021. Of the revenue generated in 2016, over two billion U.S. dollars was generated in sponsorship, and a further 7.4 billion U.S. dollars came in ticket sales. The industry is expected to grow further in the coming years as the compound annual growth rate for live music ticket sales is estimated at 5.23 percent between 2015 and 2020.

According to a July 16, 2018, post in *Pollstar*:

There is perhaps no better indicator of a robust 2018 live market than *Pollstar's* Mid-Year Top 50 Worldwide Tours chart. This year's survey saw a 12% jump in total gross from last year's \$1.97 billion to a record-setting \$2.21 billion—a \$240 million increase. It's the chart's biggest rise since 2015–16...

Concert promoters also are raising prices. Says a [July 9, 2018, report](#) by ABC News:

The average price of a concert ticket during the first six months of the year was \$46.69—4.2 percent higher than the average cost of a ticket for the same period last year, according to the latest figures from music industry magazine *Pollstar*. That price is almost 7 percent higher than the average for all of 2000, and an even more startling 43 percent increase over what concert tickets cost just three years ago, according to *Pollstar*.

That same report says sales are going down: a market signal that the prices are too high. But hey, people are clearly willing to pay a lot for live music and a participatory experience. This is an important clue.

Participation requires good signaling from both sides of the marketplace. So let's look at the demand side, shown in Figure 3, starting with what the streaming services pay to play us a tune.

FROM THE EDITOR

	DSP / STORE	PER STREAM	Streams		Streams Per Song	Streams Per Album	Marketshare	
			Quantity % of Total	Amount % of Total			Streams	Marketshare Amount
1	Spotify	\$0.00397	47.78%	51.51%	154	1,535	47.78%	51.51%
2	Apple iTunes	\$0.00783	10.48%	22.29%	78	778	10.48%	22.29%
3	Pandora	\$0.00134	21.56%	7.86%	454	4,538	21.56%	7.86%
4	Google	\$0.00611	2.41%	4.00%	100	996	2.41%	4.00%
5	Amazon	\$0.00740	1.89%	3.80%	82	823	1.89%	3.80%
6	Deezer	\$0.00624	1.91%	3.24%	98	976	1.91%	3.24%
7	Tidal	\$0.01284	0.50%	1.76%	47	474	0.50%	1.76%
8	Rhapsody	\$0.01682	0.38%	1.75%	36	362	0.38%	1.75%
9	YouTube	\$0.00074	8.38%	1.70%	818	8,181	8.38%	1.70%
10	Xbox Music	\$0.02730	0.09%	0.65%	22	223	0.09%	0.65%
11	Telecom Italia	\$0.01410	0.06%	0.24%	43	432	95.38%	98.57%
12	24-7 Entertainment	\$0.01461	0.05%	0.20%	42	417		
13	UMA	\$0.00022	2.71%	0.16%	2,831	28,311		
14	Yandex LLC	\$0.00039	1.38%	0.15%	1,555	15,551		
15	Qobuz	\$0.03816	0.01%	0.14%	16	160		
16	KKBOX	\$0.00369	0.12%	0.12%	165	1,652		
17	Peloton	\$0.04785	0.01%	0.08%	13	127		
18	iHeartRadio	\$0.01318	0.02%	0.06%	46	462		
19	Zed	\$0.04576	0.00%	0.06%	13	133		
20	Saavn	\$0.00160	0.09%	0.04%	381	3,807		
21	NMusic	\$0.00873	0.02%	0.04%	70	697		
22	Slacker	\$0.00502	0.02%	0.03%	121	1,213		
23	Cricket	\$0.00927	0.01%	0.03%	66	657		
24	Touchtunes	\$0.01409	0.00%	0.02%	43	432		
25	Akazoo	\$0.53026	0.00%	0.01%	1	11		
26	Turkcell	\$0.00462	0.01%	0.01%	132	1,318		
27	Naxos	\$0.05200	0.00%	0.01%	12	117		
28	PlayNetwork	\$0.00042	0.10%	0.01%	1,443	14,426		
29	iMusica	\$0.02314	0.00%	0.01%	26	263		
30	AWA	\$0.00894	0.00%	0.01%	68	681		
			100.00%	100.00%				

Figure 3. What the Streaming Services Pay to Play a Tune (from *Trichordist*)

To make that clearer, the top three streamers pay between 13.4% (Pandora) and 78.3% of a penny (\$.01) to play you a song.

SiriusXM pays ([it says here](#)) “19.1% of the price of all audio packages which include music channels”. That means the \$209.76 I paid last August for my SiriusXM subscription sent \$40.01 to the rights-holders of all the music played on all the SiriusXM channels for my account over the following year, whether I listened to any music or not. (And mostly I listen to non-music channels.)

YouTube is another special case. The 2018 IFPI report says, “From publicly available

FROM THE EDITOR

data, IFPI estimates that Spotify paid record companies US\$20 per user in 2015, the last year of available data. By contrast, it is estimated that YouTube returned less than US\$1 for each music user.” That’s a big part of the “value gap”.

Some of those rates are negotiated, others are set by regulation, and most are informed—one way or another—by both.

In no case, however, does the music listener pay for digital music on the jukebox model: with cash on a per-listen, per-song basis. (Note that a dime in 1960 was more than 100x what a streamer pays for the right to play a song for somebody.)

So that’s my proposal: ***create an easy way for any of us to pay what we want for the music we hear***. This will give music lovers their own way to close the value gap—and then some.

As it happens, an easy way to do this **was proposed** by **ProjectVRM** (which I run at Harvard’s **Berkman Klein Center**) way back in 2009. It’s called **EmanciPay**, and here is how it is described on the project wiki:

Simply put, EmanciPay makes it easy for anybody to pay (or offer to pay) —

1. as much as they like
2. however they like
3. for whatever they like

on their own terms

— or at least to start with that full set of options, and to work out differences with sellers easily and with minimal friction.

EmanciPay turns consumers (aka users) into customers by giving them a pricing gun (something which in the past only sellers used) and their own means to make offers, to pay outright, and to escrow the intention to pay when price and other requirements are

FROM THE EDITOR

met. Payments themselves can also be escrowed.

In slightly more technical terms, EmanciPay is a payment framework for customers operating with full agency in the open marketplace. It operates on open protocols and standards, so it can be used by any buyer, seller or intermediary...

So, as currently planned, EmanciPay would —

1. Provide a single and easy way that consumers of “content” can become customers of it. In the current system—which isn’t one—every artist, every musical group, every public radio and TV station, has his, her or its own way of taking in contributions from those who appreciate the work. This can be arduous and time-consuming for everybody involved. (Imagine trying to pay separately every musical artist you like, for all your enjoyment of each artist’s work.) What EmanciPay proposes, however, is not a replacement for existing systems, but a new system that can supplement existing fund-raising systems—one that can soak up much of today’s MLOTT: Money Left On The Table.
2. Provide ways for individuals to look back through their media usage histories, inform themselves about what they have been enjoying, and to determine how much it is worth to them. The Copyright Arbitration Royalty Panel (CARP), and later the Copyright Royalty Board (CRB), both came up with “rates and terms that would have been negotiated in the marketplace between a willing buyer and a willing seller”—language that first appeared in the 1995 Digital Performance Royalty Act (DPRA), and was tweaked in 1998 by the Digital Millennium Copyright Act (DMCA), under which both the CARP and the CRB operated. The rates they came up with peaked at \$.0001 per “performance” (a song or recording), per listener. EmanciPay creates the “willing buyer” that the DPRA thought wouldn’t exist.
3. Stigmatize non-payment for worthwhile media goods. This is where “social” will finally come to be something more than yet another tech buzzmodifier.

All these require micro-accounting, not micro-payments. In fact micro-accounting can inform ordinary payments that can be made in clever new ways that should satisfy

FROM THE EDITOR

everybody with an interest in seeing artists compensated fairly for their work. An individual listener, for example, can say “I want to pay one cent for every song I hear on the radio”, and “I’ll send [SoundExchange](#) a lump sum of all the pennies I wish to pay for songs I hear over the course of a year, along with an accounting of what artists and songs I’ve listened to”—and leave dispersal of those totaled pennies up to the kind of agency that likes, and can be trusted, to do that kind of thing.

Similar systems can also be put in place for readers of newspapers, blogs and other journals. What’s important is that the control is in the hands of the individual, and that the accounting and dispersal systems work the same way for everybody.

I also proposed this earlier in [“EmanciPay: A Content Monetization Plan for Newspapers”](#), and later in [“An Easy Way to Pay for Journalism, Music and Everything Else We Like”](#). In the first of those, I wrote:

Think of EmanciPay as a way to unburden sellers of the need to keep trying to control markets that are beyond their control anyway. Think of it as a way that “free market” can mean more than “your choice of captor”. Think of it as a way that “customer relationships” can be worthy of the label because both sides are carrying their ends of the relationship burden—rather than the sellers’ side carrying the whole thing.

It’ll be fun to start doing that in the music industry.

A number of developments make the opportunity ripe now:

1. The music industry is far less scattered and conflicted about its nature (digital now) and future (gotta make up that value gap) than it ever was in the past.
2. Former enemies can be friends. For example, open source and the music industry have both won, and many aligned incentives can be found between them.
3. Music listeners are clearly willing to pay value for value. We just need to create the ways. And it shouldn’t be hard. (Especially for *Linux Journal* readers.)
4. The jukebox and live performance examples both suggest that people shouldn’t

FROM THE EDITOR

have a problem saying “I’ll be glad to set up a way to pay one cent every time I hear music I like.”

5. Apple just bought Shazam, which is a way to identify music people hear. That kind of functionality can be brought into standard ways people can pay for music they passively hear (for example, in a restaurant or at parties) and like.
6. We’ve long needed a standards-based approach to tipping artists—or anybody—with maximal ease and minimal friction. One can be crafted out of work on EmanciPay.

While most of my usual appeals in *Linux Journal* are to the hackers among us, this appeal is mostly to my friends (old and new) in the music industry. They have the connections, the talent, the legal smarts, the money and the motivation required to make this thing work.

So let’s bring the people who love music into the marketplace as willing buyers. And let’s do it by standardizing simple ways people can, by routine or impulse, be real customers of music and not just passive consumers (or worse, what the industry used to call pirates). Let’s also create new ways, beyond payments alone, that artists and music lovers can signal each other, and have all kinds of creative fun.

The time is right. Let’s not let the opportunity pass. ■

Note: Jukebox image by Davide Cavalli [CC BY 3.0 (<https://creativecommons.org/licenses/by/3.0>)], via Wikimedia Commons.

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljournal@linuxjournal.com.

LETTERS

My Project Using Project Fi Data-Only SIMs

I've been a *Linux Journal* reader for years. Thanks to Shawn Powers for his [recent article on Project Fi](#) in the July 2018 issue. I thought he might be interested in my project using them: <https://journal.highlandsolutions.com/my-car-has-an-email-address-b5443afc84ec>.

—Mark Snyder

Shawn Powers replies: Mark, that's a really cool idea! We keep track of each other with a combination of Automatic and Life360, but the idea of a data-only connection to the car is genius!

More Regex

Thanks to Shawn Powers for his [article on globbing and regex](#) in the September 2018 issue. I enjoyed reading it and learned a few things. Please do write more about regex. Thanks again.

—Steve

Shawn Powers replies: Steve, I'll be honest, RegEx still blows my mind. I force myself to stay familiar with the basics so I can use it when needed, but anything more advanced usually requires me googling and head-scratching. If you become a RegEx superhero, perhaps consider contributing an article!

SOLID, Inrupt and PODs

I just leaned about these topics from a *Fast Company* article on Tim Berners-Lee and his startup Inrupt.

Can you please publish more about this in *LJ*?

I'm kinda surprised that you haven't presented anything yourself, given that my interpretation that some of your editorials are in high alignment with the concepts put forth of my currently cursory understanding of SOLID and POD.

I look forward to *LJ* contributors shining a light on this most hopeful technology news, especially given the skewering of Democracy we're going through at the hands of huge corporations taking over more and more of our lives and OUR government.

—Greg

Doc Searls replies: Thanks, Greg.

As you may know, I've led [ProjectVRM](#) at Harvard's [Berkman Klein Center](#) for the past 12 years, encouraging exactly the kind of things Tim and friends hope to make happen with Solid and Inrupt. In fact, both Solid and Inrupt are listed on [ProjectVRM's Developments page](#), along with many dozens of other developers doing very similar work.

Since I want to encourage all of those developers, I've gone out of my way not to favor any of them. But I'm not the only writer here at *Linux Journal*, and now is a good time for somebody to take a look at the whole list and do a summary report. So stay tuned for that, and thanks again!

Re: Amazon Pricing

I just read [Doc Searls' article on Amazon pricing](#) in the October 2018 issue, and I too have noticed that the pricing seems arbitrary and seems to change at random. I frequently will look at something and go back later to find the price has increased or decreased slightly. I now write down the prices of items if I am going to look at them again later. This includes books, tools, electronics and other things.

I find something else that Amazon does even more annoying. I will try searching for something using a generic term, say "drill press". Amazon will present me with a list of drill presses that omits some of the leading manufacturers. The manufacturer I want won't even be listed in the list of manufacturers on the left side of the screen. However, if I type in the manufacturer I want or the manufacturer's model number, all of a sudden, the listing will change. I have also noticed that I have to be careful to specify the exact model number and version of something I am looking for, otherwise

LETTERS

Amazon will show me something older. I was searching for something a few weeks ago, and I knew there was a new version of the tool out, but Amazon wouldn't show it to me. Finally, I went to the manufacturer's website and got the exact model number and version for the product. When I put this into Amazon, the item I wanted finally appeared. I'm thinking that Amazon was trying to clear out its old inventory at my expense.

Amazon needs to start being a little more honest with its customers, or we will soon be shopping somewhere else.

Keep up the good work!

—Phil Hegge

More Comments on Doc's Amazon Pricing Editorial

I use amazon.com. Although I am living in Europe, I go back and forth between the US and Europe. Generally, I read in English, so I prefer amazon.com instead of amazon.fr for books, etc. I often look at Amazon from Europe, not only from the US.

Below are some comments that may explain, in part, why Doc Searls finds different prices for the same item at Amazon. My comments are all related to books (including the Kindle versions); I don't have enough experience with non-book purchases at Amazon to comment on those.

I have found that when I search amazon.com without logging in to Amazon, particularly when I am in Europe, I often get one price. Then, when I log in, the price has changed. I understand that this is because, when you are not logged in to Amazon, Amazon uses your IP address to figure out where you are. Book publishers/distributors have divided up the world, and, as I understand it, for English language books sold by Amazon in my part of Europe, those books are from British book publishers/distributors. On the other hand, when I log in to amazon.com, I am considered to be in the US, so the English language books are handled by US book publishers/distributors. There are clear differences in price for

LETTERS

the “same” book depending on the publisher/distributor (Amazon suggests that it is the publishers/distributors who set the prices), which may explain some of the differences in pricing that Doc Searls has seen. I know from previous editorials that Doc Searls travels quite a bit. Perhaps some of his test cases used different IP addresses associated with different publishers/distributors?

Depending on whether you use a VPN, you also may see some of this because the other end of the VPN connects to the internet in various countries. Therefore, the IP address that Amazon sees may be different from time to time. And, as I mentioned above, the book price may vary depending on IP address, because Amazon associates the IP address with different publishers/distributors. Note that I do not (yet) use a VPN, but I am hypothesizing about the effect of using a VPN vis-à-vis Amazon.

The difference between being logged in and not being logged in also affects availability. I don't recall the precise example, but there was one time when the Kindle version of the book I wanted was available when I was not logged in. But when I logged in, the Kindle version was not available. Again, a difference between publishers/distributors.

One situation where I have seen the price change for a given book is when I am not logged in, using the same browser, but I look at it on different days. For example, if there is a book I might buy, I may hold it in a tab in my browser for quite some time (even weeks). Then, when I return to this tab, perhaps ready to buy, and I click on the “refresh” button, the screen is updated, which sometimes (not always) results in a change in price. This change (increase or decrease) is simply a function of the date because I am using the same browser and the same IP address.

These are some suggested explanations for why Doc Searls may have seen different book prices with Amazon. However, I don't mean to suggest that all of the differences mentioned in Doc's editorial are related to the reasons in my comments. It is quite possible that Doc's explanations for the differences also cover many cases.

—J.

LETTERS

Doc Searls replies: Thanks, J. Yours is an interesting and fully helpful case in point—or several points, all well made and taken.

What matters most is that Amazon remains, like so many other large and proprietary systems we depend on, a mess of opacities, on purpose. Thus, in spite of all the good Amazon does, one can't help sensing we are each being taken advantage of, in ways we'll never know, much of the time.

I believe the only solution to this problem in the long run is to increase agency—the power to act with full effect in the world—on the individual's side, in ways that are standard for all of us. In fact, aside from my work here at *Linux Journal*, that's my main mission in life.

System76

I just read [Rob Hansen's review of the System76 Onyx Pro laptop](#) in the October 2018 issue. I own a Meerkat desktop system from System76. It's based on Intel's NUC. I'm running POP_OS after running Ubuntu.

My Linux box is a workhorse. It takes whatever I throw at it with elan. I've been mostly Linux since 2004, and this version is really sweet. I'm a middle school computer teacher, so that's all Chromebooks. Accordingly, I have the third iteration of the Pixelbook. It is also a workhorse. Over time, I've realized that the OS doesn't matter as long as I can do what I want to do. Linux just makes that more probable.

—Ron Smith

Rob Hansen replies: With respect to “doing what you want to do”, I'm overjoyed at how much things have changed since I first started using Linux in 1996. I can honestly say that not even work has me tied to Windows anymore. It used to be that I was dependent on Windows for Skype to talk to remote team members, Outlook for syncing my calendar with my office, and many more. In the last five years, all of those have migrated to web applications or desktop versions have been released. It's

a great time to be a Linux user. The age of the Linux desktop never really arrived, but the long dark night of Windows dependency seems to be at an end. Viva freedom, viva choice!

From Social Media

In response to Zack Brown's "Linus' Behavior and the Kernel Development Community"

From commenter Dr Richard Stallman, President, Free Software Foundation, Internet Hall-of-Famer, MacArthur Fellow:

"Trolling" means making insincere statements to get a rise out of people. When I asked Linux developers to use the term "GNU/Linux" to refer to the GNU system with Linux as the kernel, I did so because I believe that is what fairness calls for. I think that the GNU project, which started development of this system, and contributed the largest portion, deserves equal mention.

The developers of the kernel, Linux, may have felt annoyed that I asked for fair treatment, but that doesn't make it "trolling".

See <https://gnu.org/gnu/linux-a...> and <https://gnu.org/gnu/gnu-lin...>, plus the history in <https://gnu.org/gnu/the-gnu...>

In response to Glyn Moody's "Now Is the Time to Start Planning for the Post-Android World"

From commenter Marlock:

I read the whole article expecting Purism to be mentioned, they are doing some serious work and upstreaming it all so it doesn't just vanish as vaporware, dies out of an app ecosystem or agonizes for lack of trustworthy code like so many before it did without much of a legacy.

LETTERS

Their approach seems much more sustainable than Canonical's Ubuntu on Android or the Ubuntu phone did, plus it's easily more trustworthy than Samsung's Tizen (Tizen is not a high standard), and then their PR is actually transparency, not just PR, so to me, they are looking great for the not too distant future.

And they have other hardware sales businesses going to keep them from running out of cash all of a sudden.

Author Glyn Moody responds:

It actually was supposed to be mentioned, but the sentence was left out somehow. It's now been added back.

From commenter Gwen Lynn:

I like **eelo's** approach to bringing a de-Googled mobile environment to the market. They provide both the smartphone ROM and web services, without any piece of Google inside!

The project has started only nine months ago with a successful yet modest Kickstarter campaign, but they already have a first beta available for 20 devices, with online drive (and automatic syncing from the ROM), calendar, email, etc. All this with a single eelo identity and the promise that in further versions, you can self-host your data.

Of course they are doing all the work by forking existing software (lineageos, nextcloud...), but they are doing really a great and credible job on the UI consistency, and they provide everything as a 100% open-source solution that could eventually go to the mass market in my opinion.

What's still missing though is an application installer because they are not ready yet with it, but at least it seems they won't make the same error as FirefoxOS or other "pure" platforms where you cannot install existing Android apps.

LETTERS

I'm super excited about this project, really.

In response to Kyle Rankin's "Raspberry Pi Alternatives"

Nick Danger @niqdanger:

i now have 3 odroids and 2 raspis as my home network. I blame @kylerankin for this. At least they are all quiet!

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.*

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

What's Your System's Uptime?

Keep track of your system's uptime and downtime with the **tuptime** tool.

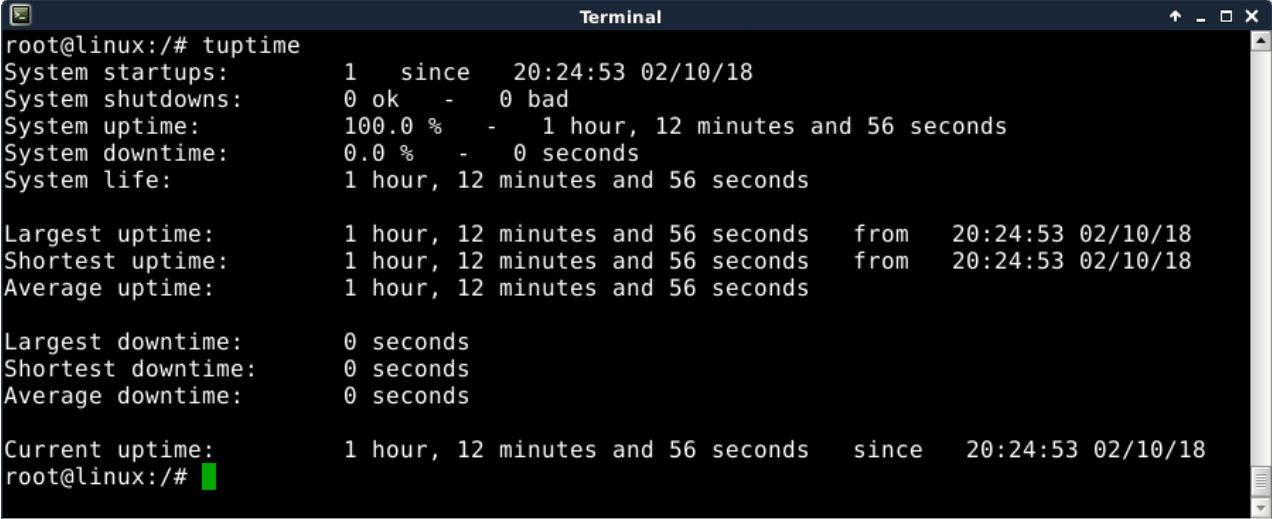
Finding your system's uptime is easy if the “beginning” means the last startup; the historical **uptime** command reports that information. But what happens if by “beginning” you mean the first startup ever of the system? Or the last 365 days? Or the last month?

Is there any way to have an accumulated uptime—or even better, a look at the whole system's life? For example, cars have odometers, and you can see the miles/kilometers since the first day. For computers, a tool was developed exactly for this task: **tuptime**.

tuptime reports the historical and statistical running and stopped time of your system, keeping track between restarts. Its main goals are:

- Count system startups.
- Register the first boot time (since installation).
- Count intended and accidental shutdowns.
- Show the uptime and downtime percentage since the first boot time.
- Show the accumulated system uptime, downtime and total.

UPFRONT

A terminal window titled "Terminal" showing the output of the 'tuptime' command. The output is as follows:

```
root@linux:/# tuptime
System startups:      1   since   20:24:53 02/10/18
System shutdowns:   0 ok    -   0 bad
System uptime:      100.0 % -   1 hour, 12 minutes and 56 seconds
System downtime:    0.0 %  -   0 seconds
System life:        1 hour, 12 minutes and 56 seconds

Largest uptime:     1 hour, 12 minutes and 56 seconds   from   20:24:53 02/10/18
Shortest uptime:    1 hour, 12 minutes and 56 seconds   from   20:24:53 02/10/18
Average uptime:     1 hour, 12 minutes and 56 seconds

Largest downtime:   0 seconds
Shortest downtime:  0 seconds
Average downtime:   0 seconds

Current uptime:     1 hour, 12 minutes and 56 seconds   since   20:24:53 02/10/18
root@linux:/# █
```

Figure 1. Example `tuptime` Execution after Installation

- Show the longest, shortest and average uptime and downtime.
- Show the current uptime.
- Print a formatted table or list with most of the previous values.
- Register used kernels.
- Create reports since and/or until a given startup or timestamp.
- Create reports in CSV format.

It works very simply. `tuptime` falls to the init manager for execution at startup and shutdown, and then into a cron task that launches regular executions in the meantime—there isn't any `dæmon` to worry about. Internally, it looks at the `btime` value (available in `/proc/stat`) and the `uptime` value (from `/proc/uptime`), and that's basically it.

The installation process is easy in Debian, Ubuntu and derivative distributions,

UPFRONT

using their respective package managers, and it should be available in all the official repositories. As prerequisites, it needs Python 3 and the SQLite library, which usually are included in core packages by default.

Once it's available on your system, you can get the information. It has three output formats: the default is a summary, and there also are table and list outputs to print the registered behavior.

The first execution reports the time since the system was booted, and the lines are self-explanatory (note that the date format is based on the system's locale settings):

```
$ tuptime
System startups:    1   since   22:21:49 02/02/18
System shutdowns:  0 ok    -    0 bad
System uptime:     100.0 %   -    40 minutes and 22 seconds
System downtime:   0.0 %    -    0 seconds
System life:       40 minutes and 22 seconds

Largest uptime:    40 minutes and 22 seconds   from
↳22:21:49 02/02/18
Shortest uptime:   40 minutes and 22 seconds   from
↳22:21:49 02/02/18
Average uptime:    40 minutes and 22 seconds

Largest downtime:  0 seconds
Shortest downtime: 0 seconds
Average downtime:  0 seconds

Current uptime:    40 minutes and 22 seconds   since
↳22:21:49 02/02/18
```

UPFRONT

When getting this report from an older system (see below), the information becomes more interesting. Apart from the fact that the counts increase, there also are more facts about the behavior. For example, the **System shutdowns** line has 11 “bads”, reflecting that the shutdown process wasn’t executed correctly, maybe due to power failure or system hangs. The percentage of uptime and downtime reflects that this report is from a lightly used system:

```
$ tuptime
```

```
System startups:   688   since   22:21:49 09/10/15
System shutdowns: 676 ok    <-    11 bad
System uptime:    4.6 %   -    40 days, 7 hours, 7 minutes
↳and 48 seconds
System downtime:  95.4 %   -    2 years, 105 days, 17 hours,
↳19 minutes and 25 seconds
System life:      2 years, 146 days, 0 hours, 27 minutes
↳and 13 seconds
```

```
Largest uptime:   12 hours, 51 minutes and 48 seconds   from
↳09:29:18 02/03/16
Shortest uptime:  5 seconds   from   22:20:54 12/02/17
Average uptime:   1 hour, 24 minutes and 21 seconds
```

```
Largest downtime: 23 days, 3 hours, 23 minutes and 30 seconds
↳from 13:49:42 04/12/16
Shortest downtime: 8 seconds   from   17:08:00 03/01/17
Average downtime: 1 day, 5 hours, 11 minutes and 44 seconds
```

```
Current uptime:   1 hour, 50 minutes and 0 seconds   since
↳17:37:32 02/07/18
```

UPFRONT

You can change the report to a table format (**-t**) used in combination with any other options, in this case, since the last two startups (**-S -2**):

```
$ tuptime -t -S -2
687  14:07:36 02/04/18   1 hour, 28 minutes and 22 seconds
↳15:35:58 02/04/18   OK   3 days, 2 hours, 1 minute
↳and 34 seconds
688  17:37:32 02/07/18   1 hour, 26 minutes and 13 seconds
```

Or you can change to the list report format (**-l**) and show the results until the second startup (**-U 2**):

```
$ tuptime -l -U 2
Startup:  1  at  22:21:49 09/10/15
Uptime:   50 minutes and 44 seconds
Shutdown: OK  at  23:12:33 09/10/15
Downtime: 13 seconds

Startup:  2  at  23:12:46 09/10/15
Uptime:   1 minute and 2 seconds
Shutdown: OK  at  23:13:48 09/10/15
Downtime: 18 hours, 57 minutes and 18 seconds
```

tuptime also accepts ranges between specific dates using the **tsince** and **tuntil** options. Both need an argument with the epoch date in seconds. Another example is a report from the last 365 days until the present, maybe to check your provider's SLA.

First, get the epoch date of one year ago using the **date** command:

```
$ date --date="1 year ago" +%s
1486490845
```

Next, pass it under `tsince` to `tuptime`:

```
$ tuptime --tsince 1486490845
```

Here's an example of a report from the first day to the last day of the previous month in CSV format. Again, use `date` to get the first and last days as a timestamp and pass both to `tuptime`:

```
$ date -d "-1 month 00:00" +%s
1514761200
$ date -d "this month -1 second 00:00" +%s
1517439599
$ tuptime --tsince 1514761200 --tuntil 1517439599 --csv
```

Or you can list all the entries (`-l`) ordered by the uptime (`-o u`), instead of the startup number, in reverse order (`-r`), including the kernel that was running (`-k`):

```
$ tuptime -l -o u -r -k
```

The manual has detailed information for every option and includes some interesting notes about sync date and time that can help in case of problems. For more information, see <https://github.com/rfrail3/tuptime.git>.

—Ricardo Fraile

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
- James Mayes
- James Weatherell
- Josh Simmons
- Mostly_Linux
- NDCHost.com
- Black Baron
- Magnus Magicman
- Robert J. Hansen

 **BECOME A PATRON**

Getting Started with Scilab

Scilab is meant to be an overall package for numerical science, along the lines of Maple, Matlab or Mathematica. Although a lot of built-in functionality exists for all sorts of scientific computations, Scilab also includes its own programming language, which allows you to use that functionality to its utmost. If you prefer, you instead can use this language to extend Scilab's functionality into completely new areas of research. Some of the functionality includes 2D and 3D visualization and optimization tools, as well as statistical functions. Also included in Scilab is Xcos, an editor for designing dynamical systems models.

Several options exist for installing Scilab on your system. Most package management systems should have one or more packages available for Scilab, which also will install several support packages. Or, you simply can download and install a tarball that contains everything you need to be able to run Scilab on your system.

Once it's installed, start the GUI version of Scilab with the **scilab** command. If you installed Scilab via tarball, this command will be located in the bin subdirectory where you unpacked the tarball.

When it first starts, you should see a full workspace created for your project.

On the left-hand side is a file browser where you can see data files and Scilab scripts. The right-hand side has several panes. The top pane is a variable browser, where you can see what currently exists within the workspace. The middle pane contains a list of commands within that workspace, and the bottom pane has a news feed of Scilab-related news. The center of the workspace is the actual Scilab console where you can interact with the execution engine.

UPFRONT

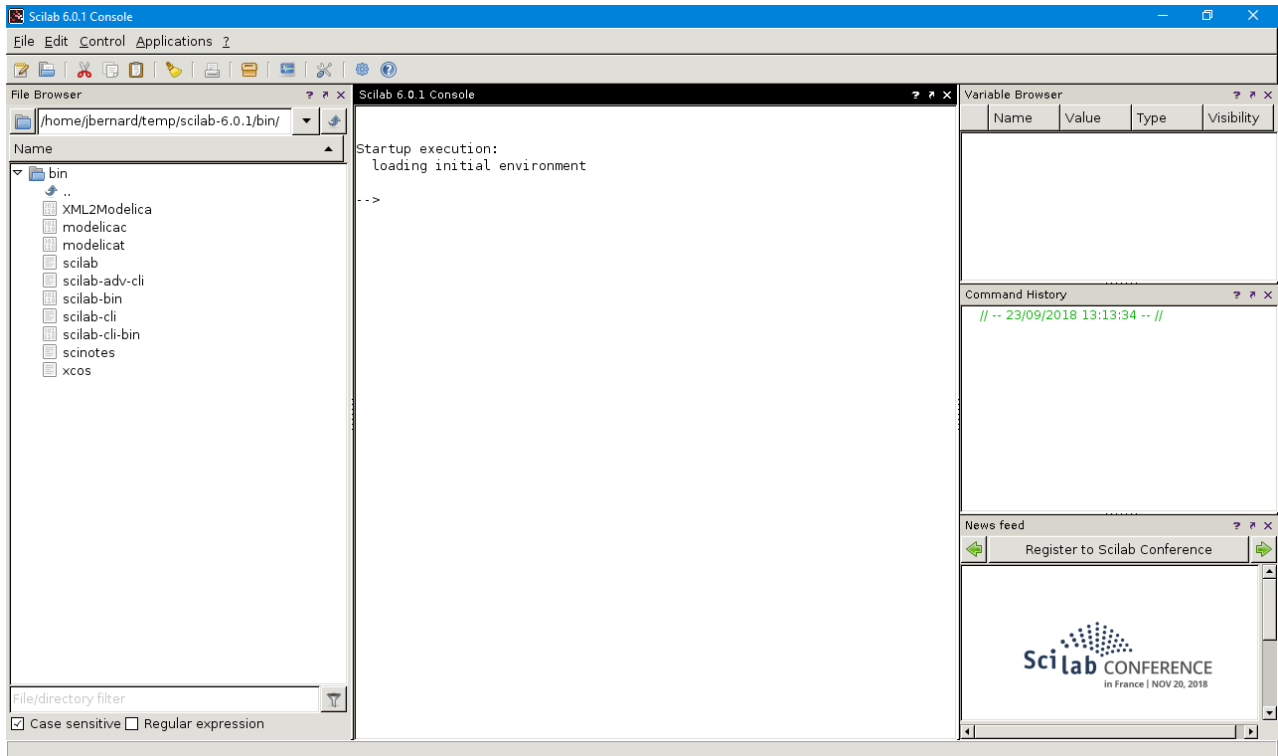


Figure 1. When you first start Scilab, you'll see an empty workspace ready for you to start a new project.

Let's start with some basic mathematics—for example, division:

```
--> 23/7  
ans =  
  
3.2857143
```

As you can see, the command prompt is `-->`, where you enter the next command to the execution engine. In the variable browser, you can see a new variable named **ans** that contains the results of the calculation.

Along with basic arithmetic, there is also a number of built-in functions. One thing to be aware of is that these function names are case-sensitive. For example, the statement `sqrt(9)` gives the answer of 3, whereas the statement `SQRT(9)`

UPFRONT

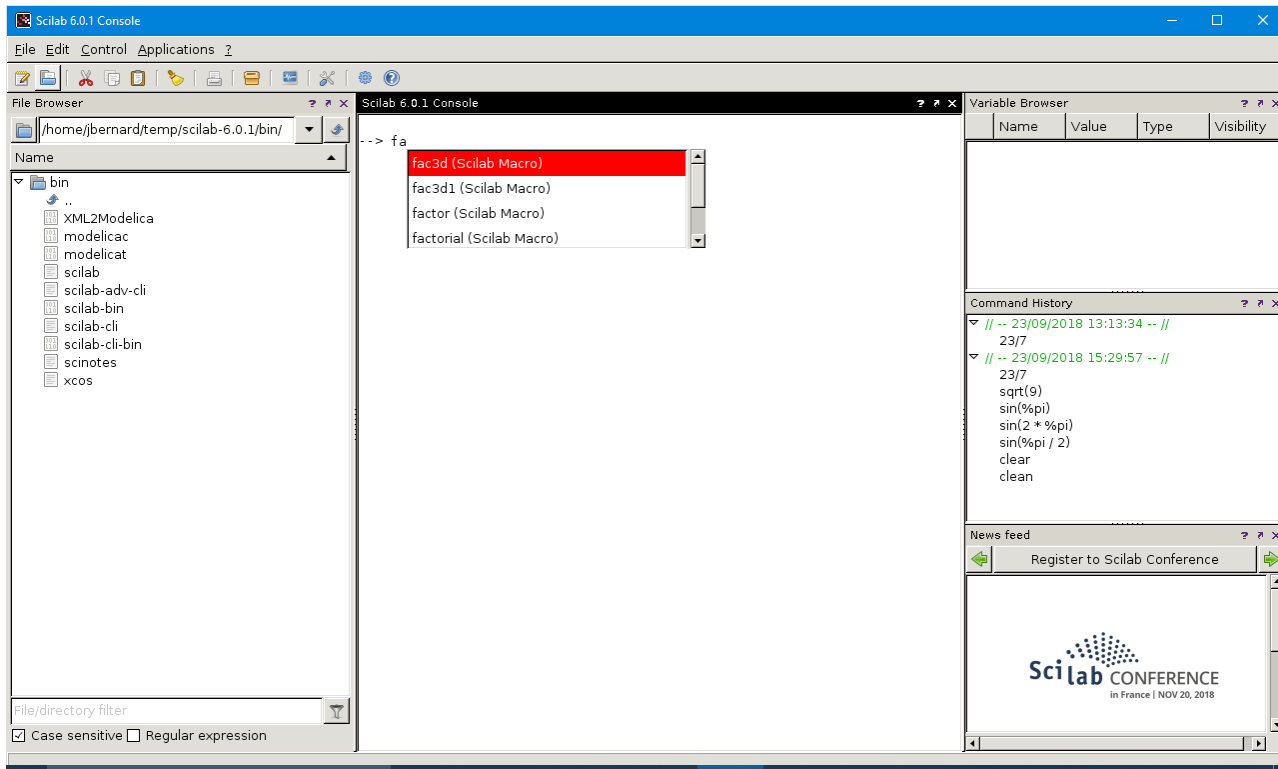


Figure 2. Use tab-completion to avoid typos while typing commands in the Scilab console.

returns an error.

There also are built-in constants for numbers like e or π . You can use them in statements, like this command to find the sine of $\pi/2$:

```
--> sin(%pi / 2)
ans =
```

1.

If you don't remember exactly what a function name is, but you remember how it starts, you can use the tab-completion functionality in the Scilab console. For example, you can see what functions start with "fa" by typing those two letters and then pressing the tab key.

You can assign variables with the “=” symbol. For example, assign your age to the **age** variable with:

```
--> age = 47
age =

47.
```

You then can access this variable directly:

```
--> age
age =

47.
```

The variable also will be visible in the variable browser pane. Accessing variables this way basically executes the variable, which is also why you can get extra output. If you want to see only the value, use the **disp()** function, which provides output like the following:

```
--> disp(age)

47.
```

Before moving onto more complex ideas, you’ll need to move out of the console. The advantage of the console is that statements are executed immediately. But, that’s also its disadvantage. To write larger pieces of code, you’ll want to use the included editor. Click the Applications→SciNotes menu item to open a new window where you can enter larger programs.

Once you’ve finished writing your code, you can run it either by clicking the run icon on the toolbar or selecting one of the options under the Execute menu item. When you do this, SciNotes will ask you to save your code to a file, with

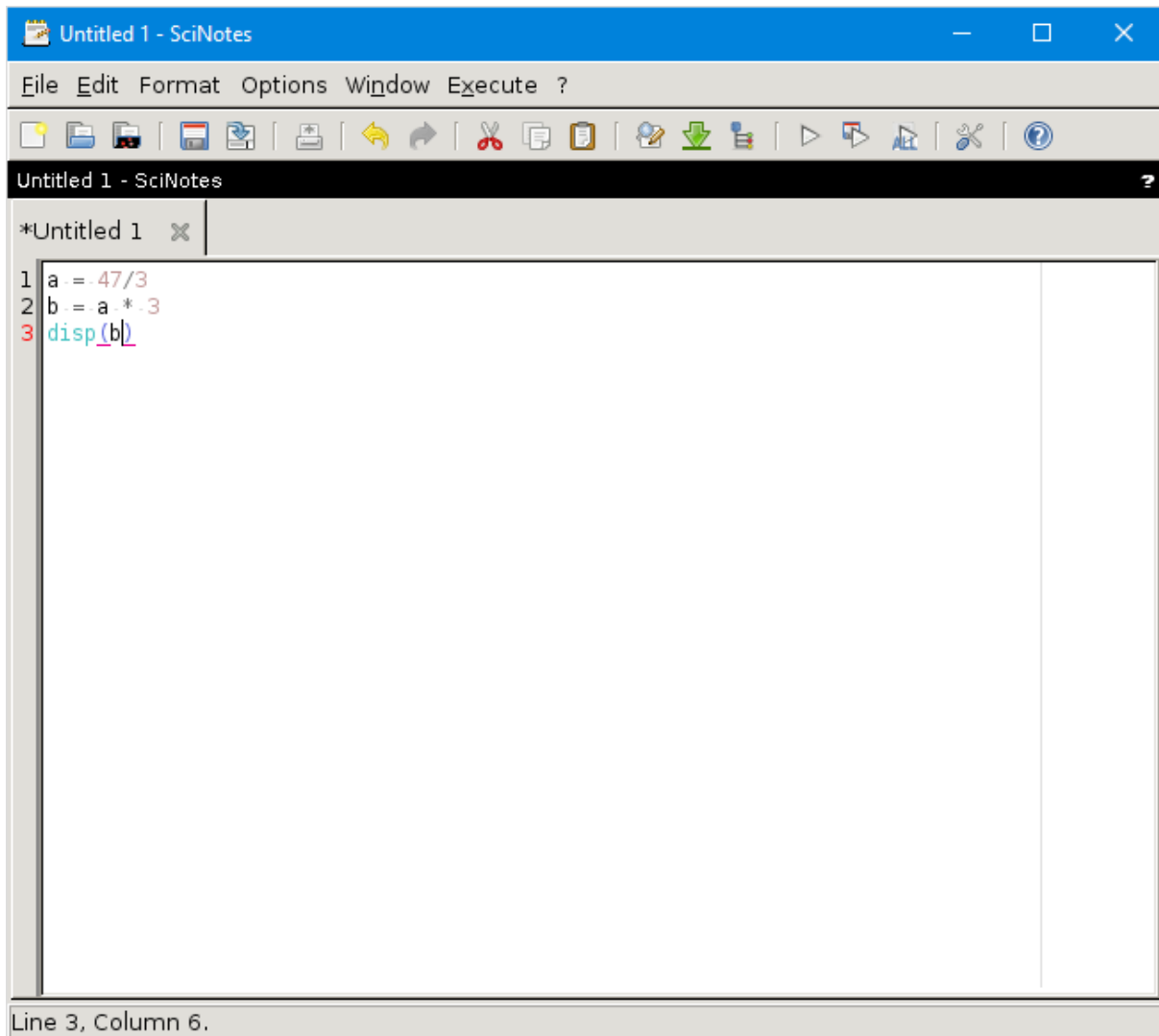


Figure 3. The SciNotes application lets you write larger programs and then run them within Scilab as a single unit.

the file ending “.sce”, before running. Then, it gets the console to run this file with the following command:

```
exec('/home/jbernard/temp/scilab-6.0.1/bin/test1.sce', -1)
```

If you create or receive a Scilab file outside of Scilab, you can run it yourself using a

similar command.

To build more complex calculations, you also need a way to make comparisons and loop over several calculations. Comparisons can be done with either:

```
if .... then
    stmts
end
```

or:

```
if .... then
    stmts
else
    stmts
end
```

or:

```
if .... then
    stmts
elseif .... then
    stmts
else
    stmts
end
```

As you can see, the **if** and **elseif** lines need to end with **then**. You can have as many **elseif** sections as you need for your particular case. Also, note that the entire comparison block needs to end with the **end** statement.

There also are two types of looping commands: **for** loops and **while** loops. As an example, you could use the following to find the square roots of the first

100 numbers:

```
for i=1:100
    a = sqrt(i) disp(a)
end
```

The **for** loop takes a sequence of numbers, defined by **start:end**, and each value is iteratively assigned to the dummy variable **i**. Then you have your code block within the **for** loop and close it with the statement **end**.

The **while** loop is similar, except it uses a comparison statement to decide when to exit the loop.

The last quick item I want to cover is the graphing functionality available within Scilab. You can create both 2D and 3D graphs, and you can plot data files or the results of functions. For example, the following plots the sine function from 0 to $\pi \times 4$:

```
t = linspace(0, 4 * %pi, 100) plot(t, sin(t))
```

You can use the **linspace** command to generate the list of values over which the function will be executed. The **plot** function opens a new window to display the resultant graph. Use the commands under the Edit menu item to change the plot's details before saving the results to an image file.

You can do 3D graphs just as simply. The following plots a parametric curve over 0 to $4 \times \pi$:

```
t=linspace(0,4*%pi,100); param3d(cos(t),sin(t),t)
```

This also opens a new plotting window to display the results. If the default view isn't appropriate, click Tools → 2D/3D Rotation, and with this selected, right-click on the graph and rotate it around for a better view of the result.

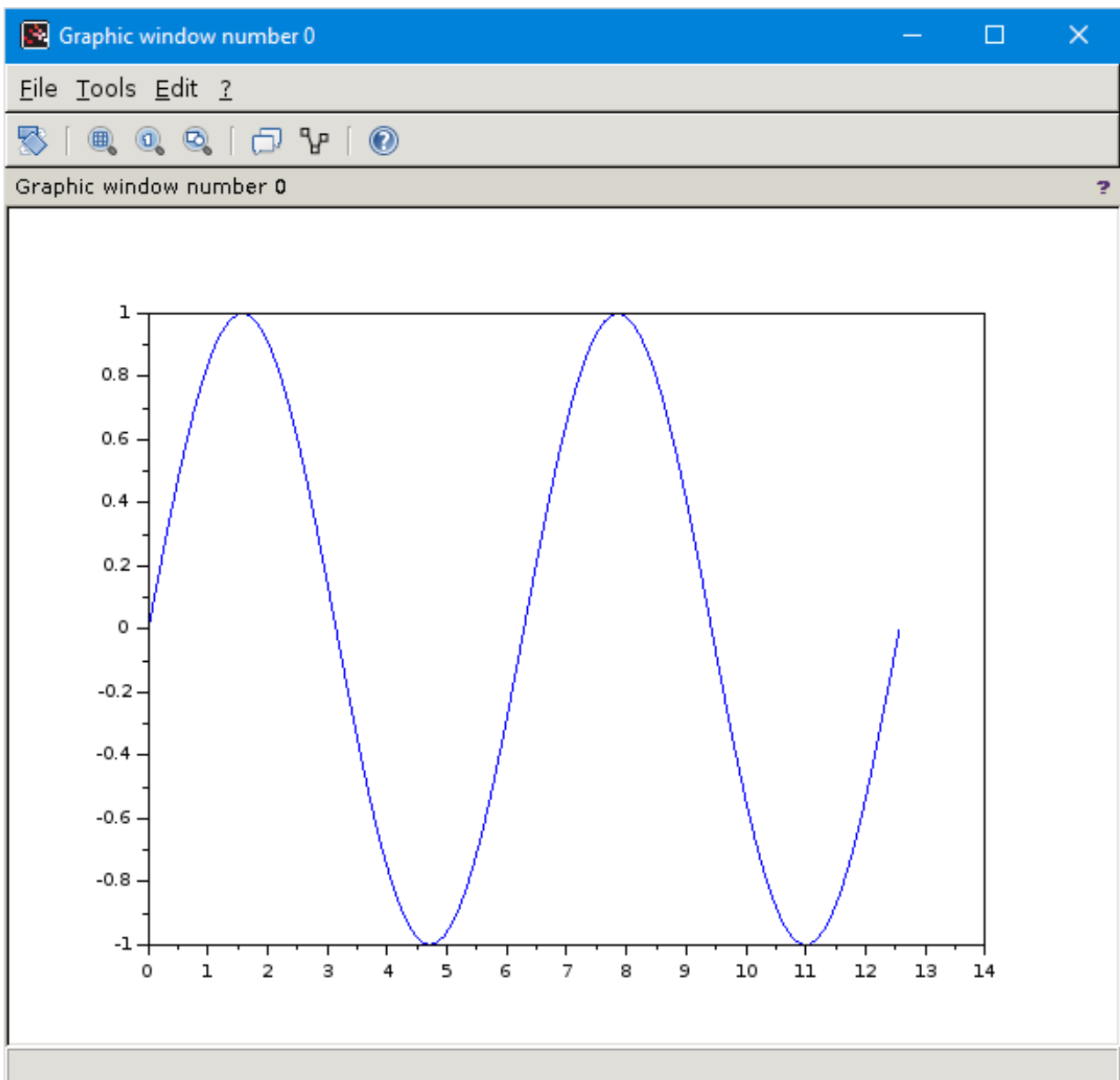


Figure 4. Calling the plot function opens a new viewing window where you can see the generated graphs.

Scilab is a very powerful tool for many types of computational science. Since it's available on Linux, macOS and Windows, it's a great option if you're collaborating with other people across multiple operating systems. It might also prove to be a effective tool to use in teaching environments, giving students access to a

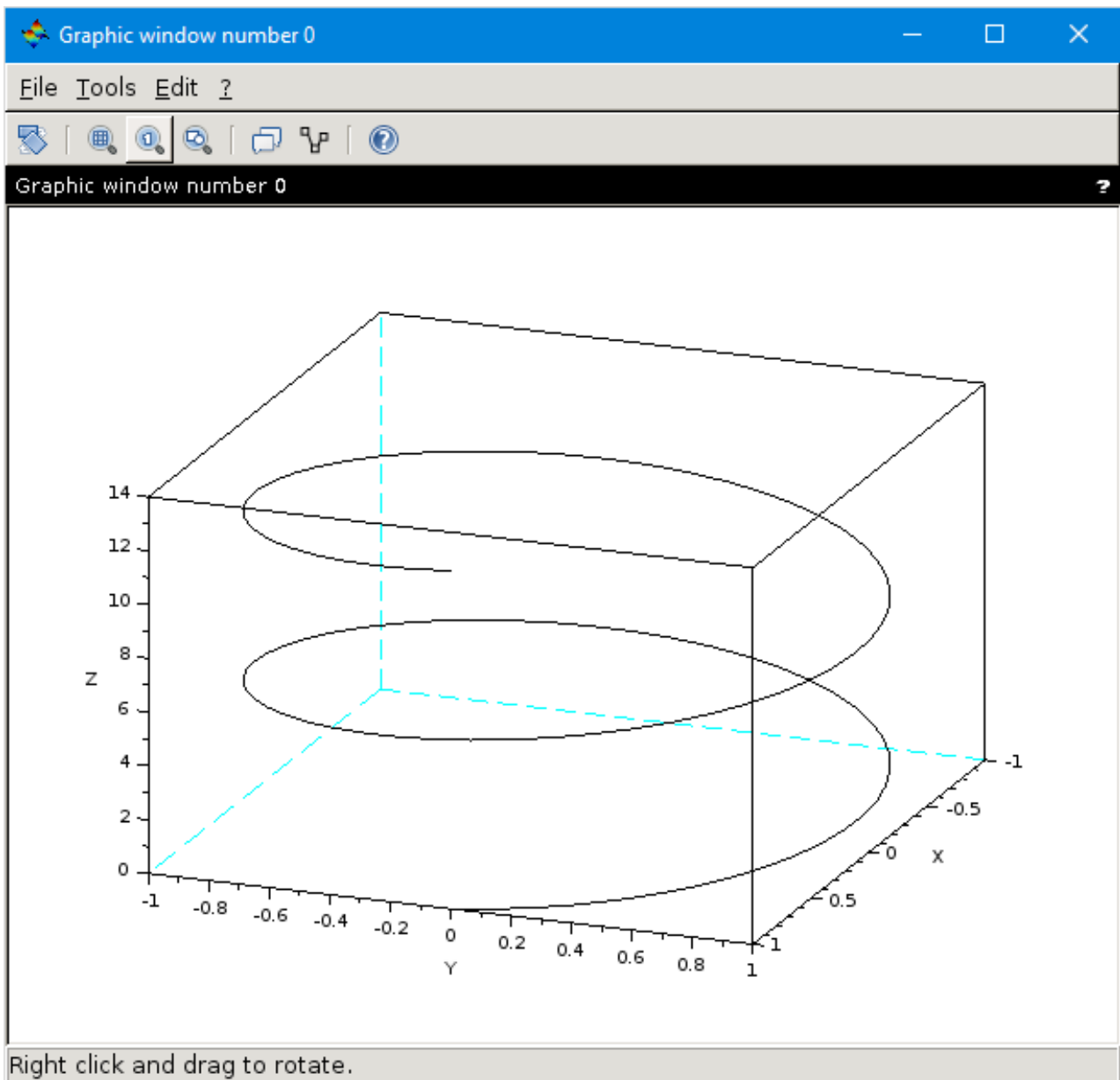


Figure 5. Generating 3D graphs is as easy as generating 2D plots, as this parametric curve example shows.

powerful computational platform for no cost, no matter what type of computer they are using. I hope this short article has provided some ideas of how it might be useful to you. I've barely covered the many capabilities available with Scilab, so be sure to visit the [main website](#) for a number of good tutorials.

FOSS Project Spotlight: BlueK8s

Deploying and managing complex stateful applications on Kubernetes.

Kubernetes (aka K8s) is now the de facto container orchestration framework. Like other popular open-source technologies, Kubernetes has amassed a considerable ecosystem of complementary tools to address everything from storage to security. And although it was first created for running **stateless applications**, more and more organizations are interested in using Kubernetes for **stateful applications**.

However, while Kubernetes has advanced significantly in many areas during the past couple years, there still are considerable gaps when it comes to running complex stateful applications. It remains challenging to deploy and manage distributed stateful applications consisting of a multitude of co-operating services (such as for use cases with large-scale analytics and machine learning) with Kubernetes.

I've been focused on this space for the past several years as a co-founder of **BlueData**. During that time, I've worked with many teams at Global 2000 enterprises in several industries to deploy distributed stateful services successfully, such as Hadoop, Spark, Kafka, Cassandra, TensorFlow and other analytics, data science, machine learning (ML) and deep learning (DL) tools in containerized environments.

In that time, I've learned what it takes to deploy complex stateful applications like these with containers while ensuring enterprise-grade security, reliability

and performance. Together with my colleagues at BlueData, we've broken new ground in using Docker containers for big data analytics, data science and ML/DL in highly distributed environments. We've developed new innovations to address requirements in areas like storage, security, networking, performance and lifecycle management.

Now we want to bring those innovations to the Open Source community to ensure that these stateful services are supported in the Kubernetes ecosystem. BlueData's engineering team has been busy working with Kubernetes, **developing prototypes** with Kubernetes in our labs and collaborating with multiple enterprise organizations to evaluate the opportunities (and challenges) in using Kubernetes for complex stateful applications.

To that end, we recently **introduced** a new Kubernetes open-source initiative: BlueK8s. The BlueK8s initiative will be composed of several open-source projects that each will bring enterprise-level capabilities for stateful applications to Kubernetes.

Kubernetes Director (or KubeDirector for short) is the first open-source project in this initiative. KubeDirector is a custom controller designed to simplify and streamline the packaging, deployment and management of complex distributed stateful applications for big data analytics and AI/ML/DL use cases.

Of course, other existing open-source projects address various requirements for both stateful and stateless applications. The Kubernetes **Operator** framework, for instance, manages the lifecycle of a particular application, providing a useful resource for building and deploying application-specific Operators. This is achieved through the creation of a simple finite state machine, commonly known as a reconciliation loop:

- *Observe*: determine the current state of the application.
- *Analyze*: compare the current state of the application with the expected state of the application.

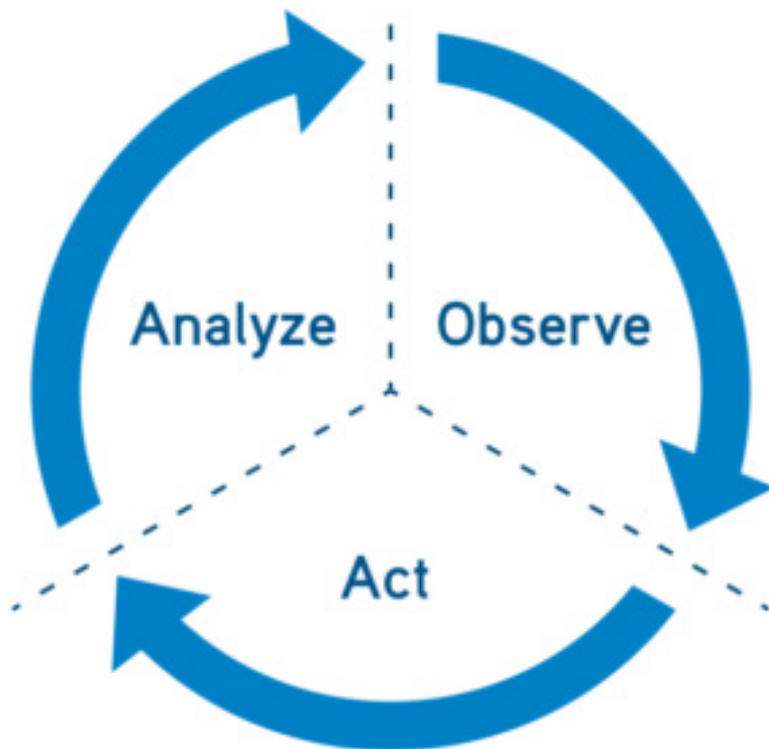


Figure 1.
Reconciliation Loop

- *Act*: take the necessary steps to make the running state of the application match its expected state.

It's pretty straightforward to use a Kubernetes Operator to manage a cloud native stateless application, but that's not the case for all applications. Most applications for big data analytics, data science and AI/ML/DL are not implemented in a cloud native architecture. And, these applications often are stateful. In addition, a distributed data pipeline generally consists of a variety of different services that all have different characteristics and configuration requirements.

As a result, you can't easily decompose these applications into self-sufficient and containerizable microservices. And, these applications are often a mishmash of tightly integrated processes with complex interdependencies, whose state is distributed across multiple configuration files. So it'd be challenging to create, deploy and integrate an application-specific Operator for each possible configuration.

The KubeDirector project is aimed at solving this very problem. Built upon the Kubernetes **custom resource definition** (CRD) framework, KubeDirector does the following:

- It employs the native Kubernetes API extensions, design philosophy and authentication.
- It requires a minimal learning curve for any developers that have experience with Kubernetes.
- It is not necessary to decompose an existing application to fit microservices patterns.
- It provides native support for preserving application configuration and state.
- It follows an application-agnostic deployment pattern, reducing the time to onboard stateful applications to Kubernetes.
- It is application-neutral, supporting many applications simultaneously via application-specific instructions specified in YAML format configuration files.
- It supports the management of distributed data pipelines consisting of multiple applications, such as Spark, Kafka, Hadoop, Cassandra, TensorFlow and so on, including a variety of related tools for data science, ML/DL, business intelligence, ETL, analytics and visualization.

KubeDirector makes it unnecessary to create and implement multiple Kubernetes Operators in order to manage a cluster composed of multiple complex stateful applications. You simply can use KubeDirector to manage the entire cluster. All communication with KubeDirector is performed via kubectl commands. The anticipated state of a cluster is submitted as a request to the API server and stored in the Kubernetes etcd database. KubeDirector will apply the necessary application-specific workflows to change the current state of the cluster into the expected state of the cluster. Different workflows can be specified for each application type, as

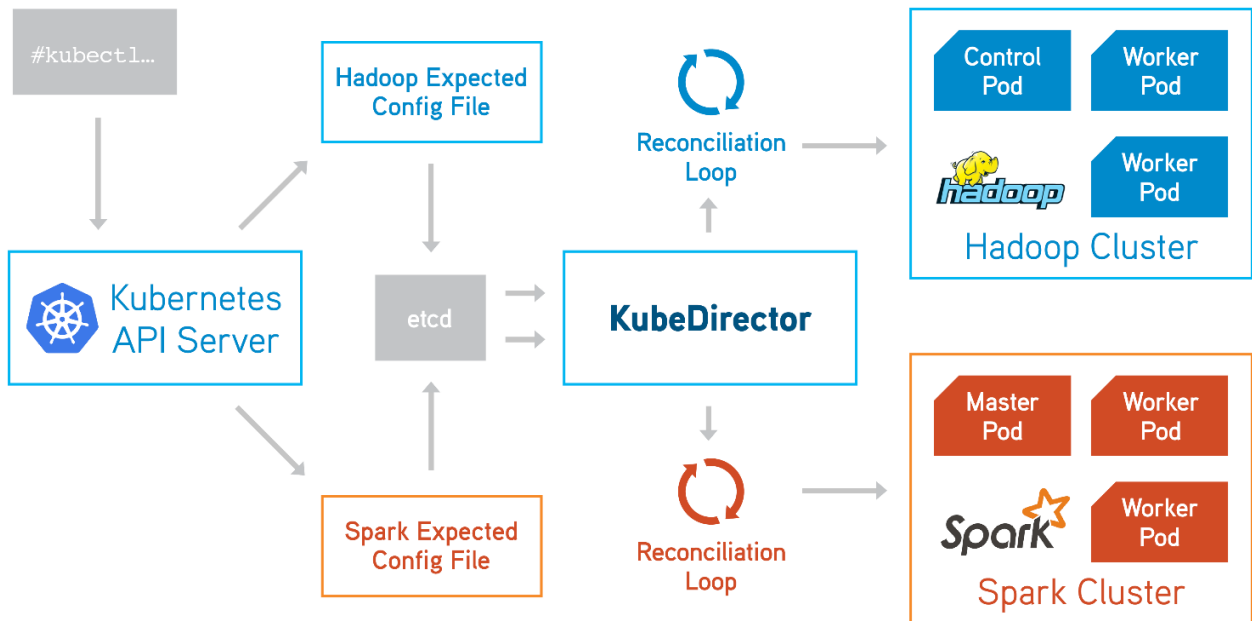


Figure 2. Using KubeDirector to Deploy and Manage Containerized Hadoop and Spark Application Clusters

illustrated in Figure 2, which shows a simple example (using KubeDirector to deploy and manage containerized Hadoop and Spark application clusters).

If you're interested, we'd love for you to join the growing community of KubeDirector developers, contributors and adopters. The initial pre-alpha version of KubeDirector was recently released at <https://github.com/bluek8s/kubedirector>. For an architecture overview, refer to the [GitHub project wiki](#). You can also read more about how it works in this [technical blog post on the Kubernetes site](#).

—Tom Phelan, co-founder and chief architect, BlueData

Lessons in Vendor Lock-in: Shaving

Learn how to embrace open standards while you remove stubble.

Freedom is powerful. When you start using free software, a whole world opens up to you, and you start viewing everything in a different light. You start noticing when vendors don't release their code or when they try to lock you in to their products with proprietary protocols. These vendor lock-in techniques aren't new or even unique to software. Companies long have tried to force customer loyalty with incompatible proprietary products that make you stay on an upgrade treadmill. Often you can apply these free software principles outside the software world, so in this article, I describe my own object lesson in vendor lock-in from the shaving industry.

When I first started shaving, I was pretty intimidated with the notion of a sharp blade against my face so I picked the easiest and least-intimidating route: electric razors. Of course, electric razors have a large up-front cost, and after some time, you have to buy replacement blades. Still, the shaves were acceptable as far as I knew, so I didn't mind much.

At some point in my shaving journey, Gillette released the Mach 3 disposable razor. For some reason, this design appealed to a lot of geeks, and I ended up hearing about it on geek-focused blogs like Slashdot back in the day. I decided to try it out, and after I got over the initial intimidation, I realized it really wasn't all that hard to shave with it, and due to the multiple blades and lubricating strip along the top, I got a much closer shave.

I was a convert. I ditched my electric razor and went all in with the Mach 3. Of course, those disposable blades had the tendency to wear out pretty quickly, along with that blue lubricating strip, so I'd find myself dropping a few bucks per blade to get refills after a few shaves. Then again, Gillette was famous for the concept of giving away the

razor and making its money on the blade, so this wasn't too surprising.

We're Going to Four Blades!

The tide started turning for me a few years later when Gillette decided to deprecate the Mach 3 in favor of a new design—this time with four blades, a lubricating strip *and* a rubber strip along the bottom! Everyone was supposed to switch over to this new and more expensive design, but I was perfectly happy with what I was using, and the new blades were incompatible with my Mach 3 razor, so I didn't pay it much attention.

The problem was that with this new design, replacement Mach 3 blades became harder and harder to come by, and all of the blades started creeping up in price. Eventually, I couldn't buy Mach 3 blades in bulk at my local warehouse store, and finally I gave up and bought one of the even more expensive new Gillette razors. What else could I do?

Learn from Your Elders

The turning point came when I read an article from The Art of Manliness blog called “Shave like your Grandpa” that described classic wet shaving techniques using big metal double-edged safety razors. I was fed up with the upgrade treadmill from proprietary plastic disposable razors, and I was intrigued at learning this lost art. More important, I found out that safety razors all used the same generic razor blade design and have for decades. As a result, you don't have to worry about incompatibility whether you find a safety razor in an antique store or buy one manufactured last week. Also, since every supplier is working off of the same open standard, blades are cheap—packs of five blades in a drug store are a dollar or two compared to three or four dollars per blade for modern disposables, and in bulk, they are less than ten cents apiece!

I was sold. I went to a local antique store and found a safety razor for about \$10. Next I got a cheap pack of razor blades from the drug store, and I was ready to go. Of course, there was a learning curve with this solution, just like when I switched from Windows to Linux so many years ago. Everything was different, and many things were more difficult to do at first. With Linux, this meant spending a lot of time reading documentation, figuring out commands and repairing systems I broke. With shaving,

this meant using a styptic pen to stop the bleeding!

Of course, in both cases, before too long, I climbed the learning curve and was getting superior results and had no intention of ever going back. If you've never tried wet shaving, you may not realize what this freedom from vendor lock-in means. It means if I see an interesting used safety razor in an antique store or if I see a nice new one online, I can get it without having to throw away all my existing razor blades. It also means if I find a new razor blade brand with a better price or better quality, I can switch over to it knowing my razor is automatically compatible.

Because there's no vendor lock-in and because the razor blade design is open, competition flourishes, and blade prices drop as a result, even among the higher-end vendors. I buy razor blades in packs of 100 online for less than \$10 knowing that I'm set for another two to three years. No more worrying about the constantly increasing prices of disposable blades or planned obsolescence from manufacturers to force me to some new design that adds a vibrating motor. Instead, I'm free to pick from a huge array of safety razor designs from many different companies both past and present. Just this last weekend, I was walking through an antique store and saw a 1961 Gillette "Fat Boy" adjustable safety razor that is well known and popular in safety razor collector's forums both for the unique design and for the great shave it can give. I bought it knowing that after cleaning it up, I could pop in a new blade from my collection, and it would just work.

The Legacy of Open Standards

The power of open standards extends beyond today into the future. When my son gets old enough to shave, I can pass down one of my all-metal, decades-old antique razors to him, and it will still work. While everyone else in a decade will have to shave with some \$20-per-blade disposable razor with three aloe strips, seven blades, and some weird vibrating and rotating motor, he will be able to pick any razor from my collection and find affordable replacement blades. This is the power of open standards and the freedom to avoid vendor lock-in.

—Kyle Rankin

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>.



Reality 2.0

Brought to
you by **LINUX**
JOURNAL

News Briefs

Visit LinuxJournal.com for daily news briefs.

- Linus Torvalds is taking a break. In an [rc4 email update](#), he writes about his scheduling mix-up with the kernel summit and having a “look yourself in the mirror moment”, and then (to summarize), he writes: “hey, I need to change some of my behavior, and I want to apologize to the people that my personal behavior hurt and possibly drove away from kernel development entirely. I am going to take time off and get some assistance on how to understand people’s emotions and respond appropriately.”
- Following Linus Torvalds’ apology for his behavior, the Linux Community has announced it will adopt a [“Code of Conduct”](#), which pledges to make “participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.”
- Purism launched the [Librem Key](#), the “first and only OpenPGP smart card providing a Heads-firmware-integrated tamper-evident boot process”. The Librem key is the size of an average thumb drive, allows you to keep your secret encryption keys in your pocket, and it alerts you if anyone tampers with your kernel or BIOS while you’re away from your laptop. The key works with all laptops but has extended features with Purism’s Librem laptop line. You can order one from [here](#) for \$59. See also [Kyle Rankin’s post](#) for more details on the Librem key.
- Yubico [announced the launch of the YubiKey 5 series](#), which are the first multi-protocol security keys to support FIDO2/WebAuthn and allow you to replace “weak password-based authentication with strong hardware-based authentication”. You can purchase them [here](#) for \$45.

- Google Chrome recently has begun automatically signing your browser in to your Google account for you every time you log in to a Google property, such as Gmail, without asking and without notification. See [Matthew Green's blog post](#) for more information on the huge privacy implications of this new practice.
- Mozilla [launched Firefox Monitor](#), a free service that alerts you if you've been part of a data breach. Enter your email at [Firefox Monitor](#) for a basic scan.
- [Tim Berners-Lee, creator of the world wide web, announced his new project Solid](#), “an open-source project to restore the power and agency of individuals on the web”. He writes “Solid changes the current model where users have to hand over personal data to digital giants in exchange for perceived value. As we've all discovered, this hasn't been in our best interests. Solid is how we evolve the web in order to restore balance by giving every one of us complete control over data, personal or not, in a revolutionary way.”
- Microsoft has joined the Open Invention Network (OIN), an open-source patent consortium. According to [ZDNet](#), this means “Microsoft has essentially agreed to grant a royalty-free and unrestricted license to its entire patent portfolio to all other OIN members.” OIN's CEO Keith Bergelt says “This is everything Microsoft has, and it covers everything related to older open-source technologies such as Android, the Linux kernel, and OpenStack; newer technologies such as LF Energy and HyperLedger, and their predecessor and successor versions.”
- The Libre Computer Project recently announced its new open-source, libre ARM SBC called La Frite. [Phoronix reports](#) the new 512MB model will ship for \$5 USD, or you can get the 1GB version for \$10 USD. In addition, “the \$5 ARM SBC is said to be 10x faster than the Raspberry Pi Zero” and also includes real HDMI, Ethernet and USB ports. La Frite, the miniature version of Le Potato SBC supported by mainline Linux and Android 8, should be available in November 2018. See the [Kickstarter page](#) for details.

- Canonical announced that Plex has arrived in its Snap Store. You now can [download the multimedia platform](#) as a snap for Ubuntu, KDE Neon, Debian, Fedora, Manjaro, OpenSUSE and Zorin. For more details, see the [Ubuntu Blog](#).
- A grey-hat hacker is breaking into MikroTik routers and patching them so they can't be compromised by cryptojackers or other attackers. [According to ZDNet](#), the hacker, who goes by Alexey, is a system administrator and claims to have disinfected more than 100,000 MikroTik routers. He told ZDNet that he added firewall rules to block access to the routers from outside the local network, and then “in the comments, I wrote information about the vulnerability and left the address of the @router_os Telegram channel, where it was possible for them to ask questions.” Evidently, a few folks have said “thanks”, but many are outraged.

Schedule One-Time Commands with the UNIX at Tool

Cron is nice and all, but don't forget about its cousin **at**.

By Kyle Rankin

When I first started using Linux, it was like being tossed into the deep end of the UNIX pool. You were expected to use the command line heavily along with all the standard utilities and services that came with your distribution. A lot has changed since then, and nowadays, you can use a standard Linux desktop without ever having to open a terminal or use old UNIX services. Even as a sysadmin, these days, you often are a few layers of abstraction above some of these core services.

I say all of this to point out that for us old-timers, it's easy to take for granted that everyone around us innately knows about all the command-line tools we use. Yet, even though I've been using Linux for 20 years, I still learn about new (to me) command-line tools all the time. In this "Back to Basics" article series, I plan to cover some of the command-line tools that those new to Linux may never have used before. For those of you who are more advanced, I'll spread out this series, so you can expect future articles to be more technical. In this article, I describe how to use the **at** utility to schedule jobs to run at a later date.



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.

at vs. Cron

at is one of those commands that isn't discussed very much. When people talk about scheduling commands, typically cron gets the most coverage. Cron allows you to schedule commands to be run on a periodic basis. With cron, you can run a command as frequently as every minute or as seldom as once a day, week, month or even year. You also can define more sophisticated rules, so commands run, for example, every five minutes, every weekday, every other hour and many other combinations. System administrators sometimes will use cron to schedule a local script to collect metrics every minute or to schedule backups.

On the other hand, although the **at** command also allows you to schedule commands, it serves a completely different purpose from cron. While cron lets you schedule commands to run periodically, **at** lets you schedule commands that run only once at a particular time in the future. This means that **at** fills a different and usually more immediate need from cron.

Using at

At one point, the **at** command came standard on most Linux distributions, but these days, even on servers, you may find yourself having to install the **at** package explicitly. Once installed, the easiest way to use **at** is to type it on the command line followed by the time you want the command to run:

```
$ at 18:00
```

The **at** command also can accept a number of different time formats. For instance, it understands AM and PM as well as words like “tomorrow”, so you could replace the above command with the identical:

```
$ at 6pm
```

And, if you want to run the same command at that time tomorrow instead:

```
$ at 6pm tomorrow
```


HACK AND /

Once you press enter, you'll drop into an interactive shell:

```
$ at 6pm tomorrow
warning: commands will be executed using /bin/sh
at>
```

From the interactive shell, you can enter the command you want to run at that time. If you want to run multiple commands, press enter after each command and type the command on the new **at>** prompt. Once you're done entering commands, press Ctrl-D on an empty **at>** prompt to exit the interactive shell.

For instance, let's say I've noticed that a particular server has had problems the past two days at 5:10am for around five minutes, and so far, I'm not seeing anything in the logs. Although I could just wake up early and log in to the server, instead I could write a short script that collects data from **ps**, **netstat**, **tcpdump** and other command-line tools for a few minutes, so when I wake up, I can go over the data it collected. Since this is a one-off, I don't want to schedule something with cron and risk forgetting about it and having it run every day, so this is how I would set it up with **at**:

```
$ at 5:09am tomorrow
warning: commands will be executed using /bin/sh
at>
at> /usr/local/bin/my_monitoring_script
```

Then I would press Ctrl-D, and the shell would exit with this output:

```
at> <EOT>
job 1 at Wed Sep 26 05:09:00 2018
```

Managing at Jobs

Once you have scheduled **at** jobs, it's useful to be able to pull up a list of all the **at** jobs in the queue, so you know what's running and when. The **atq** command lists the current **at** queue:

```
$ atq
1   Wed Sep 26 05:09:00 2018 a kyle
```

The first column lists the number `at` assigned to each job and then lists the time the job will be run and the user it will run as. Let's say that in the above example I realize I've made a mistake, because my script won't be able to run as a regular user. In that case, I would want to use the `atrm` command to remove job number 1:

```
$ atrm 1
```

If I were to run `atq` again, I would see that the job no longer exists. Then I could `sudo` up to root and use the `at` command to schedule the job again.

at One-Liners

Although `at` supports an interactive mode, you also can pipe commands to it all on one line instead. So, for instance, I could schedule the above job with:

```
$ echo /usr/local/bin/my_monitoring_script | at 5:09am tomorrow
```

Conclusion

If you didn't know that `at` existed, you might find yourself coming up with all sorts of complicated and convoluted ways to schedule a one-off job. Even worse, you might need to set an alarm clock so you can wake up extra early and log in to a problem server. Of course, if you don't have an alarm clock, you could use `at`:

```
$ echo "aplay /home/kyle/alarm.wav" | at 7am tomorrow
```



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

Testing Your Code with Python's `pytest`

Don't test your code? `pytest` removes any excuses.

By Reuven M. Lerner

Software developers don't just write software; they also use software. So, they're the first to recognize, and understand, that software is complex and inevitably contains bugs.

But, just because bugs are inevitable doesn't mean that developers can or should try to prevent them. And, thus, during the past few decades, there's been rapid growth in software testing. Testing is no longer seen as an optional or "nice to have" part of software development; it's considered an absolute must—part of the software development process. In many cases, the people in the Python courses I teach at various companies aren't developers *per se*, but instead testers—people with the full-time job of writing tests to ensure that the company's software is robust.

I must admit that even though I've been writing software for a long time, I have rarely been as good about testing as I'd like to be. Sure, when I'm working on a large, complex app, I'll write tests, but it always seemed to be a bit of a burden. I know that it's good for me, will save tons of time in the future, will make the software more robust and maintenance easier, but really, if I just want to get my program out the



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.

door, why test? And besides, the various test frameworks I've used through the years never struck me as very impressive or easy to use.

So for the past few years, I've been in a bit of a holding pattern. I want to test more, but testing is annoying, so I don't test, which makes it seem like even more of a burden, because it's not part of my regular process.

All of this has changed for me recently, thanks to my discovery (long after other people, I admit) of the `pytest` library for Python. `pytest` turns out to be easy to use, easy to work with and easy to integrate into my work. Part of the reason for this is that `pytest` abandons the Python idea of "there's only one way to do it", giving developers a great degree of flexibility and freedom in choosing how to write tests.

So in this article, I provide an introduction to `pytest`, showing how to start integrating it into your development process today. I plan to expand on this in my next article and describe some more advanced `pytest` features that you might need to use.

Basic pytest

The idea behind `pytest` is that if you want to test a function, you'll write a separate function to test it. Actually, you'll probably want to write more than one test function, but that's in addition.

For example, let's assume you have the following function that sums numbers:

```
def mysum(numbers):  
    output = 0  
    for one_number in numbers:  
        output += one_number  
    return output
```

How can you test this function? (And yes, I'm ignoring the "test-driven development" mode of testing, in which you first write the tests and then write the code. You certainly can do TDD with `pytest`, but that isn't my point right now.)

AT THE FORGE

I put this function definition in `mysum.py`. I next can create a file called `test_mysum.py` in the same directory. Then, when I run `pytest` in the current directory, it'll run all of the files starting with `test_`. How might `test_mysum.py` look? Let's start with something simple:

```
from mysum import mysum

def test_sum_integers():
    assert mysum([0,1,2,3,4]) == 10
```

As you can see, my test file `test_mysum.py` is fairly short. But it contains an actual test, and it also points to how tests can and will be written.

First, you have to import the file that you want to test. This can be a simple “import XYZ” statement, or you can import names selectively from the module with “from X import Y”. Either way, you'll need to import the functions and classes you'll be testing.

The tests themselves are written as Python functions whose names begin with `test_`. (Yes, this means that tests are written in files whose names begin with `test_`, and then with functions in those files whose names begin with `test_`.)

In simple cases, these test functions take no parameters. The functions are called by `pytest`, and the key to the tests is the `assert` statement. Normally, the `assert` statement in Python evaluates an expression. If the expression returns True, the assertion is recorded as a success, but otherwise ignored.

So in the case of these example test functions, I'm basically saying “if I call the function with one argument, the list `[0,1,2,3,4]`, I'm expecting to get the integer 10 back as a result”.

How do I run my test? I go into the directory where my files are located, and I type:

```
pytest
```

AT THE FORGE

Sure enough, pytest notices that there's a file matching the "test_*" pattern, which it runs. After some initial boilerplate indicating my system's configuration, I get the following output:

```
collected 1 item

test_mysum.py . [100%]

=====1 passed in 0.02 seconds=====
```

In other words, there was one file (test_mysum.py). It contained a single test function, represented by a dot (.). And, 100% of those tests ran successfully—meaning, what I asserted is indeed what was actually returned.

But of course, it's not enough to test with this sort of thing. I should probably call it with an empty list to make sure I get a 0 value back. So, let's add another test. Now test_mysum.py looks like this:

```
from mysum import mysum

def test_sum_integers():
    assert mysum([0,1,2,3,4]) == 10

def test_sum_nothing():
    assert mysum([]) == 0
```

And when I run the tests, I get:

```
collected 2 items

test_mysum.py .. [100%]

===== 2 passed in 0.10 seconds =====
```

Let's add another test to see what happens if I invoke it with some floating-point numbers:

```
from mysum import mysum

def test_sum_integers():
    assert mysum([0,1,2,3,4]) == 10

def test_sum_floats():
    assert mysum([0.1,1.2,2.3,3.4,4.5]) == 11.5

def test_sum_nothing():
    assert mysum([]) == 0
```

And now, when I test things, I get:

```
collected 3 items

test_mysum.py ... [100%]

===== 3 passed in 0.06 seconds =====
```

Sure enough, I've done a great job of testing so far.

I should note that while I've used only a single **assert** statement in each function here, you definitely can have more than one. I prefer to keep each test function as focused as possible, and thus, I use as few **assert** statements as I can.

Failing Tests

What if a test fails? Let's give it a shot by deliberately introducing a test that will fail. In `test_mysum.py`, I've added:

```
def test_one_and_one_are_three():
    assert mysum([1,1]) == 3
```

AT THE FORGE

When I run the tests, I get the following output:

```
test_mysum.py ...F [100%]

===== FAILURES =====
_____ test_one_and_one_are_three _____

    def test_one_and_one_are_three():
>         assert mysum([1,1]) == 3
E         assert 2 == 3
E         + where 2 = mysum([1, 1])

test_mysum.py:15: AssertionError
===== 1 failed, 3 passed in 0.30 seconds =====
```

First, you can see that four test ran in test_mysum.py. The first three ran successfully and were represented by dots. The fourth test failed though. “Failure” in this case means that the **assert** statement claimed that there would be one answer (3), but that running the function produced a different answer (2). pytest not only indicates that there was a failure, but it also indicates in which test function the error occurred and the line where it took place. This allows you to figure out where the problem lies.

In the case of failure, of course, there are two possibilities: the original code is wrong, or your test is wrong. Don’t forget that tests are code, which means that they can be prone to problems too! However, if you write your tests cleanly and clearly (and before or as you write the code), I’ve found that most tests will be simple and straightforward, making it less likely that the tests have problems and easier to identify the location of bugs.

You even can get more detailed output from pytest with the **-v** option:

```
test_mysum.py::test_sum_integers PASSED [ 25%]
test_mysum.py::test_sum_floats PASSED [ 50%]
test_mysum.py::test_sum_nothing PASSED [ 75%]
```



```
test_mysum.py::test_one_and_one_are_three FAILED [100%]
```

```
===== FAILURES =====
_____ test_one_and_one_are_three _____
```

```

    def test_one_and_one_are_three():
>     assert mysum([1,1]) == 3
E     assert 2 == 3
E         + where 2 = mysum([1, 1])
```

```
test_mysum.py:15: AssertionError
```

```
===== 1 failed, 3 passed in 0.22 seconds =====
```

Now you can see precisely which tests passed and failed, as well as where the failures took place.

Parametrized Tests

The successful tests created so far (`test_sum_nothing`, `test_sum_integers` and `test_sum_floats`) are all great and useful. But if you're like me, you might be wondering why you need three separate test functions just to check those three similar, but not identical, invocations. The pytest people agree, and they suggest the use of "parametrized tests". The idea here is that you define the test a single time, but tell pytest which inputs and outputs to provide.

You can do this by applying a Python decorator to the test function. The decorator will take two arguments: a string with comma-separated names representing the parameters you want to pass to the test and a list of two-element tuples describing the inputs and outputs. For example, given all of these tests:

```
def test_sum_integers():
    assert mysum([0,1,2,3,4]) == 10
```

```
def test_sum_floats():
```

AT THE FORGE

```
assert mysum([0.1,1.2,2.3,3.4,4.5]) == 11.5
```

```
def test_sum_nothing():  
    assert mysum([]) == 0
```

You can replace them all with a single test:

```
import pytest  
@pytest.mark.parametrize('numbers,output', [  
    ([], 0),  
    ([10, 20, 30], 60),  
    ([0.1, 1.2, 2.3, 3.4, 4.5], 11.5)])  
def test_mysum(numbers, output):  
    assert mysum(numbers) == output
```

While this does the same thing as before, it definitely looks a bit more complex. Let's break it down:

- First, you need to import **pytest**, so that you'll have access to the decorator.
- Then you use **@pytest.mark.parametrize** as the decorator. Note that if you are like me and prefer to spell it "parameterize", you'll get an error message scolding you for misspelling it.
- The first decorator argument is a string containing the comma-separated names of the variables you want to pass. You'll always need at least two: one for the input and one for the output. If your function takes two inputs, you'll need to define three parameters (two input and one output).
- The second argument is a list of two-element tuples. Each tuple describes a test run. Each tuple element will be assigned to a test function parameter.
- Finally, the test function now needs to take two parameters, with the same names

as defined in the decorator argument.

With this in place, you can now run your tests, and you'll get the following output:

```
test_mysum.py::test_mysum[numbers0-0] PASSED [33%]
test_mysum.py::test_mysum[numbers1-60] PASSED [66%]
test_mysum.py::test_mysum[numbers2-11.5] PASSED [100%]

===== 3 passed in 0.12 seconds =====
```

If you're thinking, "wow, that looks a lot like the output from the three separate tests"—well, that's exactly right.

Summary

There's much more to say about pytest, but what I've written here covers most of the cases you'll encounter in your day-to-day work. Next time, I plan to cover a few other topics, including how to deal with exceptions, user input and output, and checking the code coverage. ■

Resources

The pytest website is at <https://docs.pytest.org/en/latest>.

An excellent book on the subject is Brian Okken's *Python Testing with pytest*, published by Pragmatic Programmers. He also has many other resources, about pytest and code testing in general, [here](#).

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljournal@linuxjournal.com.

Roman Numerals and Bash

Fun with retro-coding a Roman numeral converter—I head back to my college years and solve me homework anew!

By Dave Taylor

I earned a bachelor's degree in computer science back in the dawn of computing. Well, maybe it wasn't quite that long ago, but we did talk about Ada and FORTRAN in class. As a UCSD alumnus, however, it's no surprise that UCSD Pascal was the programming language of choice. Don't worry; no punch cards and no paper tape were involved in my educational endeavors.

As with modern computer science study, we spent a lot of time coding algorithms and solving problems and puzzles. I'm a board-gamer, so I was quite happy to try to solve the "dining philosophers problem", the "four color problem" or the "traveling salesman problem". You might well have tried to solve the same darn problems.

One coding problem that has stuck with me is a Roman numeral conversion program. As part of my first programming class, I recall it being a pretty tricky problem, but we didn't have the internet or GitHub to scrounge around for smart solutions or inspiration.



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](#).

WORK THE SHELL

So, in the spirit of retro-coding, let's build a script that can convert Roman numerals into regular decimal equivalent values.

Roman Numerals

I know, you're saying "um, remind me, what are Roman numerals?", even though you see them all the time in movies and books. You just ignore the MCMLXIII that shows up as a copyright notice. What's funny is that the general industry consensus is that studios use those Roman numerals instead of the more understandable "Copyright 1963" to obfuscate the age of the film (by the way, MCMLXIII = 1963).

It turns out that Roman numerals are interesting because they are essentially grouped into logical segments. At its most basic, each letter has a specific decimal value, so let's start there (see Table 1 for the values).

Table 1. Roman Numerals and Their Values

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

If you wanted to write the value 60 as Roman numerals, that's easy: LX. Reverse the two values, however, and it's a completely different value: XL = 40. Why? Because when a lower value symbol appears before a higher value symbol, it's considered a reduction of that value. The fancy name for this is *subtractive notation*.

In other words, LX = 50 + 10, but XL = L - X = 50 - 10.

Now you can see how the earlier value breaks into clusters of values based on whether a subsequent value is higher or lower than the current value. Here's the logic:

$$\text{MCMLXIII} = \text{M} + \text{CM} + \text{L} + \text{X} + \text{III} = 1000 + 900 + 50 + 10 + 3$$

The general rule involves a single character look-ahead. That is, the value 4 is

WORK THE SHELL

represented as IV (literally 5 – 1), so while 8 is VIII, the value 9 is IX. Fortunately, 8 will never be IIX because that violates the single value subtraction, and the values always are adjacent so you won't see IM or VC.

Got it? It's pretty easy, really, once you know the letter values and the subtractive notation.

Writing a Roman Numeral Converter

Let's get to some coding.

The first challenge is to figure out how to convert individual Roman numeral values to their decimal equivalent. We humans can just glance at the table presented earlier and remember that L = 50, but we've got to teach the shell that trick too.

This would be a perfect use for a two-dimensional array where the primary index would be a character value, and the equivalent secondary value would be the corresponding decimal value. Alas, one of the weaknesses of Bash is that it doesn't support multi-dimensional arrays.

I'm going to be lazy and just utilize a function with a case statement. This also lets me support both uppercase and lowercase values:

```
mapit() {
case $1 in
I|i) equiv=1 ;;
V|v) equiv=5 ;;
X|x) equiv=10 ;;
L|l) equiv=50 ;;
C|c) equiv=100 ;;
D|d) equiv=500 ;;
M|m) equiv=1000 ;;
* ) echo "Error: Value $1 unknown" >&2 ; exit 2 ;;
esac
}
```

WORK THE SHELL

Recall that in Bash you can't have a function return a value, but you can send parameters to the function. They are then accessible within the function as positional parameters \$1, \$2, \$3 and so on. Ergo **\$1** is the letter being mapped to a decimal value. The return value is the global variable **\$equiv**. It's a bit clumsy versus a more elegant language, but...so it goes.

Let's put that snippet of code aside for a moment and look at the common question of how to parse a string one character at a time. There are a ton of different ways to figure out exactly how that should be done, but I'm going to tap the nifty Bash function **seq**.

The **seq** command generates a sequence of values from the starting to ending value—most easily at the command line:

```
$ seq 3 7
3
4
5
6
7
```

This is where the Bash string functions are going to be really helpful! Let's utilize two of them. First, **\${#string}** returns the number of characters in the string. Then the more complex reference of **\${string:start:num}** returns the substring starting at **start** for **num** characters from the variable **string**.

Put these three things together, and you have an elegant way to step through a string, character by character. If the user-specified value is known as **romanvalue** (so as to be maximally mnemonic, of course), this simple **for** loop breaks down the string:

```
for index in $(seq 1 ${#romanvalue}) ; do
echo "Letter $index is ${romanvalue:index-1:1}"
done
```

WORK THE SHELL

Now you can integrate that with the function `mapit` presented earlier. Combined, you get this script:

```
for index in $(seq 1 ${#romanvalue}) ; do
mapit ${romanvalue:index-1:1}
echo "${romanvalue:index-1:1} = $equiv"
done
```

The result? Let's decompose the earlier sequence:

```
$ sh roman.sh MCMLXIII
converting MCMLXIII
M = 1000
C = 100
M = 1000
L = 50
X = 10
I = 1
I = 1
```

It's pretty easy to sum it up:

```
sum=$(( $sum + $equiv ))
```

Without compensating for the subtractive notation, you'll find that—incorrectly!—MCMLXII sums up to 2163. It makes sense: the CM ended up as 1100 instead of 900, so it's exactly 200 off the correct value.

Obviously, there's a whole 'nother section of smarts needed in this program—an algorithm that basically says, "If the next value is greater than the current, subtract the current value from the next. If not, add the current to the next value and add that to the sum total."

WORK THE SHELL

The problem is that this means there's a fairly substantial change in the code, because you can't just say "look up value, add it to sum", but you need to implement a look-ahead concept.

And, that's exactly what I plan to cover in my next article, when I finish this script and look at the reverse—one that lets you get the Roman numeral equivalent of a decimal value. The latter is particularly interesting, because there's no Roman numeral greater than 1000, so given the notational conventions, the max value is going to be $3 \times 1000 + 999$.

Or is it? You tell me. ■

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

What's New in Kernel Development

By Zack Brown

Warning: this article contains profanity.

Linus' Behavior and the Kernel Development Community

On September 16, 2018, **Linus Torvalds** released the **4.19-rc4** version of the kernel, and he also announced he was taking a break from Linux development in order to consider his own behavior and to come up with a better approach to kernel development. This was partly inspired by his realization that he wasn't looking forward to the **Kernel Summit** event, and he said that "it wasn't actually funny or a good sign that I was hoping to just skip the yearly kernel summit entirely."

He also wrote that it was partly inspired when:

...people in our community confronted me about my lifetime of not understanding emotions. My flippant attacks in emails have been both unprofessional and uncalled for. Especially at times when I made it personal. In my quest for a better patch, this made sense to me. I know now this was not OK and I am truly sorry.

So he said, "I am going to take time off and get some



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.

assistance on how to understand people's emotions and respond appropriately.”

He compared the situation to the kind of “pain points” the Linux kernel project has experienced on a technical level in the past, like moving from tarballs to **BitKeeper**, and from BitKeeper to **git**. And he remarked that “We haven't had that kind of pain-point in about a decade. But this week felt like that kind of pain point to me.”

He also added, by way of clarification, that “This is not some kind of ‘I'm burnt out, I need to just go away’ break. I'm not feeling like I don't want to continue maintaining Linux. Quite the reverse. I very much **do** want to continue to do this project that I've been working on for almost three decades.”

That was the last post Linus sent to the mailing list, up to the time of this writing. However, he and several other kernel developers signed off on a patch to the kernel tree, incorporating a new Code of Conduct policy. It's fairly boilerplate—basically, don't be mean, don't discriminate, violations will be investigated, and appropriate measures taken.

It's not a new idea. Long ago, **Richard Stallman** used to troll the mailing list trying to start an argument about “Linux” vs. “GNU/Linux”, until the mailing list maintainers threatened to ban him if he kept it up. They phrased it as a general rule, not unlike a code of conduct.

There's been a wide range of responses to Linus' announcement and to the Code of Conduct itself. Some felt that Linus' earlier behavior had been community-strengthening, encouraging people to respond as equals and duke it out with Linus on the issues they cared about.

Some felt that Linus was taking a really wonderful step, seeking feedback and reflecting on the issues, and they in turn offered their own insights and assistance.

Some, on the other hand, felt that it was wrong to start welcoming political agendas in software projects. They felt that adopting codes of conduct was a way for certain

interests to gain other forms of control over software projects.

Some people felt that the Code of Conduct had not been properly discussed in a public forum, as other patches were, and should not go directly into the kernel without that kind of process.

Some felt that the “Code of Conflict”, which had been in the kernel source tree since 2015, was plenty good enough and did not need to be revised into this new Code of Conduct. The two are very similar—both call for investigations to be conducted by the **Linux Foundation’s Technical Advisory Board** (TAB)—but the Code of Conduct is more explicit about the types of communications that would be considered violations. Some people felt that the added specificity was not needed and shouldn’t be adopted.

Michael Woods made a particularly impassioned case against a code of conduct:

Whoever convinced you to add the Code of Conduct was convincing you to give control over to a social justice initiative that has no interest in the kernel’s core function or reason for existence.

Codes of conduct are tools used by the incompetent to wrest control away from the people who own the project, so they can feed on the corpse and wear the skin of the project as a fetish play.

Examples of these people trying to introduce codes of conduct, with commentary on the emotions and motivations driving CoC introduction:

- LLVM: <http://voxdays.blogspot.com/2018/05/the-costs-of-code-of-conduct.html>
- PHP: <http://voxdays.blogspot.com/2016/01/initial-sjw-attack-defeated.html>
- PHP 2: <http://voxdays.blogspot.com/2016/01/a-second-sjw-attack-on-php.html>

diff -u

- Ruby: <http://voxdays.blogspot.com/2016/01/more-sjw-attacks-in-tech.html>
- Ruby 2: <http://voxdays.blogspot.com/2016/01/the-sjw-war-on-ruby-continues.html>
- Node.js: <http://voxdays.blogspot.com/2017/08/how-sjws-react-to-defeat.html>
- Awesome-Django: <http://voxdays.blogspot.com/2015/10/exposing-true-face-of-sjw.html>
- Go: <http://voxdays.blogspot.com/2015/06/you-cant-run-you-cant-hide.html>

Pavel Snajdr replied to Michael's argument by saying, "how about if we viewed the new Code of Conduct as about the same thing as BitKeeper was for the development process? It was not perfect, but was *something* for a start. And I believe that Linus will probably come back with a Git of CoC, or something in that fasion."

Meanwhile, **Luke Kenneth Casson Leighton** objected to the Code of Conduct perhaps even more forcefully than Michael, saying directly to Linus:

this is beginning to remind me of dr who films, the comedy film "the world's end", and various other b-movie horror shows where people were taken over through mind-control or replaced.

so i apologise, i'm going to stop pussy-footing around and ask HAVE YOU FUCKING LOST IT, GET YOUR HEAD OUT YOUR ARSE, STOP FEELING SORRY FOR YOURSELF AND GET BACK TO BEING AN ENGINEER, YOU ARE ON A CHEARRRRGEUUH YOU SORRY LITTLE PROGRAMMERRRRR

cough. enough NLP-esque shock tactics with a bit of comedy thrown in to take the sting out of it...allow me to return to rational insights.

(1) you apologised for your behaviour, and it's fantastic that you recognised that there was a problem and asked for help. however, you *may* be feeling a little guilty, and it's clearly knocked your confidence, and that unfortunately has allowed political

diff -u

correctness to “creep in” where we know it never, ever belongs: in engineering.

the next thing you know, the fucking guilt-ridden morons who want the words “master” and “slave” erased from the history books will be telling you that we have to change SPI’s “MOSI” and “MISO” to...god...i dunno...”ROWI and RIWO” - “requestor” and “worker” or something incredibly stupid:

Requestor: “i’m awfully sorry, if you wouldn’t mind, if it’s not too much trouble mr worker, when you have the time and you’re not on your union-mandated break, could you deal with this bit-change for me?”

(2) more and more people are raising the fact that the change was made without consultation. this **is** going to bite everyone. i strongly, strongly suggest reverting it: i made the point very clear that it wasn’t the actual CoC that was the problem, it was that you, yourself, were not really obeying it (so nobody else could, either).

(3) let’s look at what toxic documents named “codes of conduct” look like from an engineering perspective:

```
#define BEHAVIOUR_GOOD() ((~BEHAVIOUR_BAD) == 0)
#define BEHAVIOUR_BAD BEHAVIOUR_SEXIST | BEHAVIOUR_RACIST |
                    BEHAVIOUR_NAZI |
BEHAVIOUR_UNPLEASANT |
                    BEHAVIOUR_RELIGIOUS_EXTREMIST
....
#define BEHAVIOUR_RELIGIOUS_EXTREMIST \
                    BEHAVIOUR_ANTI_CHRISTIAN \
                    BEHAVIOUR_ANTI_MUSLIM \
                    ...
....
....
#define BEHAVIOUR_ANTI_MUSLIM 0x1
#define BEHAVIOUR_ANTI_CHRISTIAN 0x2
```

diff -u

...

...

...

```
// oops fuck we're gonna run out of bits extremely quickly....
```

do you see where that's going? do you get the point already? if an engineer proposed the above patch to create the toxic CoC document that insidiously crept in recently, you and pretty much everyone would think that the submitter had a fucking screw loose and needed psychiatric help.

these toxic documents do not have to spell it out, but they **imply** that there are (deep breath...) [*insert list of racist names here*] and their mothers too all trying to ATTACK the project, and we'd better make sure that they're all excluded, otherwise we're all in trouble, eh?

i apologise for using these words: if you are a decent human being you should by now be feeling physically sick to your stomach at having read that paragraph, that those words were even used...yet they're not actually **in** that toxic document, but they don't have to be: people are still thinking them. like the "don't think of a pink elephant" our subconscious mind cannot help by strip out the "don't".

bottom line: the **entire linux kernel project** has now been **completely poisoned** by that document.

put another way: an engineer would go, "wtf???" and would say "we don't need to fill every single bit in the bitfield and then invert it for god's sake! just say 'good behaviour is expected' and be done with it!!"

so why not say, instead of that absolute god-awful list, "everyone is welcome; everyone belongs". you see the difference? you see how simple and empowering that is? it's INVITING people to participate, and it's pretty obvious that if someone feels **UN**welcome, the rules have been broken and they can raise it as an issue. rather than absolutely terrifying and sickening absolutely everybody.

the analogy is the story of mother theresa being invited to an “anti-war” rally. she declined...and said, “if ever you hold a PEACE rally, i’d be delighted to attend”.

so come on, linus: wake up, man. just because this is outside of your area of expertise does not mean that you have to let go of the reins. *get a grip*. use your engineering expertise, apply it to the problem, work with *EVERYONE* and work out an *ACCEPTABLE* solution.

This particular topic was more about everyone responding to Linus’ announcement, than about discussing the issues among themselves. In general, some people were in favor of the new Code of Conduct, and some were opposed.

For myself, I hope Pavel Snajdr is right—that the Code of Conduct is the “BitKeeper” of this particular issue. The last time Linus took a break from kernel development, he came out with git, a transformative tool that completely changed the way people developed software all over the world.

But it may not be very realistic to expect Linus to pull something like that out of his butt in this case. This isn’t just a technical issue. It’s a political issue, with strong, uncompromising feelings on all sides of it—not to mention powerful entities with a vested financial interest in seeing Linux itself fall to ruin. If Linus returns to kernel development with anything like the “git” of community relations, maybe we should then ask him to take a longer break from kernel development and address the issues of race, poverty, global warming and the rise of fascism.

Virtualizing the Clock

Dmitry Safonov wanted to implement a namespace for time information. The twisted and bizarre thing about virtual machines is that they get more virtual all the time. There’s always some new element of the host system that can be given its own namespace and enter the realm of the virtual machine. But as that process rolls forward, virtual systems have to share aspects of themselves with other virtual systems and the host system itself—for example, the date and time.

Dmitry's idea is that users should be able to set the day and time on their virtual systems, without worrying about other systems being given the same day and time. This is actually useful, beyond the desire to live in the past or future. Being able to set the time in a container is apparently one of the crucial elements of being able to migrate containers from one physical host to another, as Dmitry pointed out in his post.

As he put it:

The kernel provides access to several clocks: `CLOCK_REALTIME`, `CLOCK_MONOTONIC`, `CLOCK_BOOTTIME`. Last two clocks are monotonous, but the start points for them are not defined and are different for each running system. When a container is migrated from one node to another, all clocks have to be restored into consistent states; in other words, they have to continue running from the same points where they have been dumped.

Dmitry's patch wasn't feature-complete. There were various questions still to consider. For example, how should a virtual machine interpret the time changing on the host hardware? Should the virtual time change by the same offset? Or continue unchanged? Should file creation and modification times reflect the virtual machine's time or the host machine's time?

Eric W. Biederman supported this project overall and liked the code in the patch, but he did feel that the patch could do more. He thought it was a little too lightweight. He wanted users to be able to set up new time namespaces at the drop of a hat, so they could test things like leap seconds before they actually occurred and see how their own projects' code worked under those various conditions.

To do that, he felt there should be a whole "struct timekeeper" data structure for each namespace. Then pointers to those structures could be passed around, and the times of virtual machines would be just as manipulable and useful as times on the host system.

In terms of timestamps for filesystems, however, Eric felt that it might be best to limit

diff -u

the feature set a little bit. If users could create files with timestamps in the past, it could introduce some nasty security problems. He felt it would be sufficient simply to “do what distributed filesystems do when dealing with hosts with different clocks”.

The two went back and forth on the technical implementation details. At one point, Eric remarked, in defense of his preference:

My experience with namespaces is that if we don't get the advanced features working there is little to no interest from the core developers of the code, and the namespaces don't solve additional problems. Which makes the namespace a hard sell. Especially when it does not solve problems the developers of the subsystem have.

At one point, **Thomas Gleixner** came into the conversation to remind Eric that the time code needed to stay fast. Virtualization was good, he said, but “timekeeping_update() is already heavy and walking through a gazillion of namespaces will just make it horrible.”

He reminded Eric and Dmitry that:

It's not only timekeeping, i.e. reading time, this is also affecting all timers which are armed from a namespace.

That gets really ugly because when you do `settimeofday()` or `adjtimex()` for a particular namespace, then you have to search for all armed timers of that namespace and adjust them.

The original posix timer code had the same issue because it mapped the clock realtime timers to the timer wheel so any setting of the clock caused a full walk of all armed timers, disarming, adjusting and requeing them. That's horrible not only performance wise, it's also a locking nightmare of all sorts.

Add time skew via NTP/PTP into the picture and you might have to adjust timers as well, because you need to guarantee that they are not expiring early.

diff -u

So, there clearly are many nuances to consider. The discussion ended there, but this is a good example of the trouble with extending Linux to create virtual machines. It's almost never the case that a whole feature can be fully virtualized and isolated from the host system. Security concerns, speed concerns, and even code complexity and maintainability come into the picture. Even really elegant solutions can be shot down by, for example, the possibility of hostile users creating files with unnaturally old timestamps.

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.

Disclaimer

The views and opinions expressed in this article are those of the author and do not necessarily reflect those of *Linux Journal*. ■

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

A woman with glasses, wearing a light-colored button-down shirt, is looking intently at a computer monitor in a server room. The room is dimly lit with green and blue light from the server racks. The background shows rows of server racks with glowing lights.

DEEP DIVE

MONITORING

Why Your Server Monitoring (Still) Sucks

Five observations about why your server monitoring still stinks by a monitoring specialist-turned-consultant.

By Mike Julian

Early in my career, I was responsible for managing a large fleet of printers across a large campus. We're talking several hundred networked printers. It often required a 10- or 15-minute walk to get to some of those printers physically, and many were used only sporadically. I didn't always know what was happening until I arrived, so it was anyone's guess as to the problem. Simple paper jam? Driver issue? Printer currently on fire? I found out only after the long walk. Making this even more frustrating for everyone was that, thanks to the infrequent use of some of them, a printer with a problem might go unnoticed for weeks, making itself known only when someone tried to print with it.

Finally, it occurred to me: wouldn't it be nice if I knew about the problem and the cause *before* someone called me? I found my first monitoring tool that day, and I was absolutely hooked.

Since then, I've helped numerous people overhaul their monitoring systems. In doing so, I noticed the same challenges repeat themselves regularly. If you're responsible for managing the systems at your organization, read on; I have much advice to dispense.

So, without further ado, here are my top five reasons why your monitoring is crap and what you can do about it.

1. You're Using Antiquated Tools

By far, the most common reason for monitoring being screwed up is a reliance on antiquated tools. You know that's your issue when you spend too much time working around the warts of your monitoring tools or when you've got a bunch of custom code to get around some major missing functionality. But the bottom line is that you spend more time trying to fix the almost-working tools than just getting on with your job.

The problem with using antiquated tools and methodologies is that you're just making it harder for yourself. I suppose it's certainly possible to dig a hole with a rusty spoon, but wouldn't you prefer to use a shovel?

Great tools are invisible. They make you more effective, and the job is easier to accomplish. When you have great tools, you don't even notice them.

Maybe you don't describe your monitoring tools as "easy to use" or "invisible". The words you might opt to use would make my editor break out a red pen.

This checklist can help you determine if you're screwing yourself.

- Are you using Nagios or a Nagios derivative to monitor elastic/ephemeral infrastructure?
- Is there a manual step in your deployment process for a human to "Add \$thing to monitoring"?
- How many post-mortems contained an action item such as, "We weren't monitoring \$thing"?
- Do you have a cron job that tails a log file and sends an email via sendmail?

DEEP DIVE

- Do you have a syslog server to which all your systems forward their logs...never to be seen again?
- Do you collect system metrics only every five metrics (or even less often)?

If you answered yes to any of those, you are relying on bad, old-school tooling. My condolences.

The good news is your situation isn't permanent. With a little work, you can fix it.

If you're ready to change, that is.

It is somewhat amusing (or depressing?) that we in Ops so readily replace entire stacks, redesign deployments over a week, replace configuration management tools and introduce modern technologies, such as Docker and serverless—all without any significant vetting period.

Yet, changing a monitoring platform is *verboten*. What gives?

I think the answer lies in the reality of the state of monitoring at many companies. Things are pretty bad. They're messy, inconsistent in configuration, lack a coherent strategy, have inadequate automation...but it's all built on the tools we know. We know their failure modes; we know their warts.

For example, the industry has spent years and a staggering amount of development hours bolting things onto Nagios to make it more palatable (such as nagios-herald, NagiosQL, OMD), instead of asking, "Are we throwing good money after bad?"

The answer is yes. Yes we are.

Not to pick on Nagios—okay, yes, I'm going to pick on Nagios. Every change to the Nagios config, such as adding or removing a host, requires a config reload. In an infrastructure relying on ephemeral systems, such as containers, the entire

infrastructure may turn over every few minutes. If you have two-dozen containers churning every 15 minutes, it's possible that Nagios is reloading its config more than once a minute. That's insane.

And what about your metrics? The old way to decide whether something was broken was to check the current value of a check output against a threshold. That clearly results in some false alarms, so we added the ability to fire an alert only if N number of consecutive checks violated the threshold. That has a pretty glaring problem too. If you get your data every minute, you may not know of a problem until 3–5 minutes after it's happened. If you're getting your data every five minutes, it's even worse.

And while I'm on my soapbox, let's talk about automation. I remember back when I was responsible for a dozen servers. It was a big day when I spun up server #13. These sorts of things happened only every few months. Adding my new server to my monitoring tools was, of course, on my checklist, and it certainly took more than a few minutes to do.

But the world of tech isn't like that anymore. Just this morning, a client's infrastructure spun up a dozen new instances and spun down half of them an hour later. I knew it happened only after the fact. The monitoring systems knew about the events within seconds, and they adjusted accordingly.

The tech world has changed dramatically in the past five years. Our beloved tools of choice haven't quite kept pace. Monitoring must be 100% automated, both in registering new instances and services, and in de-registering them all when they go away. Gone are the days when you can deal with a 5 (or 15!) minute delay in knowing something went wrong; many of the top companies know within seconds that something isn't right.

Continuing to rely on methodologies and tools from the old days, no matter how much you enjoy them and know their travails, is holding you back from giant leaps forward in your monitoring.

The bad old days of trying to pick between three equally terrible monitoring tools are long over. You owe it to yourself and your company at least to consider modern tooling—whether it's SaaS or self-hosted solutions.

2. You're Chasing "the New Hotness"

At the other end of the spectrum is an affinity for new-and-exciting tools. Companies like Netflix and Facebook publish some really cool stuff, sure. But that doesn't necessarily mean you should be using it.

Here's the problem: you are (probably) not Facebook, Netflix, Google or any of the other huge tech companies everyone looks up to. **Cargo culting** never made anything better.

Adopting someone else's tools or strategy because they're successful with them misses the crucial reasons of *why* it works for them.

The tools don't make an organization successful. The organization is successful because of how its members think. Its approaches, beliefs, people and strategy led the organization to create those tools. Its success stems from something much deeper than, "We wrote our own monitoring platform."

To approach the same sort of success the industry titans are having, you have to go deeper. What do they do know that you don't? What are they doing, thinking, saying, believing that you aren't?

Having been on the inside of many of those companies, I'll let you in on the secret: they're good at the fundamentals. Really good. Mind-blowingly good.

At first glance, this seems unrelated, but allow me to quote John Gall, famed systems theorist:

A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and

cannot be patched up to make it work. You have to start over, beginning with a working simple system.

Dr. Gall quite astutely points out the futility of adopting other people's tools wholesale. Those tools evolved from simple systems to suit the needs of that organization and culture. Dropping such a complex system into another organization or culture may not yield favorable results, simply because you're attempting to shortcut the hard work of evolving a simple system.

So, you want the same success as the veritable titans of industry? The answer is straightforward: start simple. Improve over time. Be patient.

3. You're Unnecessarily Afraid of "Vendor Lock-in"

If there's one argument I wish would die, it's the one where people opine about wanting to "avoid vendor lock-in". That argument is utter hogwash.

What is "vendor lock-in", anyway? It's the notion that if you were to go all-in on a particular vendor's product, it would become prohibitively difficult or expensive to change. Keurig's K-cups are a famous example of vendor lock-in. They can be used only with a Keurig coffee machine, and a Keurig coffee machine accepts only the proprietary Keurig K-cups. By buying a Keurig, you're locked in to the Keurig ecosystem.

Thus, if I were worried about being locked in to the Keurig ecosystem, I'd just avoid buying a Keurig machine. Easy.

If I'm worried about vendor lock-in with, say, my server infrastructure, what do I do? Roll out both Dell and HP servers together? That seems like a really dumb idea. It makes my job way more difficult. I'd have to build to the lowest common denominator of each product and ignore any product-specific features, including the innovations that make a product appealing. This ostensibly would allow me to avoid being locked in to one vendor and keep any switching costs low, but it also means I've got a solution that only half works and is a nightmare to manage at any sort of scale. (Have

you ever tried to build tools to manage and automate both iDRAC and IPMI? You really don't want to.)

In particular, you don't get to take advantage of a product's unique features. By trying to avoid vendor lock-in, you end up with a "solution" that ignores any advanced functionality.

When it comes to monitoring products, this is even worse. Composability and interoperability are a core tenet of most products available to you. The state of monitoring solutions today favors a high degree of interoperability and open APIs. Yes, a single vendor may have all of your data, but it's often trivial to move that same data to another vendor without a major loss of functionality.

One particular problem with this whole vendor lock-in argument is that it's often used as an excuse not to buy SaaS or commercial, proprietary applications. The perception is that by using only self-hosted, open-source products, you gain more freedom.

That assumption is wrong. You haven't gained more freedom or avoided vendor lock-in at all. You've traded one vendor for another.

By opting to do it all yourself (usually poorly), you effectively become your own vendor—a less experienced, more overworked vendor. The chances you would design, build, maintain and improve a monitoring platform better—on top of your regular duties—than a monitoring vendor? They round to zero. Is tool-building really the business you want to be in?

In addition, switching costs from in-house solutions are astronomically higher than from one commercial solution to another, because of the interoperability that commercial vendors have these days. Can the same be said of your in-house solution?

4. You're Monitoring the Wrong Stuff

Many years ago, at one of my first jobs, I checked out a database server and noticed it had high CPU utilization. I figured I would let my boss know.

“Who complained about it?”, my boss asked.

“Well, no one”, I replied.

My boss’ response has stuck with me. It taught me a valuable lesson: “if it’s not impacting anyone, is there really a problem?”

My lesson is this: data without context isn’t useful. In monitoring, a metric matters only in the context of users. If low free memory is a condition you notice but it’s not impacting users, it’s not worth firing an alert.

In all my years of operations and system administration, I’ve not once seen an OS metric directly indicate active user impact. A metric sometimes can be an *indirect* indicator, but I’ve never seen it *directly* indicate an issue.

Which brings me to the next point. With all of these metrics and logs from the infrastructure, why is your monitoring not better off? The reason is because Ops can solve only half the problem. While monitoring nginx workers, Tomcat garbage collection or Redis key evictions are all important metrics for understanding infrastructure performance, none of them help you understand the software your business runs. The biggest value of monitoring comes from instrumenting the applications on which your users rely. (Unless, of course, your business provides infrastructure as a service—then, by all means, carry on.)

Nowhere is this more clear than in a SaaS company, so let’s consider that as an example.

Let’s say you have an application that is a standard three-tier web app: nginx on the front end, Rails application servers and PostgreSQL on the back end. Every action on the site hits the PostgreSQL database.

You have all the standard data: access and error logs, nginx metrics, Rails logs, Postgres metrics. All of that is great.

You know what’s even better? Knowing how long it takes for a user to log in.

Or how many logins occur per minute. Or even better: how many login failures occur per minute.

The reason this information is so valuable is that it *tells you about the user experience directly*. If login failures rose during the past five minutes, you know you have a problem on your hands.

But, you can't see this sort of information from the infrastructure perspective alone. If I were to pay attention only to the nginx/Rails/Postgres performance, I would miss this incident entirely. I would miss something like a recent code deployment that changed some login-related code, which caused logins to fail.

To solve this, become closer friends with your engineering team. Help them identify useful instrumentation points in the code and implement more metrics and logging. I'm a big fan of the statsd protocol for this sort of thing; most every monitoring vendor supports it (or their own implementation of it).

5. You Are the Only One Who Cares

If you're the only one who cares about monitoring, system performance and useful metrics will never meaningfully improve. You can't do this alone. You can't even do this if only your team cares. I can't begin to count how many times I've seen Ops teams put in the effort to make improvements, only to realize no one outside the team paid attention or thought it mattered.

Improving monitoring requires company-wide buy-in. Everyone from the receptionist to the CEO has to believe in the value of what you're doing. Everyone in the company knows the business needs to make a profit. Similarly, it requires a company-wide understanding that improving monitoring improves the bottom line and protects the company's profit.

Ask yourself: why do you care about monitoring?

Is it because it helps you catch and resolve incidents faster? Why is that important

to you?

Why should that be important to your manager? To your manager's manager? Why should the CEO care?

You need to answer those questions. When you do so, you can start making compelling business arguments for the investments required (including in the best new tools).

Need a starting point? Here are a few ideas why the business might care about improving monitoring:

- The business can manage and mitigate the risk of incidents and failures.
- The business can spot areas for performance improvements, leading to a better customer experience and increased revenue.
- The business can resolve incidents faster (often before they become critical), leading to more user goodwill and enhanced reputation.
- The business avoids incidents going from bad to worse, which protects against loss of revenue and potential SLA penalty payments.
- The business better controls infrastructure costs through capacity planning and forecasting, leading to improved profits and lower expenses.

I recommend having a candid conversation with your team on why they care about monitoring. Be sure to involve management as well. Once you've had those conversations, repeat them again with your engineering team. And your product management team. And marketing. And sales. And customer support.

Monitoring impacts the entire company, and often in different ways. By the time you find yourself in a conversation with executives to request an investment in monitoring,

you will be able to speak their language. Go forth and fix your monitoring. I hope you found at least a few ideas to improve your monitoring. Becoming world-class in this is a long, hard, expensive road, but the good news is that you don't really need to be among the best to see massive benefits. A few straightforward changes, added over time, can radically improve your company's monitoring.

To recap:

1. Use better tools. Replace them as better tools become available.
2. But, don't fixate on the tools. The tools are there to help you solve a problem—they aren't the end goal.
3. Don't worry about vendor lock-in. Pick products you like and go all-in on them.
4. Be careful about what you collect and on what you issue alerts. The best data tells you about things that have a direct user impact.
5. Learn why your company cares about monitoring and express it in business outcomes. Only then can you really get the investment you want.

Good luck, and happy monitoring. ■

Mike Julian is the Editor of the *Monitoring Weekly* newsletter, author of O'Reilly's *Practical Monitoring*, and an independent monitoring consultant at AsterLabs.io. Before embarking off as a consultant, he worked as an Ops Engineer for Taos Consulting, Peak Hosting and Oak Ridge National Laboratory, and others. You can follow him on Twitter at @mike_julian.

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

CloudWatch Is of the Devil, but I Must Use It

Let's talk about Amazon CloudWatch.

By Corey Quinn

For those fortunate enough to not be stuck in the weeds of Amazon Web Services (AWS), CloudWatch is, and I quote from the official [AWS description](#), “a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers.” This is all well and good, except for the part where there isn't a single named constituency who enjoys working with the product. Allow me to dispense some monitoring heresy.

Better, let me describe this in the context of the 14 [Amazon Leadership Principles](#) that reportedly guide every decision Amazon makes. When you take a hard look at CloudWatch's complete failure across all 14 Leadership Principles, you wonder how this product ever made it out the door in its current state.

“Frugality”

I'll start with billing. Normally left for the tail end of articles like this, the CloudWatch billing paradigm is so terrible, I'm leading with it instead. You get billed per metric, per month. You get billed per thousand metrics you request to view via the API. You get billed per dashboard per month. You get billed per alarm per month. You get charged for logs based upon data volume ingested, data volume stored and “vended logs” that get published natively by AWS

services on behalf of the customer. And, you get billed per custom event. All of this can be summed up best as “nobody on the planet understands how your CloudWatch metrics and logs get billed”, and it leads to scenarios where monitoring vendors can inadvertently cost you thousands of dollars by polling CloudWatch too frequently. When the AWS charges are larger than what you’re paying your monitoring vendor, it’s not a wonderful feeling.

“Invent and Simplify”

CloudWatch Logs, CloudWatch Events, Custom Metrics, Vended Logs and Custom Dashboards all mean different things internally to CloudWatch from what you’d expect, compared to metrics solutions that actually make some fathomable level of sense. There are, thus, multiple services that do very different things, all operating under the “CloudWatch” moniker. For example, it’s not particularly intuitive to most people that scheduling a Lambda function to invoke once an hour requires a custom CloudWatch Event. It feels overly complicated, incredibly confusing, and very quickly, you find yourself in a situation where you’re having to build complex relationships to monitor things that are themselves far simpler.

“Think Big”

All business people, when asked what they want from a monitoring platform, will respond with something that resembles “a dashboard” or “a single pane of glass view”. CloudWatch offers minutia up the wazoo, but it categorically offers no global view, no green/yellow/red status indicator that gives you even a glimmer of the overall health of your site. Want a graph of each core in your instance’s CPU for the past 30 seconds? Easy! Want to know if your entire company should be putting out the burning fire that is the current production state of your website? Keep looking—CloudWatch has nothing to offer you.

“Insist on the Highest Standards”

By its very nature, CloudWatch feels like small thinking. The entire experience, start to finish, smacks of “what’s the absolute least we could do and get away with it?” They built their MVP, and then just sorta...stopped, frozen in amber. They created a set of building blocks, except they didn’t solve the problem of

“how do I monitor my AWS resources?” Instead, it feels like the entire team phoned it in and let a large market of monitoring vendors develop as a result. None of those vendors have the level of access to the raw data that CloudWatch does; all of them have built better products. You’d think the CloudWatch team would take a clue from the innovation that’s rapidly happening in this space, but that’d require someone to **Learn and Be Curious**.

“Are Right, a Lot”

Recent data is “eventually consistent”, so you always get graphs like the one shown in Figure 1.

Here in reality, that would be a terrifying thing to see on an *accurate dashboard*—something is obviously very wrong with your site! For better or worse, the “accurate” description doesn’t apply to CloudWatch, and that’s just how your graphs always look. “Your metrics will be eventually consistent” is very nearly the last thing you want to hear about your monitoring platform, second only to “what metrics?” This ties directly to...

“Earn Trust”

Let me be very clear here—the real issue isn’t the ingestion problem. Absolutely every vendor on the planet has the same issue—you can’t display data you don’t have. Where CloudWatch drops the ball is in exposing this behavior to the end user without explanation as to what’s going on. Thus, until you grow accustomed to it, you have a heart-stopping moment of “what the hell just happened to the site” whenever you glance at a dashboard. This conditions you to be entirely too calm when looking at sensible dashboards when a disaster just happened. If you trust what the CloudWatch dashboards show you, you’re making a terrible mistake.

“Dive Deep”

If you’re using Lambda or Fargate, you have no choice but to use CloudWatch Logs, wherein searching for everything is absolutely terrible. If you’re using CloudWatch Logs to diagnose anything, congratulations: you’re diving so deep, you may drown before making it back to the surface. For example, if I have a Lambda function that

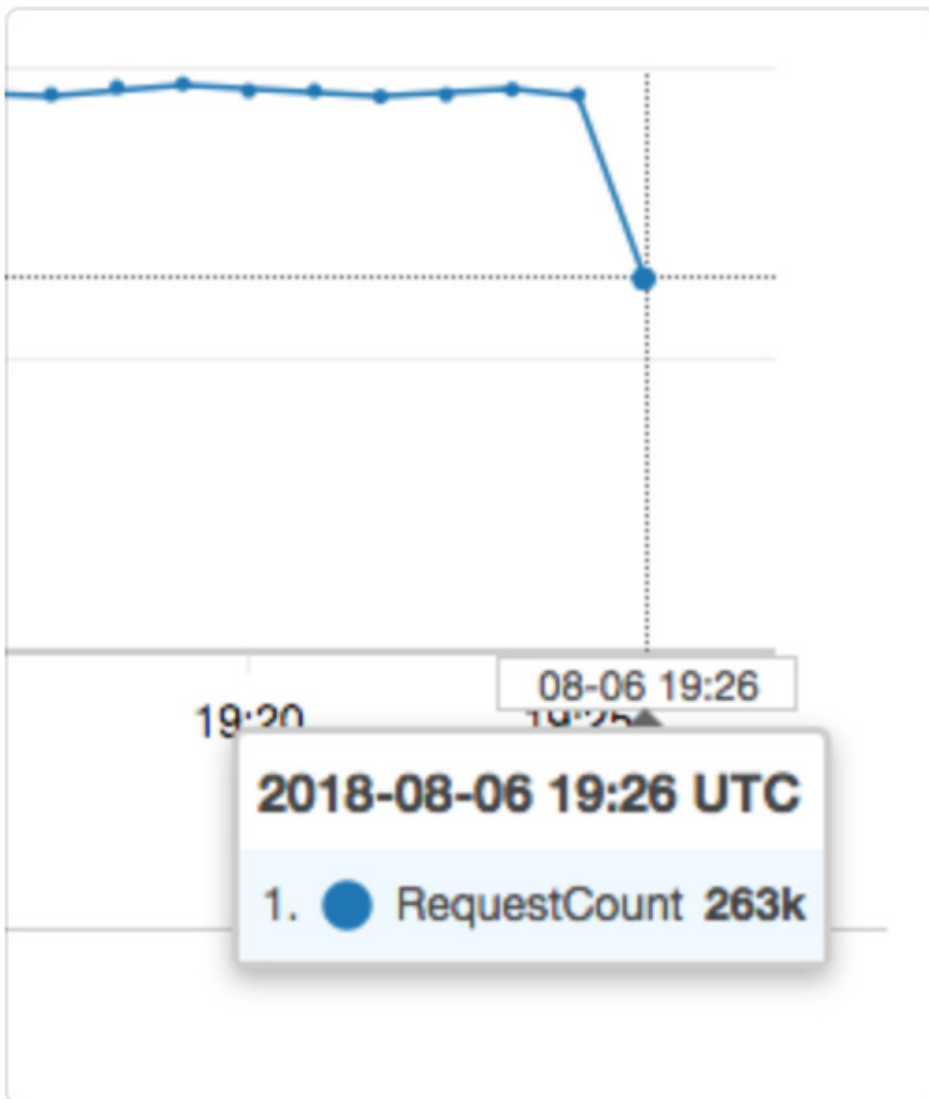


Figure 1. Example CloudWatch Graph

throws an error, in order to diagnose the problem, I must:

- Find the fact that it encountered an error in the first place by looking at the invocation error CloudWatch dashboard. I also could set up a filter to run a continuous query on the logs and alert when something shows up, except that isn't natively supported—I need a third-party tool for that (such as PagerDuty).
- Go diving into a variety of CloudWatch log groups and find the one named after

the specific erroring function.

- Scroll manually through the many, many, many pages of log groups to find the specific invocation that threw an error.
- Realize that the JSON object that's retained isn't enough to troubleshoot with, cry in despair, and go write an article just like this one.
- Do some quick math and realize I'm paying an uncomfortable percentage of my AWS bill for a service that's only of somewhat marginal utility at best.

"Deliver Results"

All of your metrics, all of your logs—they're locked away inside CloudWatch's various components. You're not going to find a "page me when this threshold is exceeded" option in CloudWatch; your options are relegated to "design an alert delivery pipeline with baling wire and SNS" or pay a non-AWS vendor for another monitoring product.

"Customer Obsession"

CloudWatch keeps all of your metrics. It keeps your logs. It lets you build custom dashboards to view your metrics all in one place. The building blocks of a great service are already here—it's the expression of that utility that falls short, sometimes drastically. The fact that large monitoring vendors are premier sponsors of AWS events would be laughable if CloudWatch ever were to get its act together. You'd not need a third party to make sense of a pure AWS environment, and many of them would starve to death as they grow too weak to interrupt your conversation to ask if they can scan your badge. Choosing to use CloudWatch vs. literally anything else is like buying a car. "Why yes, I would like to buy the Yugo instead of the Honda. After all, it checks all the boxes of technically being a car, so it's fine, right?"

"Disagree and Commit"

It may very well be that the root cause of many of CloudWatch's failings comes

from the product engineers who built it misunderstanding this (admittedly slippery!) Leadership Principle. It's envisioned as passionately expressing your reservations about a decision, but once it's reached that, you commit to the decision that was made. Unfortunately, it appears that the engineering teams responsible for CloudWatch decided to "Disagree in Commits" and inflict their arguments upon the world in the form of the product.

"Ownership"

If I were to go on the internet and post about how terrible virtually any other AWS service was, people would rally to that service's defense. It's the internet; people will do that. But when these and many more similar comments about CloudWatch appear, and nobody from AWS pipes in to say "wow, I'm sorry, why do you feel that way?", it's abundantly clear that if any people on the CloudWatch team really care about the product, they've been locked in a malfunctioning bathroom stall for the better part of a decade. [These comments](#) go back at least that far, but [Amazon is totally on it](#), rocking the company's "**Bias for Action**" principle.

"Hire and Develop the Best"

The people who build CloudWatch aren't terrible at their jobs; I genuinely believe they don't quite grasp how their product is perceived. Given that it's poor form to write a rant like this and not offer suggestions for positive improvement, here are some product enhancements I'd like to see:

- Give me the option to rate-limit API calls at arbitrary levels rather than being surprised at month end by a bill that's approximately Zanzibar's GDP.
- "Here's an error that your Lambda function threw, here's the log output from that specific function" should be at most two clicks away—not 30.
- If your dog has a litter of 14 puppies, perhaps you don't need to name all of them subtle variations of the term "CloudWatch". The proliferation of services and companies that all start with the word "Cloud" is the subject of a completely separate rant.

DEEP DIVE

Please don't misunderstand me. I use, enjoy and promote AWS services, and I'm considered to be "an authentic voice" largely because in addition to praising things that are wonderful, I'll call out things that aren't, as I've just done. I've built my career and business on working within that ecosystem. I find AWS employees to be intelligent and well-intentioned, and most of their services quite good. CloudWatch could get there with some work, but it's got a number of very painful usability issues that keep it from being good, let alone great. ■

Corey Quinn is a Cloud Economist at the Quinn Advisory Group. He has a history as an engineering director, public speaker and cloud architect. Corey specializes in helping companies address horrifying AWS bills and curates LastWeekinAWS.com, a weekly newsletter summarizing the latest in AWS news, blogs and tips, sprinkled with snark. He has never worked at Amazon, for reasons that should be obvious.

Send comments or feedback
via <http://www.linuxjournal.com/contact>
or email ljournal@linuxjournal.com.



**Decentralized
Certificate Authority
and Naming**

Free and open source contributors only:

handshake.org/signup

Bare-Bones Monitoring with Monit and RRDtool

How to provide robust monitoring to low-end systems.

By Andy Carlson

When running a critical system, it's necessary to know what resources the system is consuming, to be alerted when resource utilization reaches a specific level and to trend long-term performance. Zabbix and Nagios are two large-scale solutions that monitor, alert and trend system performance, and they each provide a rich user interface. Due to the requirements of those solutions, however, dedicated hardware/VM resources typically are required to host the monitoring solution. For smaller server implementations, options exist for providing basic monitoring, alerting and trending functionality. This article shows how to accomplish basic and custom monitoring and alerting using Monit. It also covers how to monitor long-term trending of system performance with RRDtool.

Initial Monit Configuration

On many popular Linux distros, you can install Monit from the associated software repository. Once installed, you can handle all the configuration with the `monitrc` configuration file. That file generally is located within the `/etc` directory structure, but the exact location varies based on your distribution.

The config file has two sections: Global and Services. The Global section allows for custom configuration of the Monit application. The Monit service

contains a web-based front end that is fully configurable through the config file. Although the section is commented out by default, you can uncomment items selectively for granular customization. The web configuration block looks like this:

```
set httpd port 2812 and
    use address localhost
    allow localhost
    allow admin:monit
```

The first line sets the port number where you can access Monit via web browser. The second line sets the hostname (the HTTP host header) that's used to access Monit. The third line sets the host from which the Monit application can be accessed. Note that you also can do this using a local firewall access restriction if a firewall is currently in place. The fourth line allows the configuration of a user name/password pair for use when accessing Monit. There's also a section that allows SSL options for encrypted connections to Monit. Although enabling SSL is recommended when passing authentication data, you also could reverse-proxy Monit through an existing web server, such as nginx or Apache, provided SSL is already configured on the web server. For more information on reverse-proxying Monit through Apache, see the Resources section at the end of this article.

The next items you need to enable deal with configuring email alerts. To set up the email server through which email will be relayed to the recipient, add or enable the following line:

```
set mailserver mailserver.company.com
```

Note that if there's a local SMTP server running, the server name of **mailserver.company.com** in this example may be replaced with **localhost**.

The next block to enable sets the contents of the email alert messages that will be

sent and will look similar to this:

```
set mail-format {  
    from:    Monit <monit@$HOST>  
    subject: Monit alert -- $EVENT $SERVICE  
    message: $EVENT Service $SERVICE  
            Date:      $DATE  
            Action:    $ACTION  
            Host:      $HOST  
            Description: $DESCRIPTION  
  
            Your faithful employee,  
            Monit  
}
```

Within this block, different predefined variables are used to provide alert-specific information (denoted by the \$ sign). You can modify text within the from, subject or message fields, and you also can add additional data to the message field as desired.

To finish the alerting functionality, you can configure an email address that will receive all email alerts from Monit by adding the following line:

```
set alert user@domain.com
```

At this point, the specified email address will receive all alerts generated by Monit. However, so far, no alerts are configured. To begin configuring alerts, let's first look at the Services section mentioned earlier. That section provides some basic monitoring functionality for the local machine, including CPU, memory, swap, filesystem and basic network monitoring. Each of those configuration items provides for the definition of thresholds. After the thresholds are met, actions can be taken, including sending an alert. As an example, the out-of-the-box alert for CPU/memory/swap monitoring looks like this:

```
check system $HOST
  if loadavg (1min) > 4 then alert
  if loadavg (5min) > 2 then alert
  if cpu usage > 95% for 10 cycles then alert
  if memory usage > 75% then alert
  if swap usage > 25% then alert
```

Again, note the use of variables to define the host to be monitored. While all of the triggers defined here result in an alert, other actions also can be taken. For more information on these settings, consult the Monit documentation (see Resources).

Custom Configuration of Monit

Once initial configuration is complete, you can define custom alerts. It's best to define the custom alerts outside the monitrc file. You do this by defining an include directory in the monitrc file as follows:

```
include /opt/monit-custom/*
```

This line includes all configuration files located in the /opt/monit-custom folder.

Next, let's look at two types of monitoring: host checks and program checks. Host checks allow for the monitoring of TCP-based services running on remote hosts. Although you can do basic TCP port connection testing for simpler services, Monit also provides the ability to do HTTP-based content checks to a specific URL. Consider the following example:

```
check host linuxjournal-website with address www.linuxjournal.com
  if failed
    port 443 protocol https
    with request / with content = "Become a Patron"
  then alert
```

The first line of the host check defines the identifier within Monit for this host (`linuxjournal-website`) and the address with which the host will be accessed (`www.linuxjournal.com`). In this example, the trigger within the host definition contains multiple conditions: it must be accessed via port 443 using the https protocol, and when accessed at the root URL, the text “Become a Patron” shows up in the response body. This check could be reconfigured to use port 80 and the http protocol.

Along with host monitoring, Monit allows the definition of script-based monitors, which is called a program check. Once a script is configured within Monit, the script will be executed periodically, and based on the script’s exit code, action may be taken.

Here’s an example of a script that alerts when an SSL certificate expiration date is within a specified number of days:

```
#!/bin/bash
```

```
domainexpiredate() {
    openssl x509 -text -in <(echo -n | \
    openssl s_client -connect $1:$2 2>/dev/null | \
    sed -n '/-*BEGIN/,/*END/p') 2>/dev/null | sed -n 's/
↵*Not After : *//p'
}

daysleft() {
    echo "(((date -d "$(domainexpiredate $1 $2)" +%s)-$(date
↵+%s))/24)/60)/60" | bc
}

defaultport() {
    if [ -z "$1" ]; then
        echo "443"
    fi
}
```

```
    else
        echo "$1"
    fi
}

[[ $(daysleft $2 $(defaultport $3)) -le $1 ]] && exit 1 ||
↪exit 0
```

This script is executed with two arguments: minimum number of days until expiration and the hostname of the server, with an optional third parameter for port number. Here's an example execution of the script:

```
$ checkcertexpire.sh 31 www.linuxjournal.com
$ echo $?
0
```

When the script is executed with the two required arguments, there is no console output. After the execution, if the return code is echoed (identified as `$?`), the value is 0, which indicates that the domain does not expire within 31 days. Configuring this item within Monit requires the following:

```
check program linuxjournal-ssl with path
↪"/etc/monit/scripts/checkcertexpire.sh 31 www.linuxjournal.com"
    if status != 0 then alert
```

In the same way as the host check, the program check has an identifier within Monit (`linuxjournal-ssl`, in this case). In the first line of the program check, along with the identifier, is the script to be executed along with the command-line arguments. Note that the trigger indicates that if the exit code is not 0, an alert should be sent.

Collecting Data with RRDtool

RRDtool is a very robust tool that lets you collect data over a long period of time.

Named after its database format (round-robin database), RRDtool saves time-based data to its database and then lets you retrieve and graph the data. RRDtool can graph any data that you can present through a command to a shell script.

Before capturing data, you must initialize the database. For this example, let's create a database to capture the five-minute load average. Here's the command to initialize this specific database:

```
rrdtool create loadavg_db.rrd --step 60  
↳DS:loadavg:GAUGE:120:0:10000 RRA:MAX:0.5:1:1500
```

The first two arguments indicate that a database named `loadavg_db.rrd` is being created. The `--step` argument defines the expected time gap between data samplings. In this case, 60 seconds are expected between samplings.

Let's look at two more arguments separately. The first of the two arguments begins with `DS` and defines a data set named `loadavg`. Note that the options for this data set are separated by colons. The `GAUGE` keyword says that when the data is read, it will be written to the database as is (unaltered). The `120` is the timeout in seconds to wait for data to be written to the database. If the data isn't written to the database within that window, zeros will be written to the database to indicate an error in the data feed. The `0` and `10000` are the minimum and maximum values that can be written to the database. The argument beginning with `RRA` defines the round-robin archive value. This defines how many values can be stored in the database and how long they'll be stored. The `MAX` indicates that the variable contains one value and shouldn't be modified in any way. The `0.5` indicates the initial resolution value. This is a standard value and shouldn't be changed. The `1` identifies how many steps should be averaged when storing a final value. In this case, there is one step value per value stored in the database. The final argument, `1440`, is how many steps will be stored in the database. Since the step length is 60 seconds, this configuration will provide 25 hours of data to be stored in the database.

Now that the data is initialized, you can capture and store it in the database. To

maintain accurate periodic data collection, it's best to create a crontab entry and have the data be collected at a desired interval. For this example, you would have the cron job run every minute. To collect data and put it in the database, use the following command:

```
rrdtool update loadavg_db.rrd --template loadavg N:$(cat  
↪/proc/loadavg | sed 's/^\([0-9\.\]\+\) .*$/\1/g')
```

To perform the data collection, the **update** argument along with the database name was used. The **--template** argument allows you to specify the variable name to populate with data. This is the same **loadavg** variable that was defined when the database was initialized. The **N** argument defines the data to be put into the **loadavg** variable. In this case, the result of the command substitution will be put into the database, which will be the five-minute load average. This command could be placed in the crontab for minute-by-minute execution. The crontab entry would look like this:

```
* * * * * /path/to/rrdtool-script.sh
```

Since all of the time fields contain asterisks, the specified script will run every minute. Once the database has been populated, you can render a graph with the following command:

```
rrdtool graph loadavg_graph-$(date +"%m-%d-%Y").png \  
-w 785 -h 120 -a PNG \  
--slope-mode \  
--start -86400 --end now \  
--font DEFAULT:7: \  
--title "5-minute load average" \  
--watermark "'date'" \  
--vertical-label "load average" \  
--lower-limit 0 \  
--right-axis 1:0 \  
--right-axis-label "1:0"
```

```
--x-grid MINUTE:10:HOUR:1:MINUTE:120:0:%R \  
--alt-y-grid --rigid \  
DEF:loadaverage=loadavg_db.rrd:loadavg:MAX \  
LINE1:loadaverage#0000FF:"load" \  
GPRINT:loadaverage:LAST:"Cur\ : %5.2lf" \  
GPRINT:loadaverage:AVERAGE:"Avg\ : %5.2lf" \  
GPRINT:loadaverage:MAX:"Max\ : %5.2lf" \  
GPRINT:loadaverage:MIN:"Min\ : %5.2lf\t\t\t"
```

The first line calls the RRDtool graph function along with the filename of the image to create. In this instance, the image filename will contain the current date. All of the arguments beginning with `--` set up the look and feel of the graph, including labels, axis configuration, image format and the time frame from which to pull the data. For detailed information on these arguments, see the RRDtool documentation.

The line beginning with `DEF:loadaverage` defines a graph variable named `loadaverage`, which will have the values from the `loadavg` variable you created in the database. The line beginning with `LINE` specifies the color of the graph line and the label to use in the legend. The `GPRINT` lines indicate various statistic details to be printed at the bottom of the graph. In this case, the last recorded value and the average, minimum and maximum values during the time frame will be displayed. Note that the `%5.2lf` specifies the value to be printed as a floating-point number with up to five digits to the left of the decimal point and two digits to the right.

For ease of capturing daily graphs, you also could add this command to the crontab to run daily with the following entry:

```
0 0 * * * /path/to/rrdtool-graph.sh
```

This will run the graph script every day at midnight. The images may now be placed in a folder that is accessible via a browser for easy viewing.

Although many monitoring solutions exist that provide robust graphical UIs,

these solutions provide basic monitoring and trending functionality while using a minimum of system resources and providing a basic framework for disseminating the data collected. ■

Andy Carlson has worked in IT for the past 15 years doing networking and server administration along with occasional coding. He is thankful to have chosen a career that he loves, grows in and learns from. He currently resides in Cincinnati, Ohio, with his wife, three daughters and his son. His family is currently in the process of adopting two children internationally. He enjoys playing the guitar, coding, and spending time with family and friends.

Resources

- [Monit Documentation](#)
- [Monit Apache Integration](#)
- [RRDtool Graph Function Reference](#)

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

How-To: Implementing a Real-Time Syslog Shipper for Your Terminal

Ever wondered how to tail `-F /var/log/messages` from multiple servers at once? Read on.

By Fabien Wernli

Troubleshooting Linux systems can be challenging, especially at times when the tools available to system administrators are constantly evolving. But, it's hard to avoid using some classic utilities on a daily basis. One of them can be summarized by the following command:

```
tail -F /var/log/messages
```

Reviewing logs is indeed a key player of the **“Utilization Saturation and Errors” (USE) method**. While storing historic logs from many servers in a centralized storage engine like **Elasticsearch** has become quite common nowadays, it's sometimes important to have a low-latency view of what's happening right now in your infrastructure. Unfortunately, there is no standard out-of-the-box tool to view logs in real time simultaneously on all hosts of a data center.

Here are some use cases where low-latency treatment makes sense, along with an example for each:

- Security incidents: account Y is compromised; trigger an alert for each successful connection attempt site-wide.
- Change management: committed a new change to configuration management; show each hosts' resource status as it changes.
- Real-time data mining: show all nodes on which application Z is currently serving more than N requests per second.

This article shows how to set up a site-wide low-latency (sub-millisecond) log shipping infrastructure. I'll do this with minimal intrusion and demonstrate its usage in the command-line interface, just like good-old `tail -f /var/log/messages`.

As your mileage may vary, let's stick to a simple scenario that you can adapt to your own use case. Most instructions given here are for recent Debian-based GNU/Linux distributions, but they easily can be adapted to other environments.

Scenario

Let's assume a number of Linux or UNIX servers, and that you'd like to be able to subscribe to all or a subset of their logs in real time, using a terminal. Let's refer to these servers as the *clients*.

Let's further assume that they all have a running syslog collection daemon, which you'll configure to forward the logs to a remote server that will serve as the log subscription *hub*.

Finally, you'll use a *control node* that will serve as the login host. This will be the human-machine interface. The control node can be the same machine as the hub, or you can use your workstation or laptop, provided the firewalls are set up accordingly.

Software

Although no extra software is required on the *clients*, you'll need the following on the *hub*:

- syslog-ng >= 3.6.1 (or earlier with syslog-ng-incubator).
- riemann = 0.3.0.

On the *control node*, you'll need to install the following:

- A websocket (WS) client or a server-side-event (SSE) client:
python websocket-client = 0.47.0 or nodejs wscat = 2.1.1.
- riemann-dash = 0.2.14 (optional).

How It Works

Before getting your hands dirty modifying configuration files, let's get a glimpse of the big picture. See Figure 1 for a diagram of the overall architecture and event flow.

This diagram shows the following:

1. An application “app” on the *client* calls **syslog()** to log a message about an event.
2. The local syslog daemon captures the event and sends it to the remote *hub* using the syslog protocol.
3. The syslog-ng daemon on the *hub* forwards the event to the riemann daemon using **protocol buffers**.
4. The *control node* issues a subscription request to the *hub* using either WS or SSE.

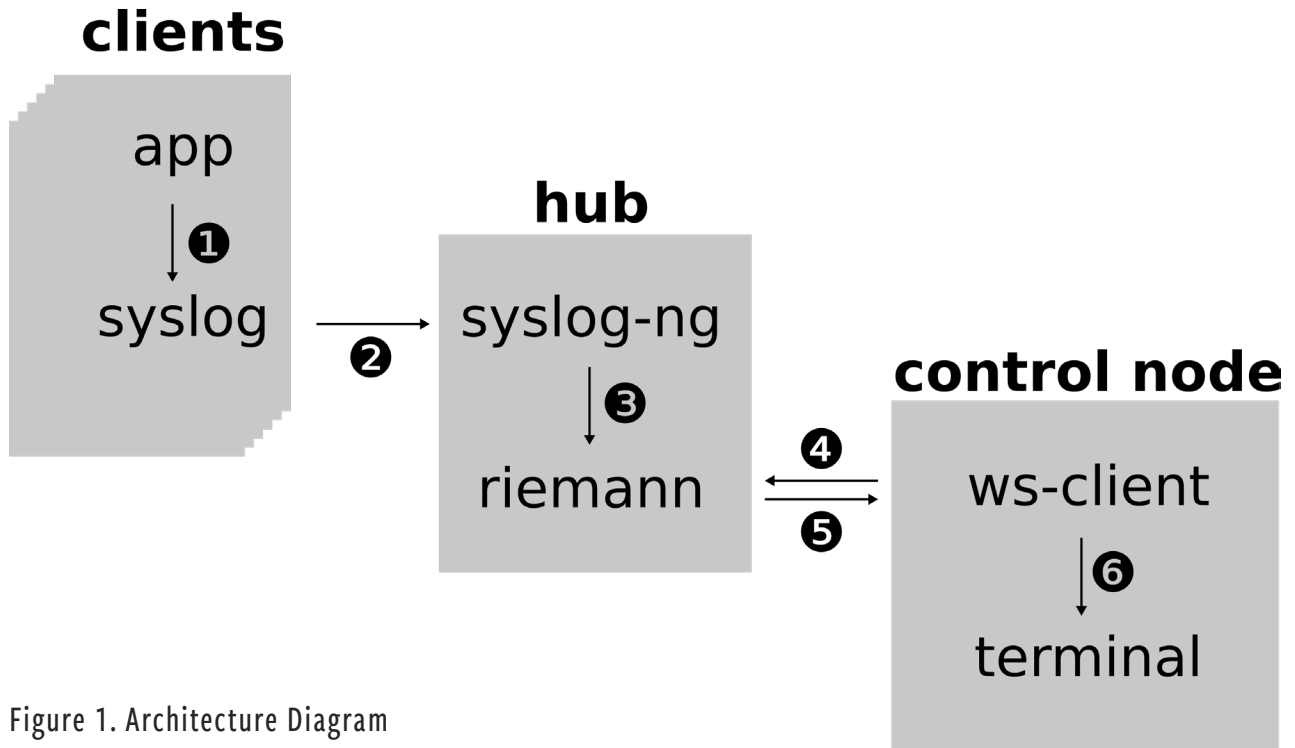


Figure 1. Architecture Diagram

5. The *riemann* daemon on the *hub* parses the query and starts forwarding matching events to the *control node*.
6. The *control node* parses incoming events and displays them on your terminal in real time.

The whole process, from step 1 to 6, usually takes less than a tenth of a millisecond (three sigma), even if tens of thousands of events happen per second.

From the user's perspective, the workflow steps are the following:

1. **ssh** to the control node.
2. Run the CLI with query as argument.
3. Read messages on the terminal.

Syslog-ng acts as a syslog-forwarder to riemann. Riemann acts as a real-time synchronous event publisher and subscription manager. It can push events matching a certain query using a websocket, for instance, to a command-line client or web browser.

Riemann Queries in a Nutshell

The query must be in riemann's domain-specific language, which is very simple but quite strict. Basically, you have to remember that riemann events have tags and attributes. You can query tags using the **tagged "foo"** pragma and attributes with **key = "value"**. You can combine conditions using **and** and **or** operators, and use the special wild-card character "%" in attribute expressions in the following form:

```
MESSAGE =~ "%quick brown fox%"
```

You can learn more about queries on the [Riemann website](#). You could, for instance, subscribe to all events having a syslog priority of "warning":

```
PRIORITY = "warning"
```

Or subscribe to all events:

```
true
```

Or match events from a given IP address:

```
HOST_FROM = "172.18.0.1"
```

Setup

Clients:

On the *clients*, you'll need to configure the local syslog daemon to forward all events to the *hub*.

The precise method depends on the syslog application you use on the *client*. If you are using legacy syslogd or rsyslog, add the following line to your (r)syslog.conf file:

```
* @hub.example.com
```

If you are using syslog-ng, add the following lines to your syslog-ng.conf file:

```
destination d_hub {  
    network(  
        'hub.example.com',  
        transport(udp),  
        port(514),  
        flags(syslog-protocol)  
    );  
};
```

And, don't forget to add the new destination to your existing log path (see the Configuration section for an example).

hub:

On the *hub*, you have more work to do, as you'll be installing and configuring both syslog-ng and riemann. Make sure to download and install at least the versions listed earlier.

Installation

The procedure to install syslog-ng highly depends on the operating system you are using. On recent Debian-based GNU/Linux distributions, chances are the distribution packages will contain a recent enough version:

```
apt install syslog-ng-mod-riemann
```

If your distribution doesn't provide the required version, the [syslog-ng project's home page](#) has pointers to download packages for various platforms. Last but not least, there's always the option of building your own binaries using the source code available on [GitHub](#). If you decide to go down that path, make sure to enable the riemann destination in the compilation options.

Installing riemann is just a matter of downloading the package from its website and grabbing a copy of a Java Runtime Environment (JRE). On Debian, the most straightforward option is to install [openjdk-8-jre-headless](#). You can also build riemann from source (see instructions on its [GitHub page](#)). On Debian/Stretch, you could do the following:

```
apt install openjdk-8-jre-headless
wget https://github.com/riemann/riemann/releases/download/
↳0.3.0/riemann_0.3.0_all.deb
dpkg -i riemann_0.3.0_all.deb
```

Configuration

The syslog-ng configuration given here is the minimum required for the task at hand. It doesn't technically conflict with the existing syslog-daemon implementation, unless it's already listening on the UDP port 514. However, on Debian-based distributions, installing syslog-ng will uninstall rsyslog, because the packages conflict with one another.

In this light, you'll add a drop-in file, `/etc/syslog-ng/syslog-ng.conf.d/riemann.conf`, that syslog-ng will include in the main configuration file. That way, it won't interfere with the configuration file provided in the distribution:

```
# syslog listener definition on *:514/udp
source s_syslog {
    network(
        ip('0.0.0.0')
        transport(udp)
    )
}
```



```
        port(514)
        flags(syslog-protocol)
    );
};
# riemann destination definition
destination d_riemann {
    riemann(
        server('127.0.0.1')
        port(5555)
        type('tcp')
        ttl('300')
        state("${state:-ok}")
        attributes(
            scope(all-nv-pairs rfc5424)
        )
        tags('syslog')
    );
};
# log path
log {
    source(s_syslog);
    destination(d_riemann);
};
```

Ensure that the `/etc/syslog-ng/syslog-ng.conf` file includes the following line; otherwise, `syslog-ng` will ignore the `/etc/syslog-ng/syslog-ng.conf.d/riemann.conf` file:

```
@include "/etc/syslog-ng/conf.d/*.conf"
```

The above configuration defines a syslog listener on standard UDP port 514, a riemann destination and a log path connecting the two. Refer to the [syslog-ng documentation](#) for any details on the syntax used here.

The riemann configuration is given in its entirety. You simply can replace the shipped `/etc/riemann/riemann.config` with the following:

```
; Configure logging
(logging/init {:file "/var/log/riemann/riemann.log"})

; Disable riemann's internal instrumentation
(instrumentation {:enabled? false})

; Listen on the local interface over TCP (5555), websockets
; (5556), and server-side-events (5558)
(let [host "0.0.0.0"]
  (tcp-server {:host host :port 5555})
  (ws-server  {:host host :port 5556})
  (sse-server {:host host :port 5558}))

; Expire old events from the index every 5 seconds.
(periodically-expire 5)

; Index all events with a default time-to-live of 60 seconds
(let [index (index)]
  (streams
   (default :ttl 60
            index)))
```

This configuration sets up three listeners:

1. Port 5555 will receive events from `syslog-ng` in `protobuf` format.
2. Port 5556 will listen for `websocket` subscriptions.
3. Port 5558 will listen for `server-side-event` subscriptions.

It also disables riemann's instrumentation service, so you're not confused with internal messages and can focus only on syslog.

Refer to riemann's resources on the website—especially the how-to section for details on the configuration syntax.

Now that both syslog-ng and riemann are configured to your needs, check the configurations for errors:

```
syslog-ng -f /etc/syslog-ng/syslog-ng.conf -s  
riemann test /etc/riemann/riemann.config
```

If both return without errors, (re)start the services:

```
service riemann start  
service syslog-ng restart
```

Control Node

The last thing you need to connect all the dots is the command-line interface that will let you `tail -F` all logs from all the *clients*. There are a number of options here: the CLI you need should support either websockets or server-side-events. Both are technologies borrowed from the web that allow the web server (Riemann in this case) to push data to the client, instead of the client pulling.

You'll be using websockets, as existing software tends to be more generally available. There is a convenient [Python package](#) that works right out of the box:

```
pip install websocket-client
```

Alternatively, you also can use a [Node.js implementation](#):

```
npm install -g wscat
```

Now, subscribe to the syslog flow:

```
# subscribe to all events (query 'true')
wscat --connect 'ws://hub.example.com:5556/index?subscribe=
↳true&query=true'
# or
wsdump.py -r 'ws://hub.example.com:5556/index?subscribe=
↳true&query=true'
```

Note that you may have to URL-encode the query:

```
# subscribe to events matching the query 'PRIORITY = "warning"'
wsdump.py -r 'ws://hub.example.com:5556/index?subscribe=
↳true&query=PRIORITY+%3D+%22warning%22'
```

Let's push some events to it by crafting a syslog message from another shell:

```
logger -d -n hub.example.com -p 4 -t foo bar baz
```

On the WS CLI, you immediately should see:

```
{"host":"172.18.0.1","service":"test","state":"ok",
↳"description":null,"metric":null,"tags":["syslog"],"time":
↳"2018-04-10T13:36:04.787Z","ttl":300.0,"DATE":"Apr 10
↳15:36:04","HOST":"172.18.0.1","FACILITY":"user","MESSAGE":
↳"bar baz",".SDATA.timeQuality.isSynced":"0","HOST_FROM":
↳"172.18.0.1","SOURCE":"s_syslog",".SDATA.timeQuality.tzKnown":
↳"1","PRIORITY":"warning","PROGRAM":"foo"}
```

If you want to see a more traditional representation of the message (as in `/var/log/messages`), you can pipe the client's output through the [jq utility](#) in the following way:

```
wsdump.py -r [...] | jq -r '"\(.time) \(.HOST) \(.PROGRAM)
↳\(.MESSAGE)"'
```

In which case, you'll see:

```
2018-04-10T14:04:53.489Z 172.18.0.1 foo bar baz
```

Troubleshooting

To troubleshoot syslog-ng, run it in the foreground in debug mode:

```
syslog-ng -Fdv
```

Although very verbose, the parser and debug messages are extremely valuable when tracking configuration or payload issues.

If needed, feel free to subscribe to the very friendly [official mailing list](#), where many users and also the core developers are active.

Debugging riemann configuration problems can be challenging, especially if you've never programmed in Clojure before. If there is a syntax error, like a missing parenthesis, you quickly can be flooded by Java stack traces. Be patient and try to find the relevant bits in the trace messages.

If that doesn't suffice, there's a very [helpful community on IRC and the mailing-list](#).

What's Next?

Now that you've got a working proof of concept (PoC), there are quite a few things you can do to improve the system. Although I won't go into much detail about those things, here are a few ideas based on our experience at [CCIN2P3](#).

Security

First off, you might want to add a bit of security to the setup. You actually made things quite worse while moving from `ssh+tail -f messages` to the

websocket solution. Anyone on the network now can subscribe to the whole site's syslog, without any authentication. Luckily, there's a simple solution to this: set up a reverse proxy in front of riemann's websocket listener. As authentication is very site-specific, I won't cover it extensively here. However, here's a simple example using the [Caddy web server](#) and basic authentication:

```
# /etc/Caddyfile
hub.example.com:5559 {
  tls self_signed
  basicauth / user pass
  proxy / localhost:5556 {
    websocket
  }
}
```

This configuration will listen to the external port 5559 and proxy the traffic to local port 5556 if the user provided correct credentials. This only makes sense if riemann is reconfigured to listen on the local network interface.

On the *control node*, you now can connect to the proxy using basic authentication:

```
wsdump.py --headers 'Authorization: Basic dXNlcjpwYXNz' -r
↳ 'wss://hub.example.com:5559/index?subscribe=true&query=true' -n
# or
wscat -n --auth user:pass --connect 'wss://hub.example.com:5559/
↳ index?subscribe=true&query=true'
```

Note that the Python CLI doesn't support supplying basic auth credentials on the command line, so you need to pass the base64 encoded **user:pass** using an HTTP header.

Another improvement could be to write a higher-level CLI that integrates with your local organization's central authentication mechanisms. For example, it

could leverage roles from a central identity directory and apply access control lists. Those could be host-based or even application-based: role A can subscribe to events matching queries X, Y and Z.

Stream Processing

Both software suites installed on the hub can be leveraged further to filter, aggregate and even correlate syslog messages.

Although syslog-ng can do this in a more traditional fashion by using configuration elements like filters, parsers and template functions, riemann on the other hand gives you full control over the event flow. In fact, its configuration file is code that will be compiled, so you can do virtually anything. One of the most common usages in the wild for both software packages are structuring incoming data. Although you saw that syslog events already feature some structure in the form of key/value pairs, both riemann and syslog-ng can help you extract or add additional features to your events. Those will help you filter the live stream of events and answer real questions.

Web App

There is a web interface ([riemann-dash](#)) that takes advantage of riemann's subscription mechanism. It can display events in textual, grid or even graphical form, and it's invaluable when you want to monitor changes in real time in a distributed application.

Resiliency

Another caveat of this PoC is that this *hub* is a single point of failure (SPoF). You could do the following to improve the situation:

- Install syslog-ng on all the clients.
- Configure all syslog-ng instances to send the logs to multiple riemann servers.

- Write a high-level CLI that uses load-balancing to subscribe to the riemann servers' streams.

Dockerfile

For your convenience, a GitHub repository containing the means to build a [Docker container for the solution described in this article](#) is at your disposal. It includes the steps on how to build, run and use the container. ■

Fabien Wernli (faxm0demi/faxmodem on GitHub, Twitter and Freenode) has been administering Linux clusters at the Computing Centre of the National Institute of Nuclear Physics and Particle Physics (CC-IN2P3) for 15 years. Among other things, he is an expert on performance-data monitoring and infrastructure management.

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

Resources

- [Utilization Saturation and Errors \(USE\) Method](#)
- [Elasticsearch](#)
- [Protocol Buffers](#)
- [Riemann Website](#)
- [syslog-ng Project's Home Page](#)
- [syslog-ng on GitHub](#)
- [Riemann GitHub Page](#)
- [syslog-ng Documentation](#)
- [WebSocket Client for Python](#)
- [Node.js WebSocket Implementation](#)
- [jq Utility](#)
- [syslog-ng Mailing List](#)
- [Riemann Support and IRC](#)
- [Centre de Calcul de L'IN2P3](#)
- [Caddy Web Server](#)
- [riemann dash Web Interface](#)
- [Distributed tail -f /var/log/messages Docker Container](#)

Further Reading:

- *The Art of Monitoring* is a fine book that covers various aspects of monitoring. Luckily, it includes a free complete section on riemann.
- The [Just enough Clojure to work with Riemann](#) section on riemann's website tries to address the steep learning curve of the "configuration syntax".
- The syslog-ng website features multiple [whitepapers](#) with various use cases, including sections on how to parse and structure your syslogs.

Taking System Monitoring to the Next Level: an Interview with Scalyr CEO Steve Newman

As computing ecosystems become more complex, monitoring and analyzing those often disconnected moving parts becomes increasingly challenging.

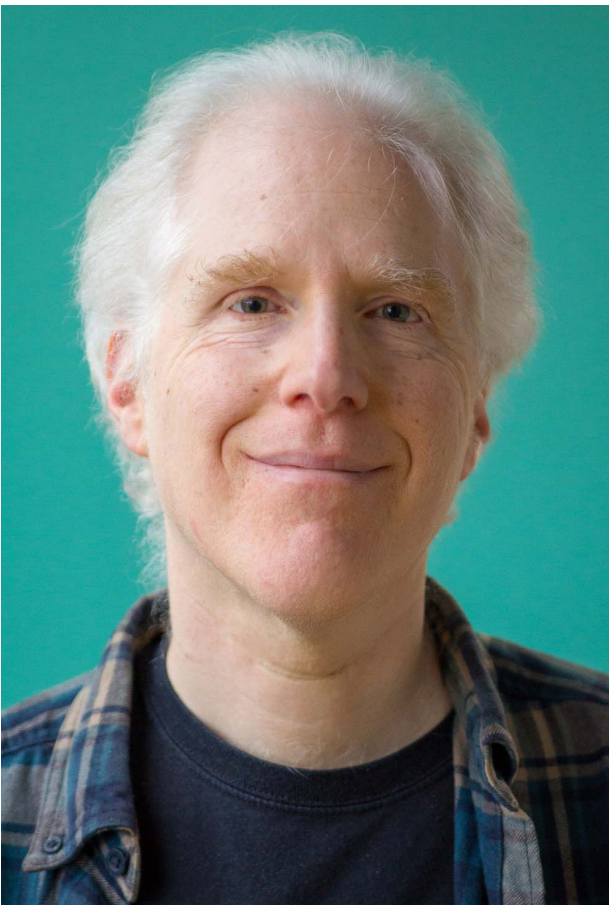
By Petros Koutoupis

Today's data center has evolved from a single supplier producing and selling all-in-one offerings, such as the days when EMC, NetApp, HP or even Sun owned your data center and you chose a vendor and stuck with it. Those same vendors provided you with the required tools to monitor, analyze and troubleshoot their entire stack.

Shifting focus to the present, the landscape now appears to be quite different. Instead, you will find environments of mixed offerings provided by an assortment

of vendors, both large and small. Proprietary machines work side by side with off-the-shelf commodity devices hosting software-defined software. Half of your applications may be hosted in virtual machines over a hypervisor or just spun up in a container. How does a modern data-center administrator or DevOps professional manage such an environment?

An assortment of platforms and frameworks exist that provide such capabilities, but they're not all one and the same. In some cases, those same tools will need to be coupled with others to produce something useful (for example, ELK: Elasticsearch + Logstash + Kibana). Unfortunately, this arrangement just adds to the complication and frustration when attempting to diagnose or discover problems in your computing environment.



Steve Newman, Scalyr CEO

Putting an end to this level of complexity, one company stands out among the rest: Scalyr. Scalyr develops and offers a complete suite of server monitoring, log management, visualization and analysis tools, which integrate with cloud services. I recently had the pleasure of chatting with Scalyr CEO Steve Newman.

His is not a household name, like Steve Jobs or Bill Gates, but you will be familiar with his work and contributions to cloud-enabled technologies. Although this is likely to change with Scalyr, Steve is best known for his work with Writely, a technology that later was acquired by Google and relabeled as Google Docs. In our conversation, Steve and I took the opportunity to discuss Scalyr, its solution and the problem it solves.

Petros Koutoupis: Tell me a bit about yourself. Who is Steve Newman?

Steve Newman: I am an engineer by both training and background and have spent most of my career in the startup environment. This is because I enjoy building things. I was at Google for a number of years following an acquisition, and while the experience itself was great, the startup bug in me drove me to Scalyr.

PK: So, now you founded a company called Scalyr. Please tell us, what is Scalyr?

SN: Scalyr is a log management platform for engineers responsible for software development. We collect logs from applications, services, containers and systems, and make that data available to help engineering teams track down problems and generally manage the complexity of modern development and operations.

PK: And why? What problem(s) does your product solve?

SN: Traditional log management tools are complex and often very slow at scale. This leads to a “gatekeeper” approach to log management, where only a select few acquire the expertise (and have the patience!) to access this critical data. Logs become a tool of last resort, hindering the team’s ability to rapidly or proactively address issues.

My co-founder Steven Czerwinski and I first experienced this problem ourselves at Google. We were leading an infrastructure project supporting Google Docs, Drive, Photos and other related applications. There were a lot of moving parts, and the engineering team spent close to half of its time simply tracking down problems. We started Scalyr in 2011 to create the tool we wished we’d had at Google—one that would allow us to make sense of the flood of telemetry data and quickly understand why a complex system is misbehaving.

PK: How does Scalyr work?

SN: We're a fully managed SaaS solution. Logs are sent to our centrally hosted search cluster, using our agent or an array of API integrations. Engineers then use our web app (or APIs) to analyze, visualize and explore the logs.

The critical component is the back end. We've built the back-end software stack from scratch, optimizing for the data access patterns that arise in log management. Some interesting aspects of our approach are:

1) Unlike other log management solutions, we don't use indexes. Keyword indexes are optimized for finding the "best ten matches", in a corpus comprised of slowly evolving, human language text such as web pages or product descriptions. Log management use cases are very different, with small units of text (individual log messages), constantly updated, full of record IDs and other non-words that balloon the vocabulary size. Most important, log management queries generally visualize a complete data set, rather than stop after ten high-ranking results. Keyword indexes don't help much there, and they are complex, expensive to maintain and often impose multi-minute ingestion delays.

We've taken a much simpler approach, building a streamlined, columnar data store that's optimized for log data. The basic idea is that we just store logs and scan them, like good old-fashioned **grep**. We then use a lot of tricks to minimize the amount of data that needs to be scanned; for instance, when querying specific fields of a log, the columnar data layout means that we need to scan only those fields.

2) We process queries one at a time, globally. This allows each customer to use our entire search cluster, with aggregate search performance of 1.5 terabytes per second. It's fast enough (96% of queries complete in less than one second) that queries almost never wait in line—we finish each query before the next one arrives. The nice thing about this approach is that there's an economy of scale: as our customer base grows, performance increases.

3) We've built a separate back end for repetitive queries, such as those used in dashboards or alerting rules. This part of the system is based on a time series database, with a custom time series for each query. The ingestion engine automatically updates these time series as log messages arrive. This means we don't need to execute queries to display a dashboard or evaluate an alerting rule—the relevant data has been precomputed in the time series. In database terms, we're automatically creating materialized views where needed.

PK: What makes Scalyr stand out or competitive with existing solutions?

SN: Speed, simplicity and scalability.

Speed was central to our mission from day one. Logs are a massively useful, detailed data source, but when it takes minutes (or longer) to run a query, engineers avoid using them. We satisfy most queries in a fraction of a second. We also ingest data in real time: new logs are available for querying within a few seconds.

Simplicity goes hand in hand with speed, which is best measured as the time from a question in an engineer's head to an answer on his or her screen. The fastest back end in the world is of little use if you spent five minutes wrestling with the query language. We rely on our performance, as well as our ability to parse logs and extract structured data on ingestion, to provide a set of visual exploration tools that allows engineers to get answers without becoming query language experts. There's a query language, but you can dive in without knowing anything about it.

Finally, customers often choose us for our scalability. We continue to work well not only as server count and data volume increase, but as a team grows. Scalyr works just as quickly whether it's three or 1,000 people looking at logs. This helps teams move away from the gatekeeper model of traditional log management to the concurrent, collaborative engineering model that modern organizations are increasingly adopting.

With Scalyr, companies for the first time have a log management platform that

does not charge ruthless licensing fees for new users, involve long ramp-up times or require learning arcane query languages. It's meant for teams who need to move fast.

PK: Who would benefit from using Scalyr?

SN: Our sweet spot is organizations where the application is critical to the business. From B2B software to online retailers, dating platforms and media companies, every business' competitive advantage increasingly is the technology stack and the speed at which that stack can evolve. Scalyr is critical to enabling that. Some of our customers include NBCUniversal, OkCupid, Zalando and ReturnPath.

Within those organizations, the primary Scalyr user usually comes from engineering or DevOps. But Scalyr is simple enough to use that we often see usage spread to other roles like support, which can search logs to track down specific client problems. Some of our customers have upwards of 1000 individuals with Scalyr logins. Typically, half the users are active on a weekly basis, which is huge engagement compared to traditional log management platforms.

PK: How easily does Scalyr integrate into current production environments?

SN: It's our mission to meet customers where they are, so we support many different models. Some run their own servers or virtual servers. Some are on Kubernetes, while others are serverless. Regardless, setup first involves retrieving logs. The most common way of doing this is with a lightweight agent that customers can install as a container, sidecar or whatever they need. We also have API integrations to retrieve logs directly from the wide array of cloud services in use today.

Once the logs are flowing in, you're off and running, but an important further step is to set up parsing rules. This allows us to extract structured data, unlocking the full power of the analysis and visualization tools. To make this as

easy as possible, we've built three generations of parsing engines. The current engine is so easy to use, we've literally put a button in the product that tells our support team to set up the rules for you. Of course, being engineers, many of our customers prefer to do it themselves.

Conclusion

With today's internet, the "app" increasingly *is* the business (think of Uber, Airbnb, Amazon and so on), and getting to the bottom of downtime is crucial. This process is made more difficult than ever as the system or code is increasingly distributed with the use of containers, serverless and other technologies. That is where Scalyr comes in with its log analysis platform. It is crazy fast and easy to use. To learn more about this wonderful product, visit <https://www.scalyr.com>. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior platform architect at IBM for its Cloud Object Storage division (formerly Cleversafe). He is also the creator and maintainer of the RapidDisk Project. Petros has worked in the data storage industry for well over a decade and has helped pioneer the many technologies unleashed in the wild today.

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

**3-4
NOV**

PRESENTING

freenode
#LIVE2018

20TH ANNIVERSARY CELEBRATION

BRISTOL, UK

**OPENS AT
9AM**

The freenode project celebrates its 20th anniversary this year at the second annual freenode #live conference

**At We The Curious in Bristol, UK
November 3-4, 2018 — 9am Saturday - 6pm Sunday**

Keynote speakers include Bradley M. Kuhn, Chris Lamb, Kyle Rankin, Leslie Hawthorn and VM Brasseur! More to come...

Registration and call for participation is open now at

[HTTPS://FREENODE.LIVE](https://freenode.live)



Review: the Dell XPS 13 Developer Edition Laptop

A look at Dell's thin and sleek XPS 13 Developer Edition laptop that now ships with Ubuntu 18.04 LTS pre-installed.

By Petros Koutoupis

Canonical recently made an official announcement on its company blog stating that the Dell XPS 13 Developer Edition laptop (that is, Project Sputnik) now ships with Ubuntu 18.04 LTS (Bionic Beaver) pre-installed. Upon reading this, I quickly reached out to Dell asking to review the laptop. I'm a Linux developer, and when a developer edition laptop is marketed with Linux pre-installed, I *need* to experience it for myself. The laptop eventually arrived, and like a child on Christmas morning, I excitedly pulled the device out of the box and powered it up for the first time.

This is a pretty rock-solid notebook. The device is very light and easy to carry—meaning, it's mobile (which is very important in my book), thin and sleek. Not only does the device look good, but it also performs very well.

General Specifications

In my possession is the 7th generation of the Dell XPS 13 Developer Edition laptop. This generation ships with an Intel Core i7 8th Gen microprocessor. It is a four-core, eight-threaded (hyperthreaded) i7-8550U CPU operating at a 1.8GHz frequency. With this configuration, the system itself reports eight CPUs. The system is installed with 16GB of RAM.

First Impressions

Upon first boot, you're greeted with a Dell welcome screen followed by a generic set of Ubuntu-related questions (such as license agreement, keyboard layouts, timezone and so on). Toward the end, you are given an option to create a recovery USB image, which could be very handy one day. If you opt out of creating one, no worries, you can go back and create one at a later time.

The first thing I did, after logging in to my user session for the very first time, was run a software update.

Although this does not at all relate to the quality of the device, I did find it a bit strange that the operating system was pre-installed with both Chrome and Chromium web browsers. I'm not sure why anyone would need both, but they both were there. If you're a Firefox user, you'll need to install it from the Ubuntu Software center.



Figure 1. The Dell Recovery Media Menu

Input Controls

One of the most important aspects of any computing device is its input controls (keyboard, mice and so on). The keyboard feels comfortable—that is, with the exception of the positioning of the PgUp/PgDn keys. Those two keys are very close to the left and right arrow keys. I'm sure, over time, I'll end up getting used to it, but during the course of my review, I constantly and accidentally pressed the PgDn key while navigating my way around in a terminal and text editor. Again, the keyboard feels nice. It's responsive, and the throwback from when you press the keys down feels just about right. Although, *why* is there still a Windows icon for the Super key?! I jest.

I also do appreciate the backlit keypad. You even can toggle the backlight on and off manually. This backlight also fades to off when the keys haven't been pressed in some time. Although, I don't recall what that idle time or timeout is set to.

The touchpad is very responsive as well—maybe a bit too sensitive. I find myself accidentally closing tabs or selecting things I never intended to select. By default, Ubuntu enables natural scrolling among a few other touchpad-specific features.

The notebook comes with a touchscreen bonded to the display. I always find it a bit awkward trying to make use of a touchscreen while maneuvering between a keyboard and a trackpad, but that's just me. However, I do see the value in it when it comes to testing and debugging applications intended for mobile, embedded or web use. I did notice a couple things with the touchscreen:

1. Although it's responsive and calibrated accordingly, not all windows are created equal. I had no issue moving GNOME native windows and applications around the screen, so long as it was *not* touching the edges of the top, bottom and sides (I thought that was weird). But for whatever reason, regardless of where I tried to hold down to drag the window of the Chrome web browser, it just would not budge.
2. Because the physical display extends to nearly the edges of both the top and the sides of the lid, every time I went to adjust the angle/tilt of the screen (from the left side), my large fingers would activate whatever was positioned on the Dock right underneath it.

What I do find a bit frustrating is that I am not able to disable the touchscreen from the Ubuntu Settings panel, unless it is very well hidden somewhere. The only way that I believe this can be done is through the Xorg configuration files.

Input controls overall rating: 4/5

Display

The display is absolutely amazing, and by amazing, I mean *beautiful*. Everything is very crisp and clean. As a software developer, I definitely can appreciate all of the available screen real estate. I live in the terminal, and more specifically, I use my preferred developer tools: vi and grep. It is a 16:9 screen scaling to a 3840 x 2160 resolution (4K). So, needless to say, there's plenty of room for me to open one or more terminal windows on the same screen.

The screen resolution is so large, some applications don't necessarily scale as well as the rest of the desktop environment. You can go to the Settings→Devices→Displays menu and change the value of the Scale field from the default 200% to something more, but some of the other applications may not comply with the changes (a good example is GIMP).

Right before I dove into this area, I went to the Ubuntu Software Center and

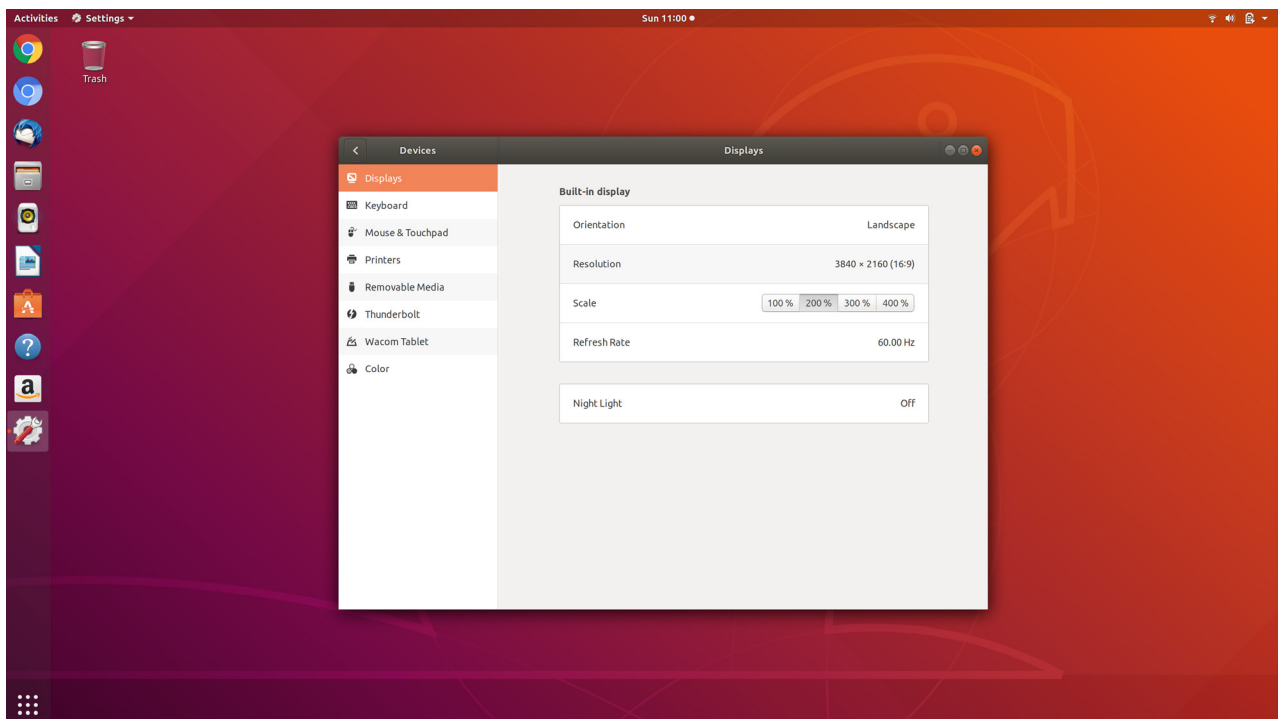


Figure 2. The Display Configuration Menu

installed the graphic editor. The application was loaded almost as soon as the installation was complete, and it was immediately apparent that the icons and options of both trays did not scale like the rest of the operating system.

Display overall rating: 5/5

Audio

I really enjoy the quality of the onboard speakers. I am streaming audio through it while typing this review, and it sounds pretty darn good. Everything is clear and crisp, as one would hope.

The microphone worked as expected, and the folks on the other end of my video chat were able to hear my voice with little to no interference or background noise.

Audio overall rating: 5/5

Power Consumption

Power consumption and battery life can make or break a device intended to be used for mobile computing. If you can't get this right, what's the point? Overall, I am extremely impressed with the power consumption of this device.

To put this into perspective, I was running from battery for 30 minutes while navigating in a browser and writing my notes for this review in Google Docs, and 95% of the battery is still remaining—not bad. Of course, if I were to increase the workload on this machine, that number would quickly drop.

As one would expect, the backlight of the panel dims to help conserve energy, and you also have the option to reduce the brightness or turn off the backlight underneath the keypad. You can do other things to reduce the power consumption as well. For example, if it isn't being used, it may be to your advantage to disable Bluetooth.

Fast-forward a few days: the notebook has been off of the charger for three

days now. I've been routinely going back to the device (and waking it up from hibernation) to use it for minor tasks here and there throughout the course of those same three days. The notebook is still above 60% battery. For a four-core device, this is *really* good.

As an added bonus, you can check battery state quickly without opening up the lid and logging in to your session. The left side of the notebook has a power gauge where by pressing a button, you are given a general and visual approximation.

Power consumption overall rating: 5/5.



Figure 3. The Visual Battery Indicator

Performance

Boot-up time was mostly reasonable. Although I didn't exactly take a stopwatch to it, it felt to me as if I was at the login screen for 15 seconds after I pressed the power button. Now, I'm not entirely sure what's going on under the hood before the operating system begins to load, but in my personal opinion, I was expecting something a bit faster when running off of an NVMe SSD. Either way, even 15 seconds is *not* unreasonable. And unless I reboot after a system update that requires it, I'll never power-cycle my machines.

Waking up from a sleep or hibernate state is also very quick. Although I do not power off my notebooks, I instead close the lid to ensure that my applications and their states are still present when I reopen the lid and reawaken the device. Within a second or two, I am prompted with the login screen.

To test the system's overall performance, I decided to build three separate Linux kernels simultaneously, all different versions:

- 4.14.67
- 4.18.5
- 4.19-rc2

What really impressed me was the speed at which the system was able to untar the archives. For instance, the 4.19-rc2 gzip file was approximately 154M in size and took less than four seconds to uncompress—3.961 seconds to be exact. Not bad!

When it came down to preparing and building these kernels all simultaneously, I was surprised that the base Ubuntu image for a developer machine did not include the developer environments for things like the C library or even the kernel. I needed to install header files from `libc-dev` just to build things in C. Eh, those are minor details.

Back to building the kernels, I gave the exact same config (`x86_64_defconfig`) of the kernel to all three and ran the build simultaneously. All three completed within 25 seconds. Wow! Now, when I parallelize the builds with the `-j` option in the `make`

command, the time dropped to less than ten seconds. I mean, holy crap, wow! This *is* a four-core hyperthreaded machine using NVMe. So, I would expect it to perform, and it does just that. The 16GB of memory obviously helps a whole lot.

Wireless network performance also was good and seemed to be very responsive.

Performance overall rating: 5/5.

External Device Support and Peripherals

Now, aside from a single microSD slot and a headphones jack, everything else is USB-C. This seems to be the trend nowadays, and it makes sense now that newer devices to market all support the protocol. Even the power adaptor connects over USB-C and will occupy a single USB-C port.

But, what does this mean for expandability? It means you are required to buy adaptors and expanders to connect more devices, which includes external monitors. Dell does provide the customer with a single USB-A to USB-C adaptor, and in most cases, that may be enough. This way, you can connect your USB thumbdrive or that external hard drive—or maybe even that one developer microcontroller board to communicate with its JTAG interface over serial.

Some folks may have issues with this, but honestly, I personally don't mind. The protocol is much more superior than its predecessors, and it can sustain more connections for those that require it. Truth be told, you need to give Dell some credit here. Unlike some of its competitors, Dell is at least easing your transition from USB-A to USB-C by providing that adaptor. Those same competitors will sell you a similar adaptor for \$60 or more on their websites.

I also am pleasantly surprised that the operating system went out to my local network and found my print server with zero effort on my part. As soon as I went to printer settings, it already was listed and ready to go.

The biggest problem I have is with the webcam. The webcam is oddly positioned

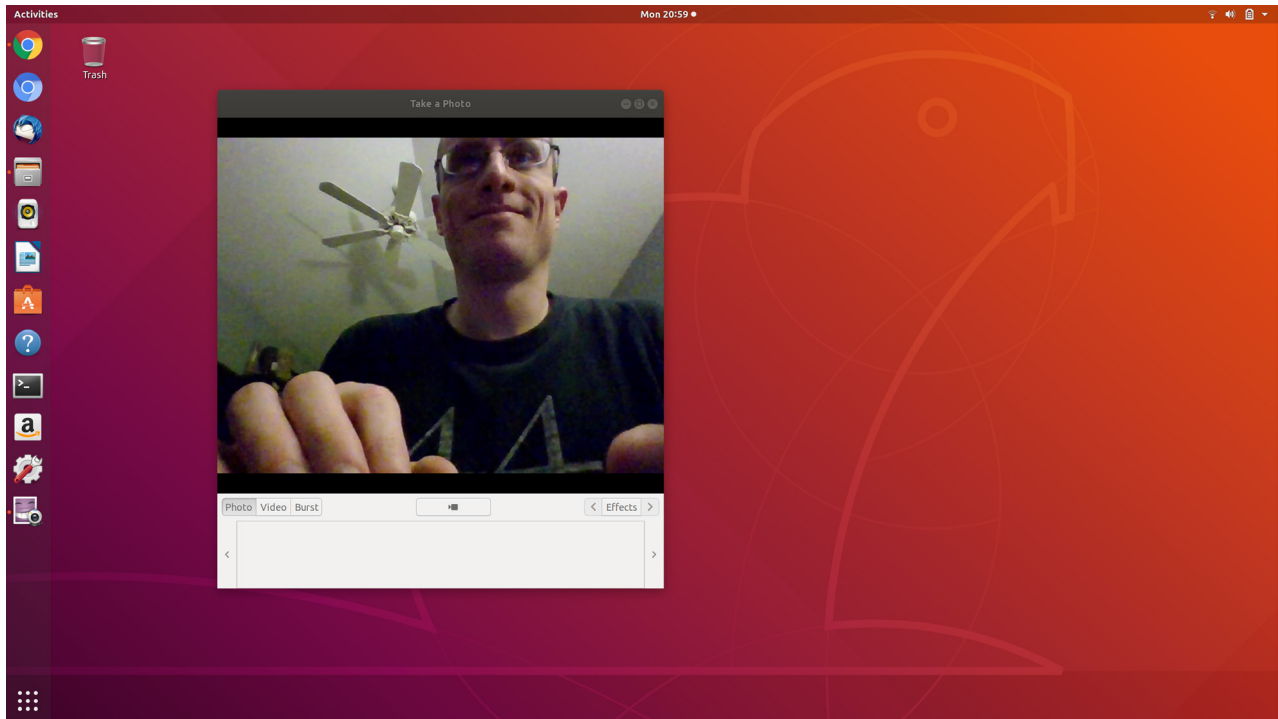


Figure 4. The Odd Placement of the Webcam

beneath the display (instead of at the top, as is typical). This means that in order to capture your face, you need to adjust the tilt of your screen, which leads to an awkward display of your face. And if you're on a video-conference call, everyone else will have a nice view of your fingers typing away at the keyboard. I hope future iterations of the XPS line will address this design issue.

External device support and peripherals overall rating: 3/5.

And the Rest

The software integration between Dell and Ubuntu seems to stand out a lot—and I mean this in the most positive way. System updates include Dell-specific hardware/firmware patches. This is huge.

I did experience a couple quirks or issues throughout this write-up. Let me state the following disclaimer before getting into the details of those quirks:

The problems I'm about to describe have nothing to do with the Dell hardware. They are focused on the Ubuntu operating system running on that hardware. I've been observing this exact same behavior in virtual machines (if applicable) and on other non-Dell physical devices.

Applications would sometimes randomly crash. I've observed this with both the Update Manager and, again, with Nautilus. I'm not sure if it's the version of Nautilus supported on Ubuntu 18.04, but regardless of the environment that I use with this particular distribution and release, the Nautilus file manager application routinely crashes and generates a "report to Canonical" message. The problem continues to occur with the latest Ubuntu package updates.

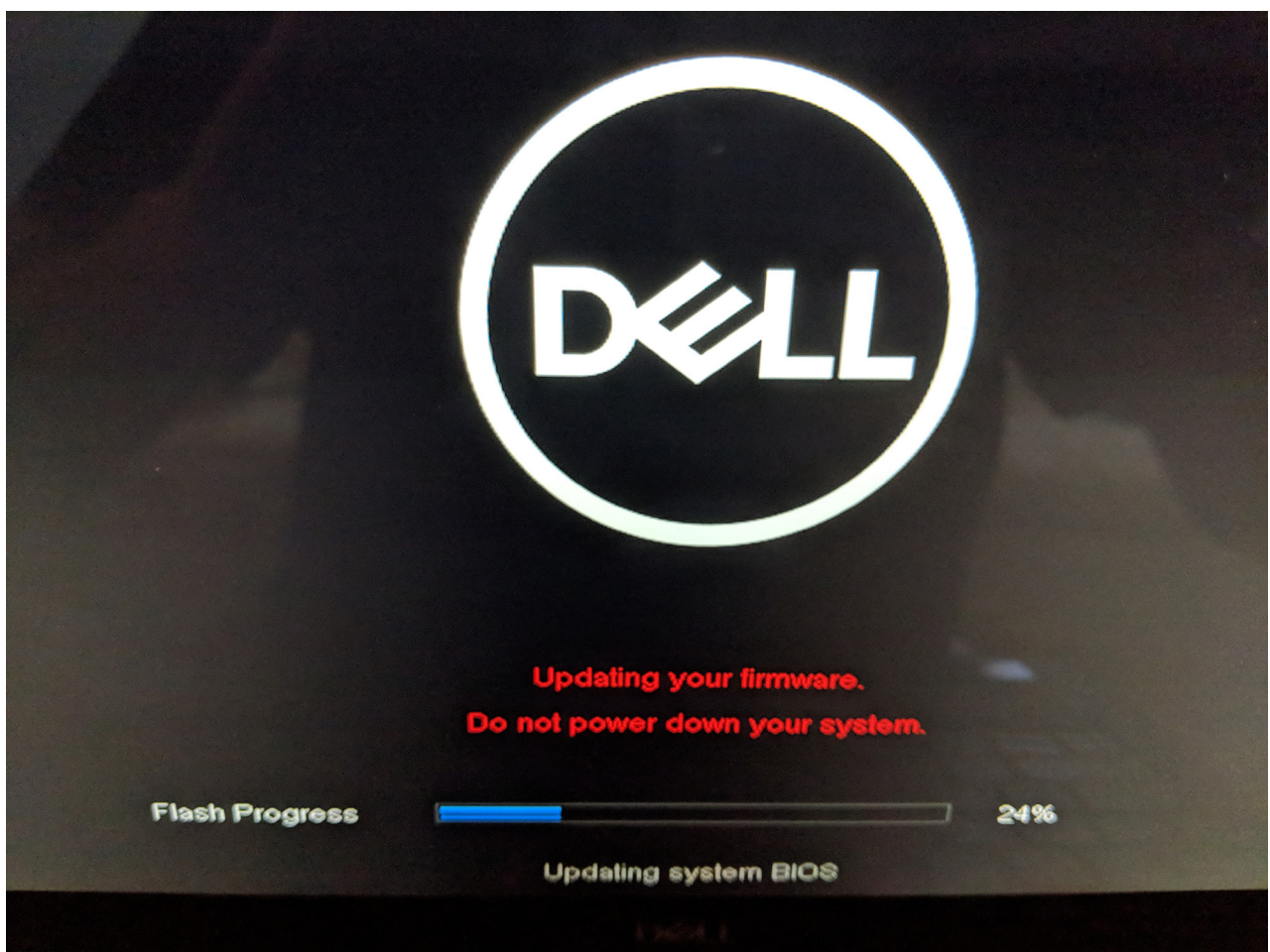


Figure 5. The System Doing a Hardware Update after Rebooting from a Software Update

Another quirk relates to the almost-never-resolved Linux problem: hibernate. To clarify, by default, Ubuntu places your machine in a sleep state when it idles or if you close the lid of your laptop. After some time, the machine transitions from sleep to hibernate mode.

Sleep saves your current state into memory and places your peripherals into a low power mode. In hibernate mode, the saved state will be placed to disk, and the device will be powered down completely. Ubuntu calls this mode “suspend”.

After six attempts to wake up the device from hibernation, only five were successful. In that one case, I was not able to bring the machine back up, and I was greeted by a completely black screen with its backlight lit up. After spending four minutes pressing every key and eventually pressing the power button hoping it would do something, I gave up and held down the power button to force a system shutdown and restarted it.

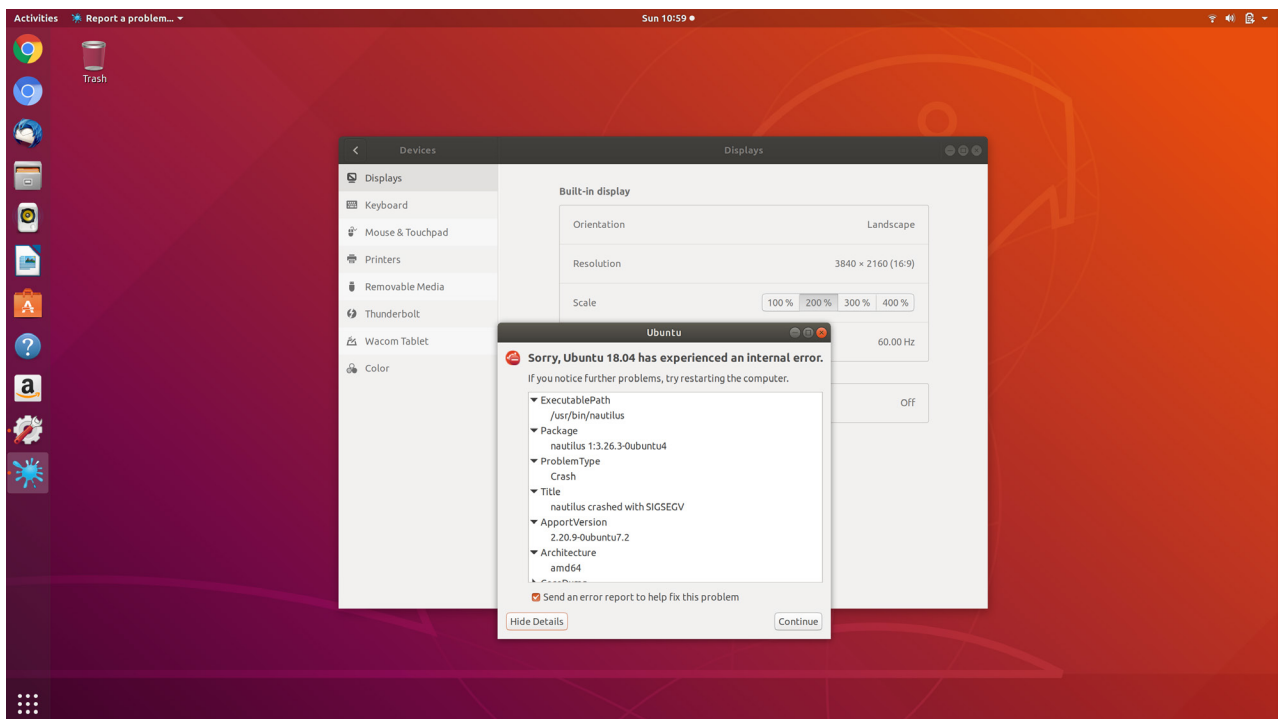


Figure 6. A Reoccurring Software Crash in the Nautilus File Manager

In that same one case, I did do something different. I closed the lid while plugged in. I unplugged the device some time later and attempted to reawaken it. I wonder if I hit some corner-case software bug handling this transition.

Anyway, you can configure or customize the operating system's suspend settings. For instance, you can extend the time it suspends from sleep or just disable it completely for when the device is plugged in and/or on battery power.

Software integration overall rating: 4/5.

Final Thoughts

Overall, I had a very positive experience with the 7th generation Dell XPS 13. It's a powerful machine and fully capable of handling all sorts of developer workloads. And if used in a professional environment, it's very mobile as well. You can carry it from conference room to conference room and resume your work with little to no disruption. Ubuntu is well integrated with the machine, and it shows. You can't ask for more in a developer's laptop. I definitely consider this device to be well worth the investment. ■

Petros Koutoupis, *LJ* Editor at Large, is currently a senior platform architect at IBM for its Cloud Object Storage division (formerly Cleversafe). He is also the creator and maintainer of the RapidDisk Project. Petros has worked in the data storage industry for well over a decade and has helped pioneer the many technologies unleashed in the wild today.

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

Chrome OS Stable Channel Gets Linux Apps

How to get started with Linux Apps for Chromebooks.

By Philip Raymond

After months of user testing in developer and beta channels, the Crostini project at Google finally delivered the goods: Linux apps for most users of Chromebooks in the stable channel—definitely worth the wait. While this still is aimed primarily at developers using Chromebooks, I think there's a good chance these Linux apps will be

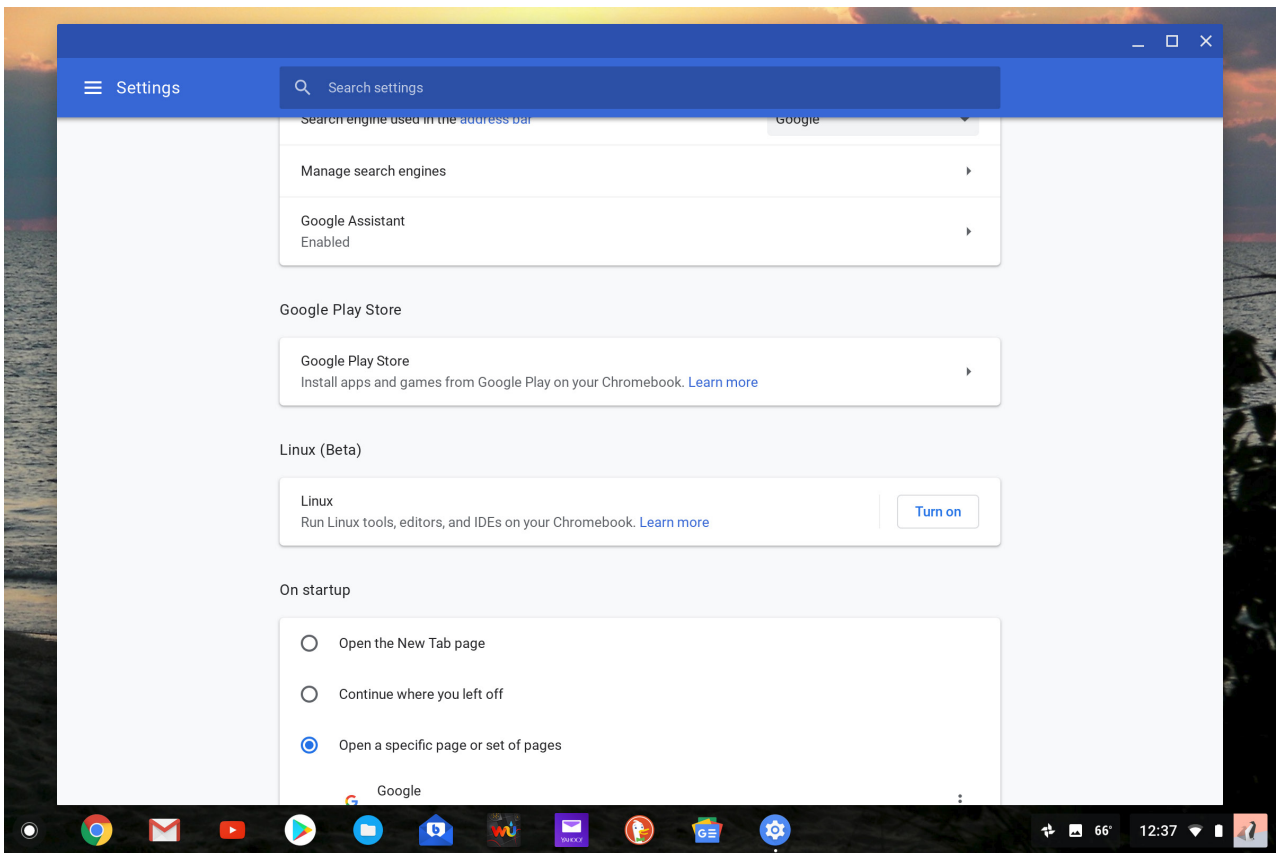


Figure 1. Linux Apps Option

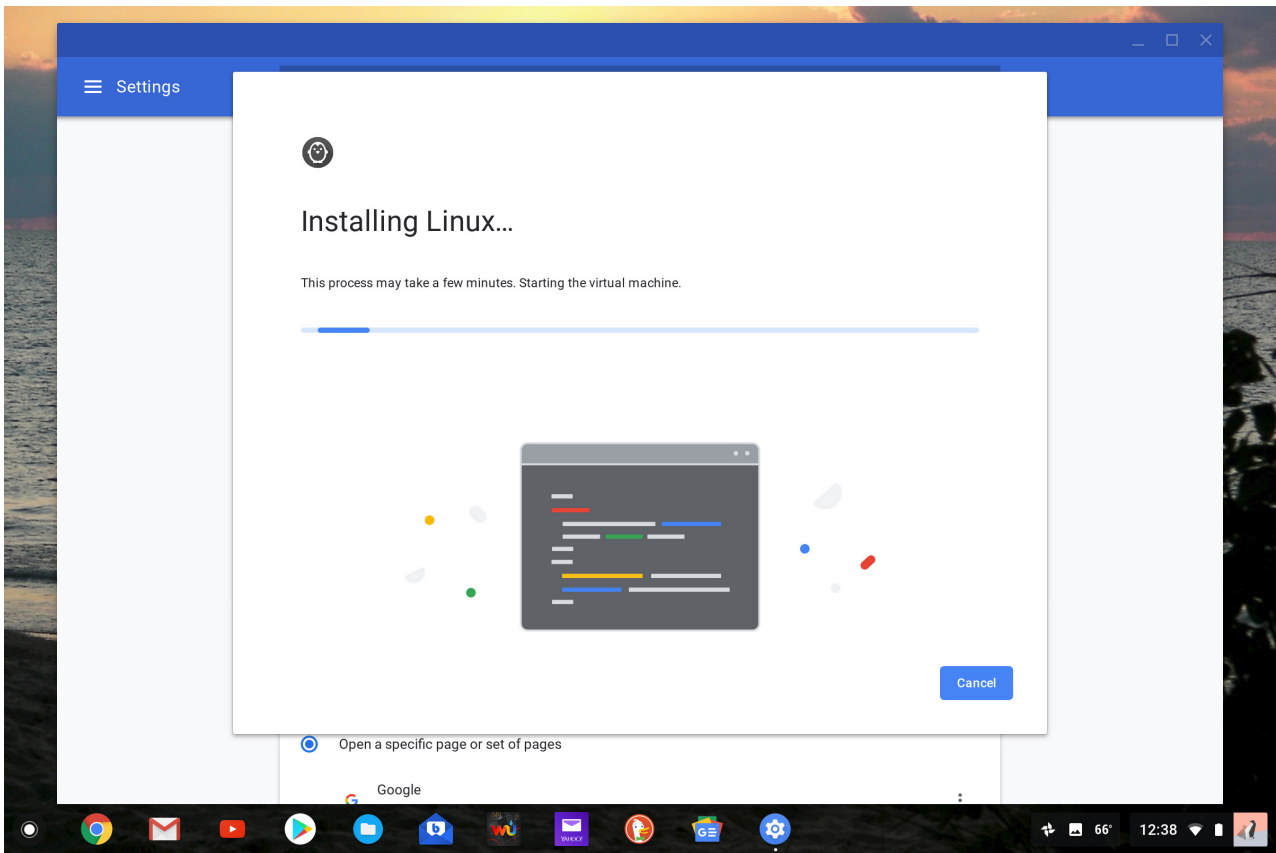


Figure 2. Installing Linux Apps

used and enjoyed by the general public using Chromebooks as well. There's still a bit of a learning curve to overcome before that possibility is realized, but if you already are a user of any Linux distro, it will feel very familiar. Here's an overview of how to install it and what to expect afterward.

After getting the update to version 69, go to Settings and scroll down a bit, and you'll see the option to turn on Linux apps. Figure 1 shows this first step. Note that this isn't available on all Chromebooks; if you're using an older one, you'll have to wait a while before this function is available. If you don't see the option to turn on Linux apps, your Chromebook currently lacks that functionality. But, if you have a Chromebook produced in the past two years, you probably will see the option.

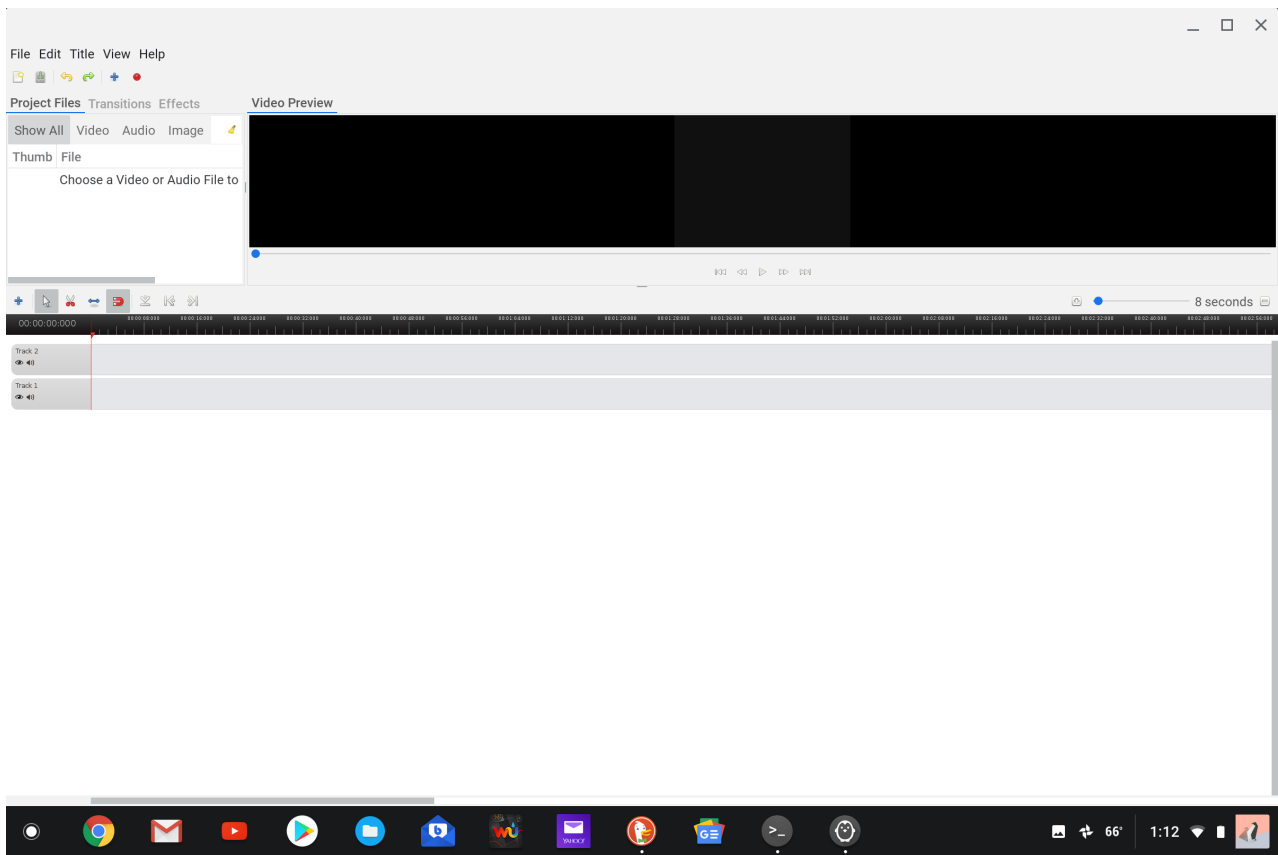


Figure 3. OpenShot

After it's done installing, you'll see the terminal appear. From here, you can do as you would with any terminal. I chose to `sudo apt-get install` the GIMP, Open Shot, Handbrake, Firefox and the GNOME Software Center, which I used to download and install Audacity. The GNOME Software Center provides an easy-to-manage GUI method of finding the more popular Linux apps, but if you prefer the terminal method of using `apt-get install`, that works just as well and provides more app choices than the GNOME Software Center.

One more thing to note about the GNOME Software Center is that you likely will not see any apps in it after first installing it. You need to reboot first before the apps appear.

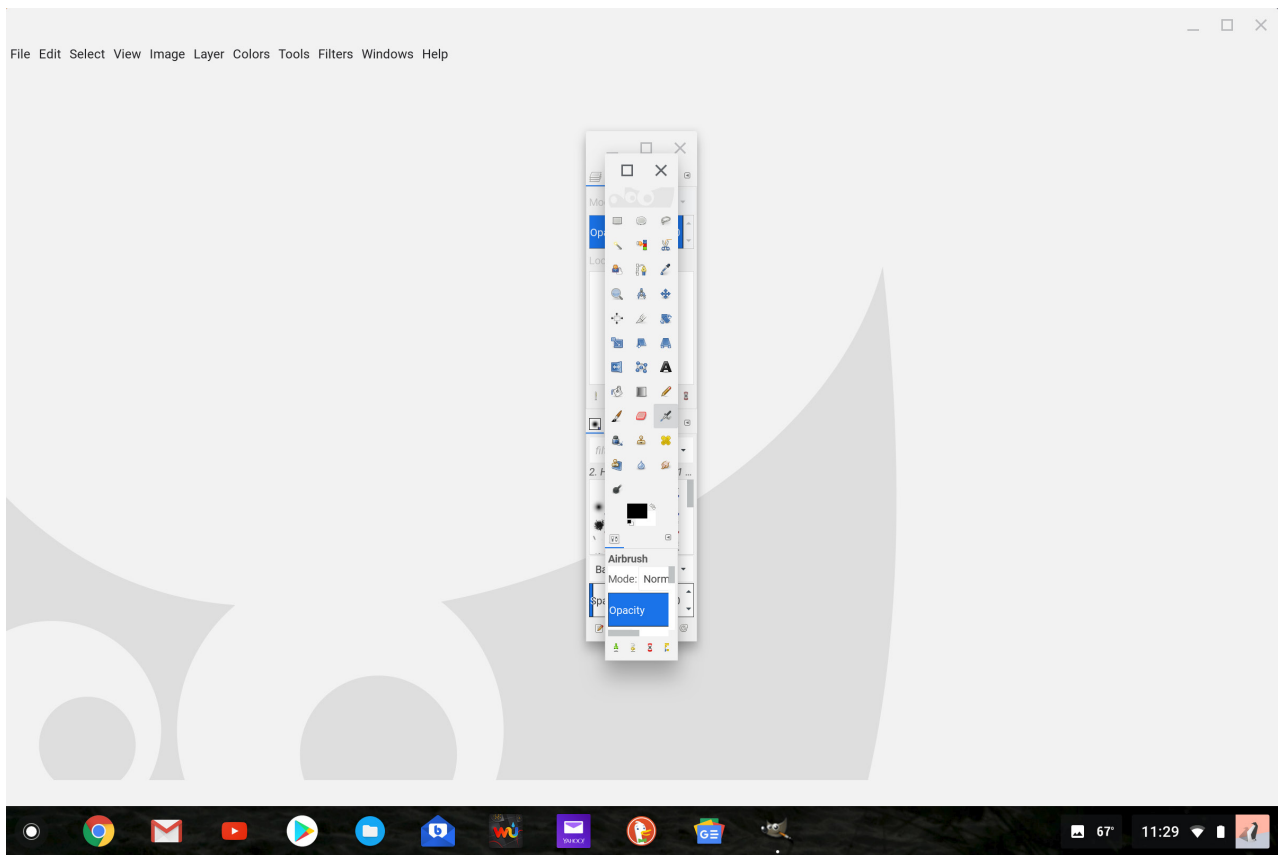


Figure 4. The GIMP

If you want to run Firefox on a Chromebook, there are actually two ways to do it. One way is to download and install Firefox from the Google Play Store as an Android app. Now with Linux apps via Crostini, you also can download and install it from the terminal using `apt-get install`, but it needs to be the extended support release version, Firefox-ESR.

Figures 3–5 show some of my installed apps up and running.

File management for Linux apps is available in the Files folder—on the bottom left side below Play files, you'll see Linux files. This folder is where all files created by a Linux app reside. Manually adding sub-folders is necessary, since this is a blank canvas when you start. You can copy and paste the Linux files folder to and from Chrome OS

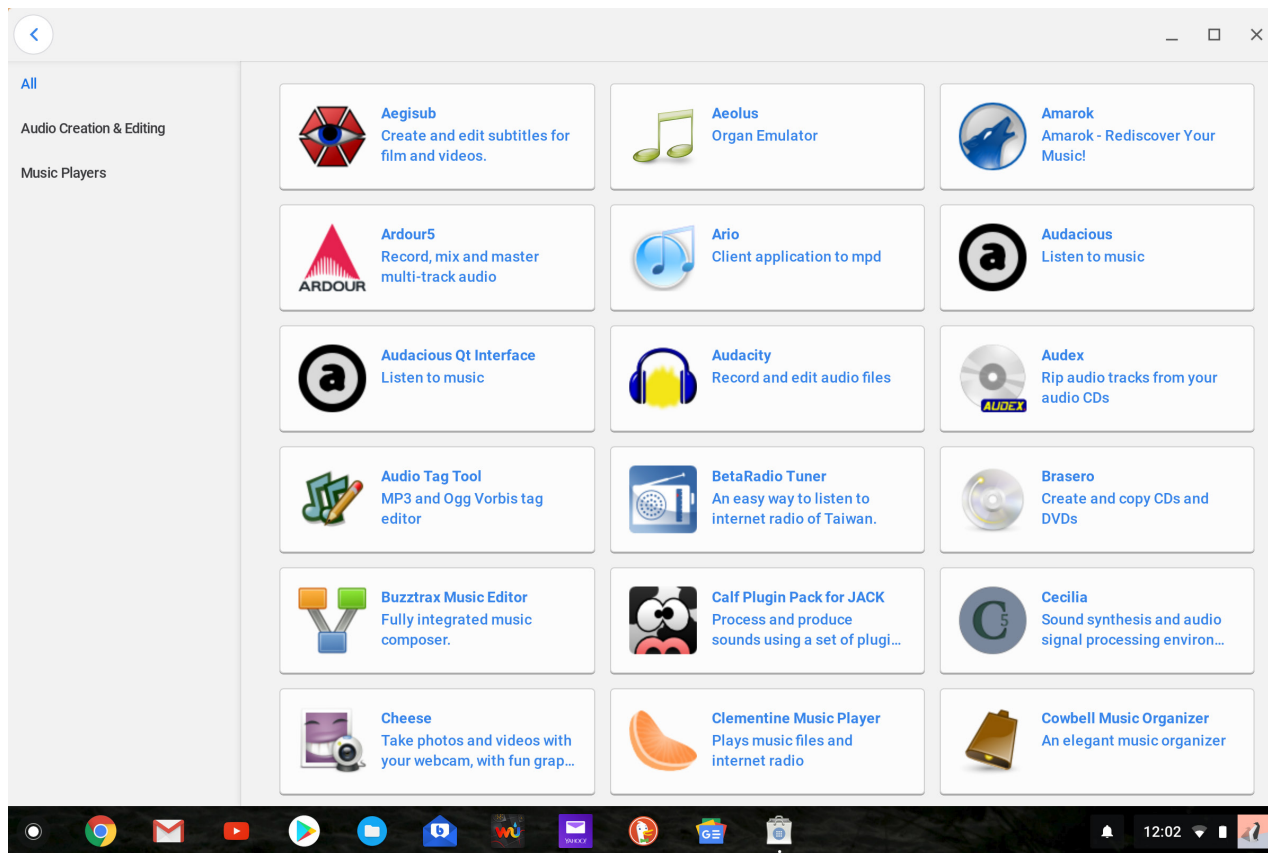


Figure 5. GNOME Software Center

folders, but the reverse is more difficult. Moving files from the Linux files folder back to Chrome OS folders involves copying to either your Google Drive or an external drive, then moving it back to the Chromebook's Chrome OS files folders. This is one function that shows Crostini is still in a beta state—hopefully it'll be corrected in future OS releases.

The rest of my experience using Linux apps on my Chromebook has been great, with the exception of Audacity; they all have functioned exactly as they do on my Ubuntu Linux laptop. The Linux apps further expand the Chromebook's functionality, which already had gotten a substantial boost last year with the addition of Android apps.

The Chromebook is rather quickly becoming a full-service laptop/tablet experience—

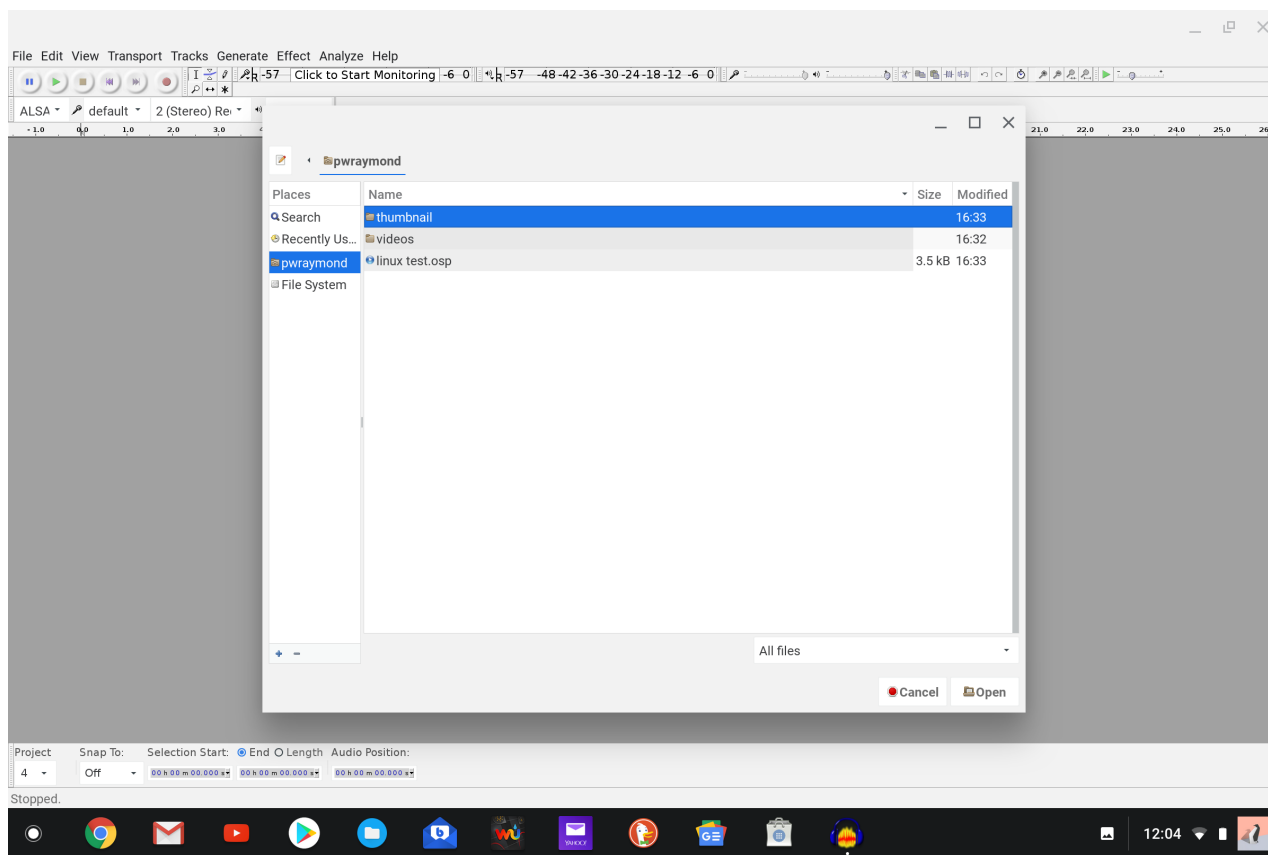


Figure 6. Linux File Folder

one that actually involves three OS experiences under one hood, each one as easy to access as the other. For me, the one Linux app to add a function I needed but didn't previously have is Audacity. Unfortunately, there is currently no audio support in Crostini that allows Audacity to do what it is designed to do—another beta hiccup that hopefully will be addressed sooner rather than later. Just to clarify, you will hear audio from a video or audio file stored in Linux files, such as something transcoded in the Handbrake Linux app, since that is actually being played by a media player in Chrome OS, you just can't currently create audio in Crostini. The GIMP also adds a missing dimension for Chromebooks, providing a full photo-editing suite of tools—who needs Photoshop when you have the GIMP? I believe the addition of Linux apps will enhance the Chromebook's appeal to the general public, not just for the developers for whom Crostini was first created. And, if you are a developer, you now

have another option on which to create.

For developers considering a Chromebook for work, the best option is one of several high-end Chromebooks—like any of the Google Pixelbooks, the Asus Flip c302 or the HP X2. If you need to know what Crostini can and can't do for developing before purchasing, see the open-source [Chromium project page](#) on containers and Crostini, which should answer all the questions you may have on this subject.

Chromebooks are now a viable option for those who wish to use open-source apps with an added layer of security that's hard to match. Plus, the added exposure to open-source apps is also a good thing for the Open Source community. Here's to hoping Crostini progresses from beta to stable and becomes easy to use for everyone. ■

Philip Raymond is a Master Control Supervisor at WFLD-Fox Television in Chicago. He has used and enjoyed using Linux for 15 years and enjoys learning about new open-source projects. You can follow Phil on Twitter @tvphil or on Facebook at www.facebook.com/tvphil.

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

About ncurses Colors

Why does ncurses support only eight colors?

By Jim Hall

If you've looked into the color palette available in curses, you may wonder why curses supports only eight colors. The curses.h include file defines these color macros:

```
COLOR_BLACK  
COLOR_RED  
COLOR_GREEN  
COLOR_YELLOW  
COLOR_BLUE  
COLOR_MAGENTA  
COLOR_CYAN  
COLOR_WHITE
```

But why only eight colors, and why these particular colors? At least with the Linux console, if you're running on a PC, the color range's origins are with the PC hardware.

A Brief History of Color

Linux started as a PC operating system, so the first Linux console was a PC running in text mode. And to understand the color palette on the PC console, you need to go all the way back to the old CGA days. In text mode, the PC terminal had a color palette of 16 colors, enumerated 0 (black) to 15 (white). Backgrounds were limited to the first eight colors:

- 0. Black
- 1. Blue
- 2. Green

- 3. Cyan
- 4. Red
- 5. Magenta
- 6. Brown
- 7. White (“Light Gray”)
- 8. Bright Black (“Gray”)
- 9. Bright Blue
- 10. Bright Green
- 11. Bright Cyan
- 12. Bright Red
- 13. Bright Magenta
- 14. Yellow
- 15. Bright White

These colors go back to CGA, IBM’s Color/Graphics Adapter from the earlier PC-compatible computers. This was a step up from the plain monochrome displays; as the name implies, monochrome could display only black or white. CGA could display a limited range of colors.

CGA supports mixing red (R), green (G) and blue (B) colors. In its simplest form, RGB is either “on” or “off”. In this case, you can mix the RGB colors in $2 \times 2 \times 2 = 8$ ways. Table 1 shows the binary and decimal representations of RGB.

Table 1. Binary and Decimal Representations of RGB

000	(0)	Black
001	(1)	Blue
010	(2)	Green
011	(3)	Cyan
100	(4)	Red
101	(5)	Magenta
110	(6)	Yellow
111	(7)	White

To double the number of colors, CGA added an extra bit called the “intensifier” bit. With the intensifier bit set, the red, green and blue colors would be set to their maximum values. Without the intensifier bit, each RGB value would be set to a “midrange” intensity. Let’s represent that intensifier bit as an extra 1 or 0 in the binary color representation, as iRGB (Table 2).

Table 2. Using the Intensifier Bit

0000	(0)	Black
0001	(1)	Blue
0010	(2)	Green
0011	(3)	Cyan
0100	(4)	Red
0101	(5)	Magenta
0110	(6)	Yellow
0111	(7)	White
1000	(8)	Bright Black
1001	(9)	Bright Blue
1010	(10)	Bright Green
1011	(11)	Bright Cyan
1100	(12)	Bright Red
1101	(13)	Bright Magenta
1110	(14)	Bright Yellow
1111	(15)	Bright White

But there’s a problem: 0000 Black and 1000 Black are the same color. There’s no red, green or blue color to intensify, so black is black whether or not the “intensifier” bit is set. To get around this limitation, CGA actually implemented a modified iRGB definition, using two intermediate values, at about one-third and two-thirds intensity. Most “normal” mode (0 to 7) colors used values at the two-thirds intensity, with the exception of yellow, which was assigned a one-third green value that turned the color brown or orange. To translate from “normal” mode to “bright” mode, convert zero values to the one-third intensity and two-thirds values to full intensity.

Table 3 shows another iteration of the color table, using 0x0 to 0xF for the color range on each RGB value, with 0x5 and 0xA as the one-third and two-thirds intensities, respectively.

Table 3. Color Table Using 0x0 to 0xF for the Color Range on Each RGB Value with 0x5 and 0xA as the One-Third and Two-Thirds Intensities, Respectively

0000	(#000)	Black
0001	(#00A)	Blue
0010	(#0A0)	Green
0011	(#0AA)	Cyan
0100	(#A00)	Red
0101	(#A0A)	Magenta
0110	(#A50)	Brown
0111	(#AAA)	White
1000	(#555)	Bright Black
1001	(#55F)	Bright Blue
1010	(#5F5)	Bright Green
1011	(#5FF)	Bright Cyan
1100	(#F55)	Bright Red
1101	(#F5F)	Bright Magenta
1110	(#FF5)	Bright Yellow
1111	(#FFF)	Bright White

You may wonder why there are only eight background colors. Note that DOS also supported a “Blink” attribute. With this attribute set, your text could blink on and off. The “Blink” bit was encoded at the end of the foreground and background bit-pattern:

Bbbbffff

That’s a full byte! Counting from right to left: four bits to represent the text foreground color (0000 Black to 1111 Bright White), three bits to code the background color (000 Black to 111 White) and one bit for the “Blink” attribute.

And, that's how curses got 16 text colors: eight standard-intensity text colors and eight high-intensity text colors. On the Linux console, these are essentially the same colors used in old DOS systems. That's also why you'll often see "brown" labeled "yellow" in some old DOS programmer references, because at least on DOS systems, it started out as plain "yellow" before the intensifier bit. Similarly, you also may see "gray" represented as "Bright Black", because "gray" is really "black" with the intensifier bit set.

Sample Program

Let me demonstrate the Linux terminal colors with a simple program. This color demo will iterate through all available color combinations using curses.

First, I need a simple function to create all possible color pairs:

```
void init_colorpairs(void)
{
    int fg, bg;
    int colorpair;

    for (bg = 0; bg <= 7; bg++) {
        for (fg = 0; fg <= 7; fg++) {
            colorpair = colornum(fg, bg);
            init_pair(colorpair, curs_color(fg), curs_color(bg));
        }
    }
}
```

The `init_colorpairs()` function also relies on a "translation" function that converts standard-intensity CGA color numbers (0 to 7) to curses color numbers, using the curses constant names like `COLOR_BLUE` or `COLOR_RED`:

```
short curs_color(int fg)
{
```

```
switch (7 & fg) {          /* RGB */
case 0:                   /* 000 */
    return (COLOR_BLACK);
case 1:                   /* 001 */
    return (COLOR_BLUE);
case 2:                   /* 010 */
    return (COLOR_GREEN);
case 3:                   /* 011 */
    return (COLOR_CYAN);
case 4:                   /* 100 */
    return (COLOR_RED);
case 5:                   /* 101 */
    return (COLOR_MAGENTA);
case 6:                   /* 110 */
    return (COLOR_YELLOW);
case 7:                   /* 111 */
    return (COLOR_WHITE);
}
}
```

To create a predictable color pair number for each foreground and background color, I also need a function `colornum()` to set an integer bit pattern based on the classic color byte:

```
int colornum(int fg, int bg)
{
    int B, bbb, ffff;

    B = 1 << 7;
    bbb = (7 & bg) << 4;
    ffff = 7 & fg;

    return (B | bbb | ffff);
}
```

The B bit that usually indicates blinking text is not used in my color demo program, so I always set B to one to guarantee that color pair 0 is never assigned. In curses, color pair 0 is reserved for the default foreground and background colors. That should be white text on a black background, but to be safe, I'll always define my own combination for white on black. For a foreground color 7 (white, binary 111) with background color 0 (black, binary 000), the bit pattern looks like this:

10000111

This is a decimal value of 135.

After `init_colorpairs()`, my program can set each color combination using a wrapper to the curses function `COLOR_PAIR()`. My wrapper function also turns bold text on or off, using the `A_BOLD` attribute:

```
void setcolor(int fg, int bg)
{
    /* set the color pair (colornum) and bold/bright (A_BOLD) */

    attron(COLOR_PAIR(colornum(fg, bg)));
    if (is_bold(fg)) {
        attron(A_BOLD);
    }
}

void unsetcolor(int fg, int bg)
{
    /* unset the color pair (colornum) and
       bold/bright (A_BOLD) */

    attroff(COLOR_PAIR(colornum(fg, bg)));
    if (is_bold(fg)) {
        attroff(A_BOLD);
    }
}
```

```
    }  
}
```

And the `is_bold()` function simply tests if the “intensifier” bit on the iRGB value (foreground colors 8 to 15) is set, using a simple bit mask:

```
int is_bold(int fg)  
{  
    /* return the intensity bit */  
  
    int i;  
  
    i = 1 << 3;  
    return (i & fg);  
}
```

With that, creating the color demonstration program is easy:

```
/* color-demo.c */  
  
#include <curses.h>  
#include <stdio.h>  
#include <stdlib.h>  
  
int is_bold(int fg);  
void init_colorpairs(void);  
short curs_color(int fg);  
int colnum(int fg, int bg);  
void setcolor(int fg, int bg);  
void unsetcolor(int fg, int bg);  
  
int main(void)  
{
```

```
int fg, bg;

/* initialize curses */

initscr();
keypad(stdscr, TRUE);
cbreak();
noecho();

/* initialize colors */

if (has_colors() == FALSE) {
    endwin();
    puts("Your terminal does not support color");
    exit(1);
}

start_color();
init_colorpairs();

/* draw test pattern */

if ((LINES < 24) || (COLS < 80)) {
    endwin();
    puts("Your terminal needs to be at least 80x24");
    exit(2);
}

mvaddstr(0, 35, "COLOR DEMO");
mvaddstr(2, 0, "low intensity text colors (0-7)");
mvaddstr(12, 0, "high intensity text colors (8-15)");

for (bg = 0; bg <= 7; bg++) {
```

```
    for (fg = 0; fg <= 7; fg++) {
        setcolor(fg, bg);
        mvaddstr(fg + 3, bg * 10, "...test...");
        unsetcolor(fg, bg);
    }

    for (fg = 8; fg <= 15; fg++) {
        setcolor(fg, bg);
        mvaddstr(fg + 5, bg * 10, "...test...");
        unsetcolor(fg, bg);
    }
}

mvaddstr(LINES - 1, 0, "press any key to quit");

refresh();

getch();
endwin();

exit(0);
}
```

Sample Output

When you run the program, you see all combinations of 16 text colors and eight background colors, for a total of $16 \times 8 = 128$ different color pairs.

Figure 1 shows how I've set up my graphics terminal to reflect the text-mode terminal, including the standard text colors. Graphical terminal programs (like GNOME Terminal) support a wide range of colors, because they can leverage the available color palette of the X Window System. Note that you can change the available colors in these programs. Most colors are pretty close to their console counterparts, but some colors look quite different. For example, the default color palette for GNOME Terminal replaces the DOS



Figure 1. Color Demo Console

brown with a yellow color (Figure 2).

Through colors, you can represent information more clearly. This color demonstration simply iterates through all color combinations to show how each color looks with every other color.

Of course, this example is just color. You can do so much more with curses, depending on what you need your program to do. In a follow-up article, I'll demonstrate other features of the ncurses library, such as how to create windows and frames. ■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project, and he also focuses on the usability of open-source software. Jim is the Chief Information Officer at Ramsey County, Minnesota.



Figure 2. Color Demo GNOME Terminal

Resources

- Pradeep Padala’s NCURSES Programming HOWTO at the Linux Documentation Project
- “Getting Started with ncurses” by Jim Hall, *LJ*, March 2018
- “Creating an Adventure Game in the Terminal with ncurses” by Jim Hall, *LJ*, April 2018
- “Programming in Color with ncurses” by Jim Hall, *LJ*, May 2018

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

Game Review: *Lamplight City*

A well lit look into Grundislav Games' latest release.

By Patrick Whelan



The universe of *Lamplight City* is rich, complex and oddly familiar. The game draws on that ever-popular theme of a steampunk alternative universe, adding dashes of Victorian squalor and just a pinch of 1950's detective tropes. Is it just a mishmash of clichés then? Yes, but it all works well together to form a likable and somewhat unique universe—like a cheesy movie, you can't help but fall in love with *Lamplight City*.



Figure 1. The *Lamplight City* Universe



Figure 2. Some Protesters

In *Lamplight City*, you play Miles Fordham, a disgraced detective turned PI following the death of his partner in Act I at the hands of a mysterious killer. Miles is

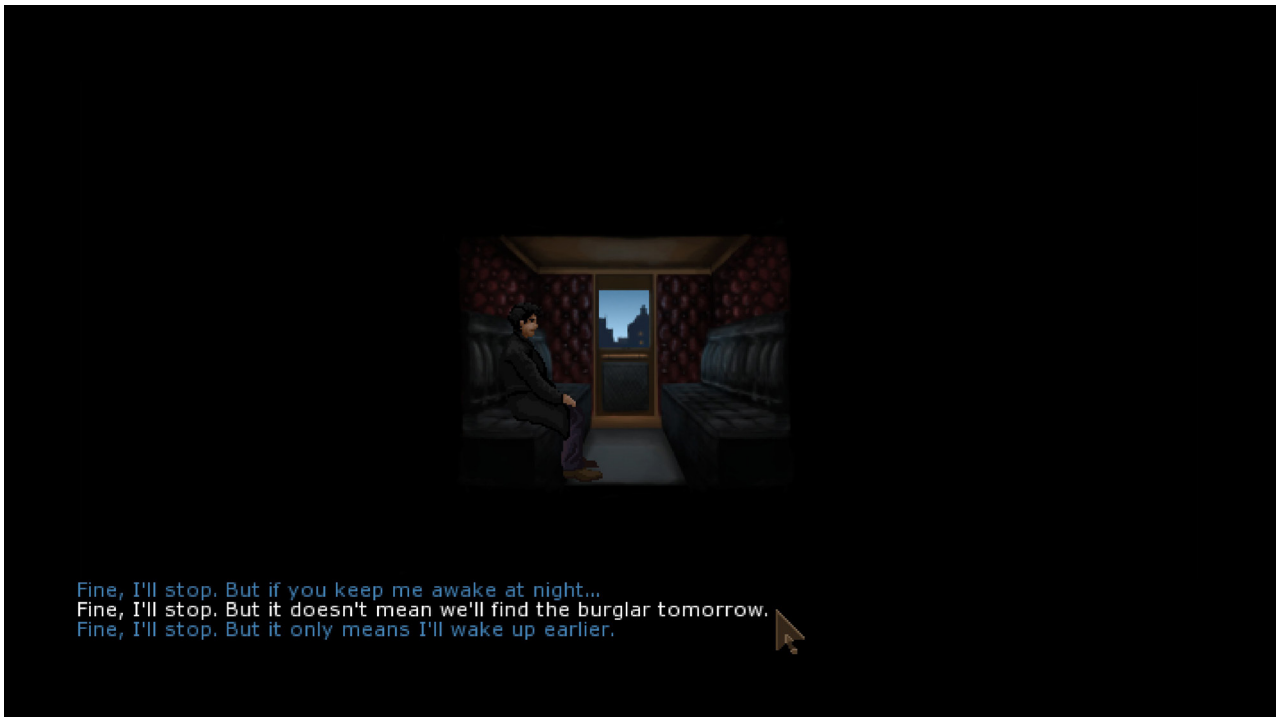


Figure 3. Miles Fordham's Schizophrenic Dialogue

accompanied by the ghostly voice of his partner Bill as a sort of schizophrenic inner monologue. It's creepy, and it's a perfect example of taking a classic trope and turning it into one of the game's biggest strengths. Bill's monologues add witty flavour to the dry protagonist and a way to explain details and scenarios to the player.

Lamplight City features multiple cases that are all tied together with an overarching story. More impressively though is the overarching story's effect on the individual cases. In my play-through, mistakes I made in one case affected another and effectively led to another case becoming unsolvable. This is a system I instinctively hated. It seemed unjustly punitive to punish players for simply exploring dialogue options. Over time, however, as the music and art slowly enveloped me into a universe I truly enjoyed exploring and experiencing, I began to see how subtleties are at the center of this universe. What at first is dismissed as unimportant or underwhelming later appears as a subtle smack in the face, with that familiar feeling of "Oh, I knew I shouldn't have done that!"

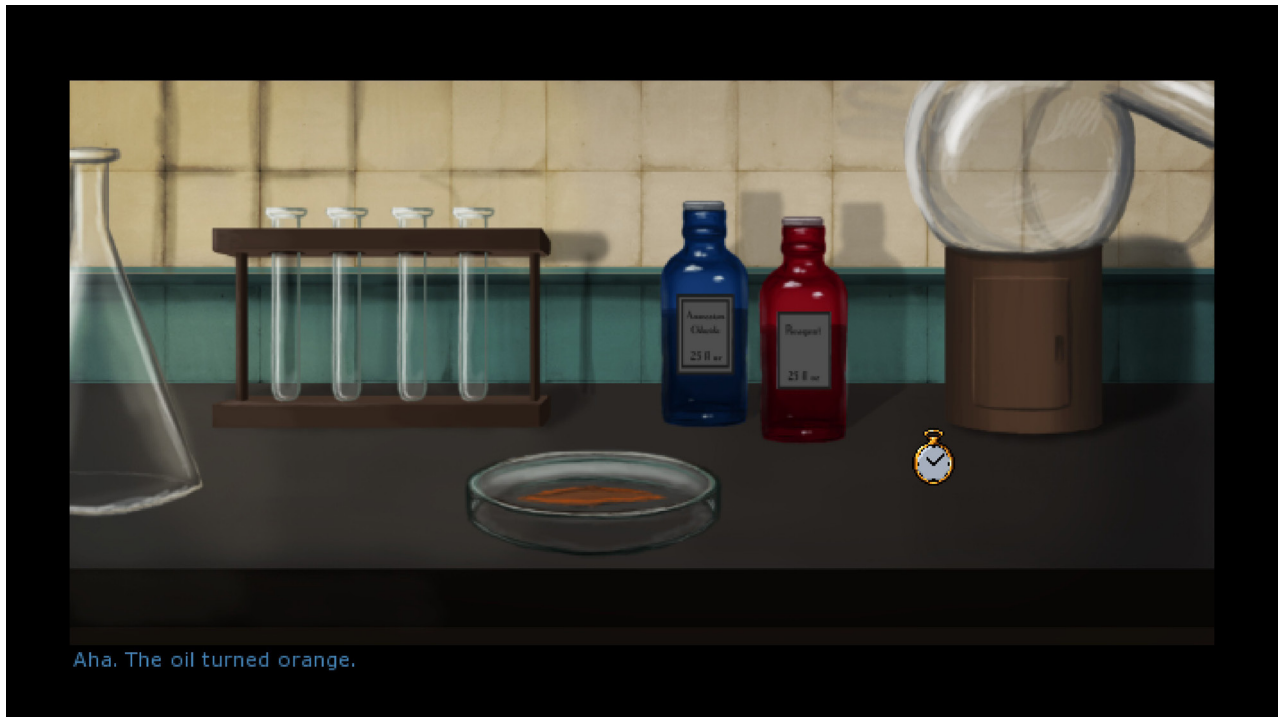


Figure 4. Petri Dish

Patience is most certainly a virtue in *Lamplight City*, which is not to say the game lacks flavor. In fact, the opposite is true. Many times it's down to you to sweep through the data and discern fact from fiction. It's important to note that *Lamplight City* isn't a logic game—not really. There's no inventory, and although there are elements of small physical puzzles, the game thrives on interpersonal relationships and dissecting dialogues, not using some half-forgotten wrench on a valve. It takes patience and discretion. This is not a simple mobile game where enough spamming clicks will win the game. Speaking of winning, let's talk about the end of the game—no spoilers, I promise!

On my first play-through of the final case, I failed, miserably, and I loved it. Most games are too afraid to let the player fail. They'll respawn you, give you tips or let you skip to the next part. When I failed in *Lamplight City*, you can imagine my disbelief and slow, emerging grin as the credits began to roll, the biggest case of the game still unsolved. It's a nice breath of fresh air after the safe world of invincible super heroes and perfect protagonists found in many popular movies and games today.

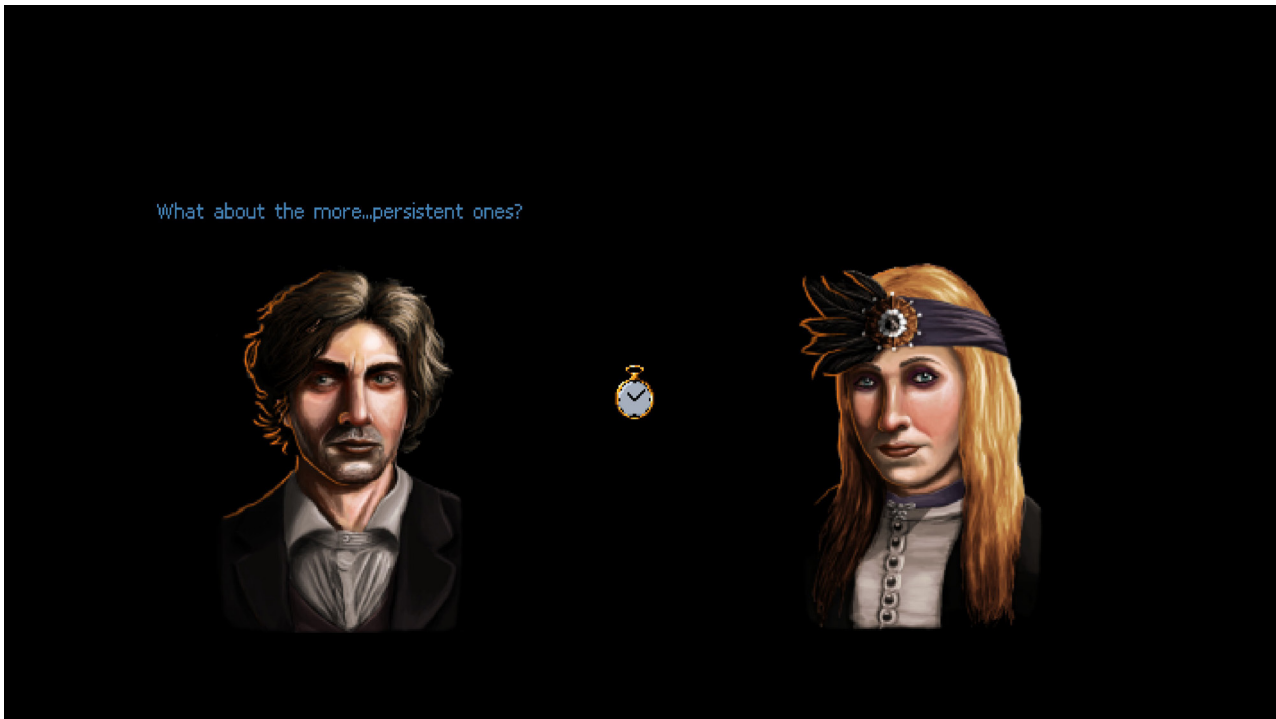


Figure 5. Example Dialogue

All this is not to say, however, that you are doomed to failure. *Lamplight City* provides you with the opportunity to establish yourself in Miles Fordham. The universe of New Bretagne no longer feels distant but real. You are the master of your own fate, and you, the player, are brought into the fold and become invested in every lead and discovery. The game does a great job at portraying a sense of responsibility onto the player for making poor decisions, like annoying your wife, which in any other game wouldn't affect a mighty protagonist nor penalize a player for doing so.

So that's what *Lamplight City* does well, but there are plenty of areas where it's quite not so exceptional. At the end of the day, it feels like a small progression on a long-standing genre of point-and-click adventure games—a small evolution, if you will, instead of a revolution in game design. That's not necessarily a bad thing if you're looking for a safe game you know you'll probably enjoy, but if you're looking to be tested or truly excited by an innovative game, *Lamplight City* isn't for you. With that said, the story is pleasantly compelling, and even if the gameplay itself isn't riveting, the story is comparable to a long book, slowly reeling you into its universe without



Figure 6. Map of New Bretagne

you noticing until 2am when you can't put the game down until you've gotten to the bottom of the case.

Just like other games of its genre, there are moments when you'll be stuck and cursing to high heaven that you wish you'd payed more attention to dialogue or details of previous cases. The game does a good job of keeping you informed through a case book that contains the most relevant information and clues, but this won't stop you from gallivanting across the city multiple times a case talking with every NPC and revisiting every room, hoping for some slight lead. I'll be honest; I watched my fair share of play-throughs to find the next clue. This isn't a fault of the game per se but a problem with the genre in general, often leaving players dazed and confused with little direction. It's a feature of the genre, and it can be extremely alluring or repulsive depending on the type of gamer you are. This game sticks with its hard-core roots of butterfly-effect dialogues, refusing to implement a hints system as in other similar games, sometimes making it frustrating.

Finally, *Lamplight City* is a relatively short game at around ten hours for a price tag of \$15 USD. This falls far short of large "triple A games" with thousands of developers

creating vast expanses of land and hundreds of characters with which to interact. However, with multiple endings, more than 50 voiced characters and a uniquely charming art style, *Lamplight City* has a great replay-ability value, and there are still cases I haven't totally solved. The price, although not entirely competitive with huge games, is fair enough for an intriguing and engaging story that will undoubtedly have you playing again! ■

Patrick Whelan is a second-year student at Edge Hill University in the UK. He is an aspiring developer, blogger and all-round hacker.

Additional Info

Lamplight City was released September 13, 2018. The game was developed by Grundislav Games and published by Application Systems Heidelberg. It's available for SteamOS + Linux, Mac OS X and Windows.

You can purchase *Lamplight City* from the [Steam Store](#).

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

Time for Net Giants to Pay Fairly for the Open Source on Which They Depend

Net giants depend on open source: so where's the gratitude?

By Glyn Moody

Licensing lies at the heart of open source. Arguably, free software began with [the publication of the GNU GPL in 1989](#). And since then, open-source projects are defined as such by virtue of [the licenses they adopt](#) and whether the latter meet the [Open Source Definition](#). The continuing importance of licensing is shown by the periodic flame wars that erupt in this area. Recently, there have been two such flarings of strong feelings, both of which raise important issues.

First, we had the incident with [Lerna](#), “a tool for managing



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+](#).

OPEN SAUCE

JavaScript projects with multiple packages”. It came about as a result of the way the US Immigration and Customs Enforcement (ICE) has been **separating families** and **holding children in cage-like cells**. The Lerna core team was appalled by this behavior and wished to do something concrete in response. As a result, it **added an extra clause to the MIT license**, which forbade a list of companies, including Microsoft, Palantir, Amazon, Motorola and Dell, from being permitted to use the code:

For the companies that are known supporters of ICE: Lerna will no longer be licensed as MIT for you. You will receive no licensing rights and any use of Lerna will be considered theft. You will not be able to pay for a license, the only way that it is going to change is by you publicly tearing up your contracts with ICE.

Many sympathized with the feelings about the actions of the ICE and the intent of the license change. However, many also pointed out that such a move went against the core principles of both free software and open source. **Freedom 0 of the Free Software Definition** is “The freedom to run the program as you wish, for any purpose.” Similarly, the Open Source Definition requires “No Discrimination Against Persons or Groups” and “No Discrimination Against Fields of Endeavor”. The situation is clear cut, and it didn’t take long for the Lerna team to realize their error, and **they soon reverted the change**:

I apologize for making the rash decision to support the addition of an unenforceable clause to the project’s MIT license. I failed to accurately assess the impact of this change, which led me to (incorrectly) focus on the intent. Despite the most noble of intentions, it is clear to me now that the impact of this change was almost 100% negative, with no appreciable progress toward the ostensible goal aside from rancorous sniping and harmful drama.

The Lerna episode was a useful opportunity for the Open Source world to remind itself that true freedom includes the freedom to use software in **ways that many might not approve of**. Stallman appreciated this early on, and he wrote the post **“Why programs must not limit the freedom to run them”** on the topic.

OPEN SAUCE

The other flare-up over licensing has been similarly instructive. It involves [Redis](#), “an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker”. Here’s the problem, as [outlined by Salil Deshpande](#), Managing Director at Bain Capital Ventures, which has invested in [Redis Labs](#):

Amazon takes Redis (the most loved database in StackOverflow’s developer survey), gives very little back, and runs it as a service, re-branded as AWS ElastiCache. Many other popular open-source projects including, Elasticsearch, Kafka, Postgres, MySQL, Docker, Hadoop, Spark and more, have similarly been taken and offered as AWS products.

To be clear, this is not illegal. But we think it is wrong, and not conducive to sustainable open-source communities.

The response from Redis Labs was to append for certain add-on modules an extra paragraph, known as the [Commons Clause](#), to its open-source license. It includes the following: “Without limiting other conditions in the License, the grant of rights under the License will not include, and the License does not grant to you, the right to Sell the Software.”

Deshpande writes: “if you want to take substantially the same software that someone else has built, and offer it as a service, for your own profit, that’s not in the spirit of the open-source community.” But that’s incorrect. Both the Free Software Definition and the Open Source Definition explicitly require that option. There’s even a page all about [selling free software](#) on the main gnu.org site. Well known voices within the coding community were [more or less unanimous](#): the Commons Clause negates any open-source license it is used in conjunction with, and really misses the point of free software.

That notwithstanding, the Commons Clause does spring from a legitimate concern. As Deshpande points out, Amazon is making a lot of money offering open-source programs on its cloud computing platform. In fact, its debt to the hacker community goes much deeper: free software permeates the company and its operations. Without open source’s low costs and ability to scale, Amazon might never have grown to become [the world’s second trillion-dollar company](#) by valuation.

OPEN SAUCE

Moreover, the same can be said of many of today's internet giants. Google and Facebook are also built on a variety of open-source programs, and they probably would have struggled to achieve the rapid growth and today's high profitability had they been forced to depend on proprietary code.

It's true that all these companies "give back" to free software in various ways. They have open-sourced some code that they have written; provided bug reports and fixes to key programs; supported top free software coders by employing them; and encouraged young people to join the Open Source world, for example through [Google's annual Summer of Code](#).

Those are all welcome. But they are not enough. The decision to craft the Commons Clause was driven largely by companies based around open source seeing internet giants like Amazon deriving great financial benefit from being a [free rider](#) on their efforts. That is neither fair nor sustainable. Indeed, it is extremely foolish for companies like Google and Facebook to exploit open source and its culture of frictionless giving. Ultimately, if these companies that are highly dependent on free software don't start providing serious financial support, paid directly to open-source projects and to their associated companies, those resources will dwindle and may disappear.

They should pay not because the license forces them, but simply because it is in their own interests to do so. That's particularly true at a time when the big internet companies are increasingly being painted as the source of all digital evil. Frankly, they need to do more on the public relations front if they are to avoid punitive legislation being passed around the world. Supporting open source generously—with some very large and regular cash payments—would be an excellent way to do that. It would burnish their public image, safeguard their core infrastructure and give a massive boost to projects whose unstinting generosity enriches the entire world. ■

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