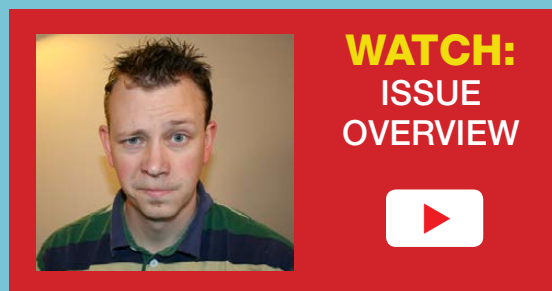


LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community



NOVEMBER 2016 | ISSUE 271
<http://www.linuxjournal.com>

Low Power Wireless for the IoT

Raspberry Pi and 6LoWPAN

GCC Inline Assembly and Its Use in the Linux Kernel



GENERAL PRACTICES FOR SIMPLE SERVER HARDENING

CODE TRIAGE HOW-TO

GRAPH DATA WITH CACTI

**Practical books
for the most technical
people on the planet.**

GEEK GUIDES



**Download books for free with a
simple one-time registration.**

<http://geekguide.linuxjournal.com>

NEW!



BotFactory: Automating the End of Cloud Sprawl

Author: John S. Tonello
Sponsor: BotFactory.io

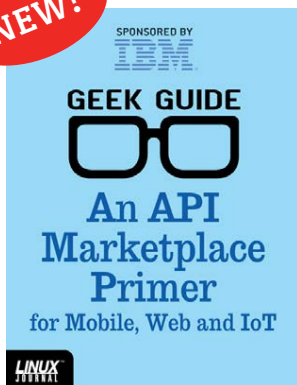
NEW!



Containers 101

Author: Sol Lederman
Sponsor: Puppet

NEW!



An API Marketplace Primer for Mobile, Web and IoT

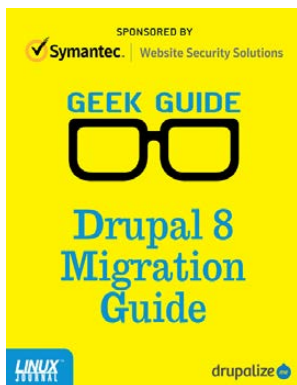
Author: Ted Schmidt
Sponsor: IBM

NEW!



Public Cloud Scalability for Enterprise Applications

Author: Petros Koutoupis
Sponsor: SUSE



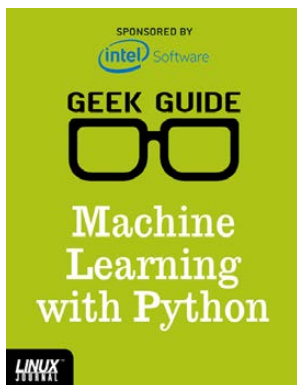
Drupal 8 Migration Guide

Author: Drupalize.me
Sponsor: Symantec



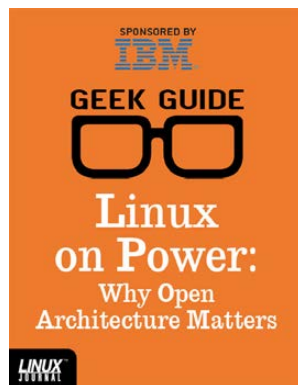
Beyond Cron, Part II: Deploying a Modern Scheduling Alternative

Author: Mike Diehl
Sponsor: Skybot



Machine Learning with Python

Author: Reuven M. Lerner
Sponsor: Intel



Linux on Power: Why Open Architecture Matters

Author: Ted Schmidt
Sponsor: IBM

FEATURES

92 **Low Power Wireless: 6LoWPAN, IEEE802.15.4 and the Raspberry Pi**

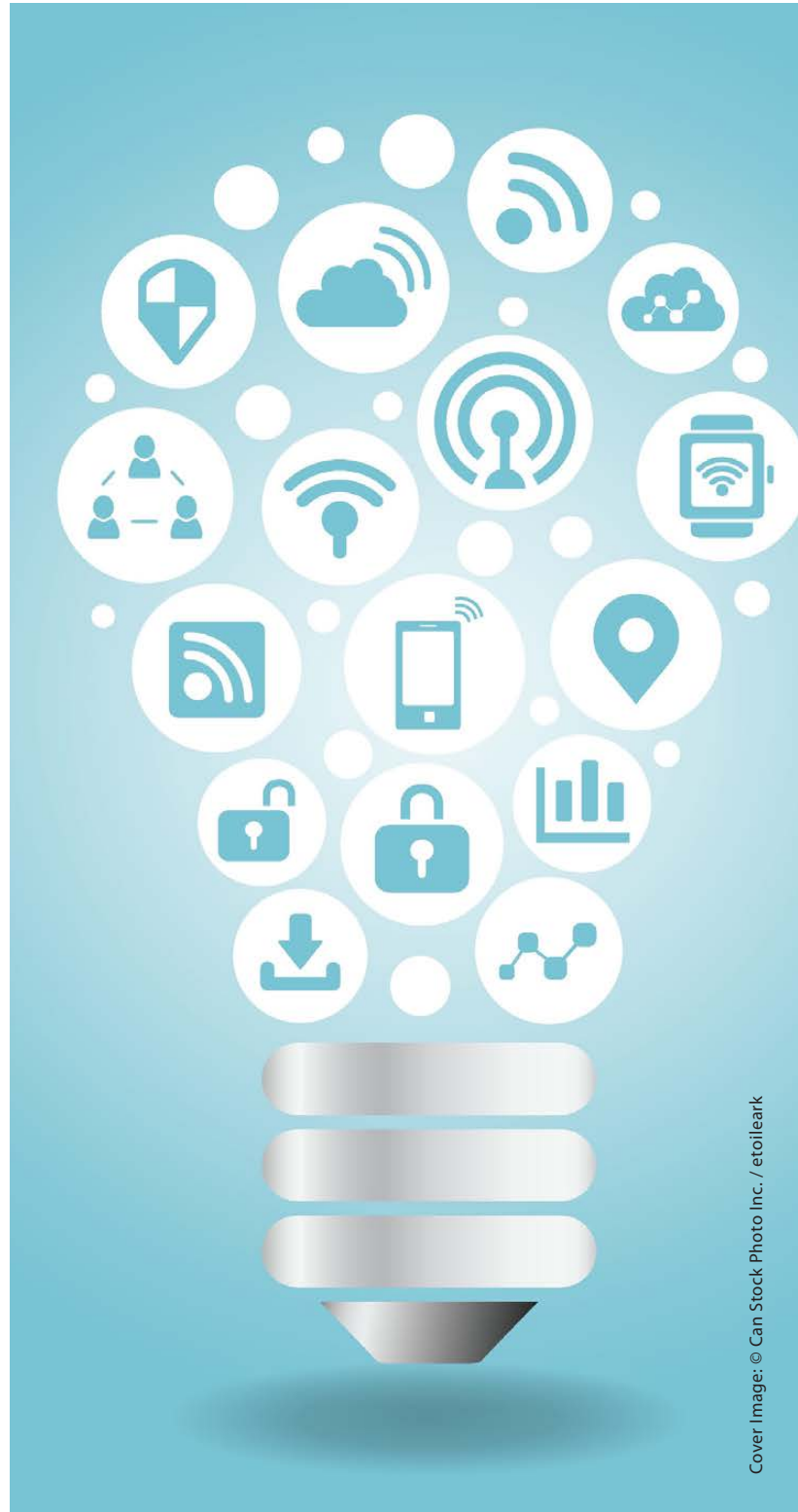
IoT applications will rely on the connections between sensors and actuators and the internet. This will likely be wireless, and it will have to be low power.

Jan Newmarch

108 **GCC Inline Assembly and Its Usage in the Linux Kernel**

Dibyendu explores why it's important to know about it.

Dibyendu Roy



COLUMNS

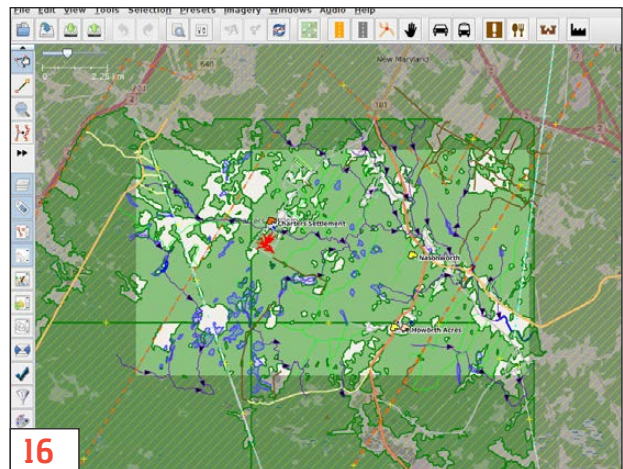
- 26** **Reuven M. Lerner's
At the Forge**
Preparing Data for
Machine Learning
- 36** **Dave Taylor's
Work the Shell**
Wrapping Up the Mars Lander
- 44** **Kyle Rankin's
Hack and /**
Simple Server Hardening, Part II
- 50** **Shawn Powers'
The Open Source
Classroom**
Graph Any Data with Cacti
- 70** **Susan Sons'
Under the Sink**
Holy Triage, Batman!
- 128** **Doc Searls' EOF**
Pancaking the
Pyramid Economy

ON THE COVER

- Low Power Wireless for the IoT: Raspberry Pi and 6LoWPAN, p. 92
- GCC Inline Assembly and Its Use in the Linux Kernel, p. 108
- Techniques and Ideas in the World of Machine Learning, p. 26
- General Practices for Simple Server Hardening, p. 44
- Code Triage How-To, p. 70
- Graph Data with Cacti, p. 50

IN EVERY ISSUE

- 8** **Current_Issue.tar.gz**
- 10** **UPFRONT**
- 24** **Editors' Choice**
- 84** **New Products**
- 131** **Advertisers Index**



LINUX JOURNAL™

Subscribe to
Linux Journal
Digital Edition
for only
\$2.45 an issue.



ENJOY:

- Timely delivery
- Off-line reading
- Easy navigation
- Phrase search and highlighting
- Ability to save, clip and share articles
- Embedded videos
- Android & iOS apps, desktop and e-Reader versions

SUBSCRIBE TODAY!

LINUX JOURNAL

Executive Editor	Jill Franklin jill@linuxjournal.com
Senior Editor	Doc Searls doc@linuxjournal.com
Associate Editor	Shawn Powers shawn@linuxjournal.com
Art Director	Garrick Antikajian garrick@linuxjournal.com
Products Editor	James Gray newproducts@linuxjournal.com
Editor Emeritus	Don Marti dmarti@linuxjournal.com
Technical Editor	Michael Baxter mab@cruzio.com
Senior Columnist	Reuven Lerner reuven@lerner.co.il
Security Editor	Mick Bauer mick@visi.com
Hack Editor	Kyle Rankin lj@greenfly.net
Virtual Editor	Bill Childers bill.childers@linuxjournal.com

Contributing Editors

Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte
Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf • Justin Ryan • Adam Monsen

President Carlie Fairchild
publisher@linuxjournal.com

Publisher Mark Irgang
mark@linuxjournal.com

Associate Publisher John Grogan
john@linuxjournal.com

Director of Digital Experience Katherine Druckman
webmistress@linuxjournal.com

Accountant Candy Beauchamp
acct@linuxjournal.com

**Linux Journal is published by, and is a registered trade name of,
Belltown Media, Inc.**

PO Box 980985, Houston, TX 77098 USA

Editorial Advisory Panel

Nick Baronian
Kalyana Krishna Chadalavada
Brian Conner • Keir Davis
Michael Eager • Victor Gregorio
David A. Lane • Steve Marquez
Dave McAllister • Thomas Quinlan
Chris D. Stark • Patrick Swartz

Advertising

E-MAIL: ads@linuxjournal.com
URL: www.linuxjournal.com/advertising
PHONE: +1 713-344-1956 ext. 2

Subscriptions

E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
MAIL: PO Box 980985, Houston, TX 77098 USA

LINUX is a registered trademark of Linus Torvalds.

SUSEconTM '16

DEFINE YOUR FUTURE

Are you ready for the software-defined future? Learn how you can build a flexible, open infrastructure that enables you to operate more efficiently, innovate faster and rapidly adapt to business needs.

150+ SESSIONS

100+ HOURS HANDS ON TRAINING

5 CERTIFICATION EXAMS

TECHNOLOGY SHOWCASE

NOVEMBER 7-11 ★ WASHINGTON D.C.

SUSECON.COM



Life Hacking

I like the idea of life hacking. I'm not sure it's a term that you'll find in the dictionary (although perhaps—dictionaries have some odd things in them now), but the idea of improving life by programmatically changing things is awesome. I think that might be why I'm such an open-source fan. When it's possible to change the things you don't like or improve on something just because you can, it makes computing far less mystical and far more enjoyable.

This month's issue starts off with Reuven M. Lerner discussing machine learning. Whether you consider it creepy or incredible (or possibly somewhere in between), sites like Amazon.com do an amazing job of determining what sorts of things you might want to buy. All those years of companies data mining for the off chance they'd connect you to something you might want to buy has turned into a very specific automation to pair buyers with items they likely want. And it works so well it's scary. Reuven talks about the science behind the magic.

Kyle Rankin follows with part two of his series on server hardening. My go-to response for "what is the best upgrade for a server?" used to be "RAM". Now, that answer has been superseded by "hardening". Kyle is a system administrator by trade, and he shares his skills with us in this ongoing series.

I decided to address a different problem this



**SHAWN
POWERS**

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via email at shawn@linuxjournal.com. Or, swing by the [#linuxjournal](https://freenode.net) IRC channel on [Freenode.net](https://freenode.net).



VIDEO:
Shawn Powers runs through the latest issue.

month: figuring out how to graph data. It's been on my back burner for years, and this seemed to be the perfect time to suss out the specifics on graphing data. If you've ever been confused and intrigued by MRTG and RRDTools, check out my look at Cacti this month. I learned a lot, and hopefully you will too.

During the past few years, I've learned to write just enough code to be dangerous. That sounds like a silly quip, but really, if you're writing code for a production environment, mistakes can be dangerous. Since I was never formally taught, it means my attempts are likely ridden with security holes and inefficient code. Susan Sons talks about code triage this month. Looking at other peoples' code is often overwhelming and seldom fun. Susan talks about how to do a relatively quick look at existing code to try to identify problems or poor programming choices. It's a great article, but it forced me to realize just how much I shouldn't be writing mission-critical code!

Jan Newmarch teaches us about 6LoWPAN in this issue. With devices around the house connecting to the Internet of Things at an increasing rate, low power connectivity is an important topic to understand. Jan discusses how to utilize the low power standards in existing equipment and how to plan your IoT implementations correctly from the start. Dibyendu Roy follows with a look at how to implement assembly code into GCC on devices that aren't strictly Intel-based. Assembly code is extremely fast, and by using chunks of it in your higher-level programs, you can really benefit on several fronts. Thankfully assembly code exists for multiple platforms and can be integrated inline. Dibyendu shows how.

If you're not happy with the world as it is, it's up to you to change the world. As open-source advocates, that's something we've been doing for years. With the advent of the Internet of Things, we have an opportunity to change the way our physical world works, and Linux is leading the way. This issue is full of insightful information about new technology, improvements on old tech and all the other things you expect from *Linux Journal*. Whether you're here for the new product announcements or want to see what weird new app I've found, we hope this issue has something for everyone. ■

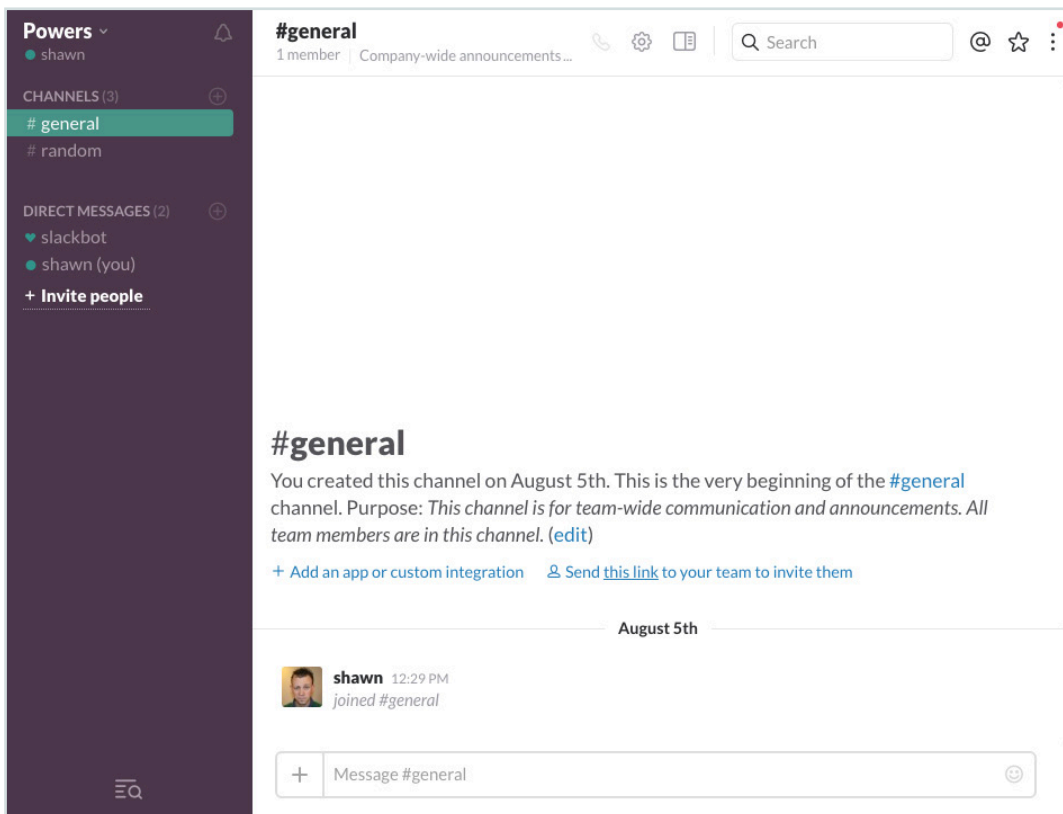
UPFRONT

NEWS + FUN



PREVIOUS
Current_Issue.tar.gz

NEXT
Editors' Choice



Now “Slack”-ing Off is Encouraged!

If your company hasn't already chosen to utilize <https://slack.com>, it's probably only a matter of time. For anyone who has been around IRC before, Slack might seem like a total ripoff. I'll be honest, when

At Your Service

one of the companies I work for starting using it, I wasn't impressed, because I could do all the same things with IRC.

I was wrong—at least partially. Slack is certainly an IRC-like communication tool that is perfect for communicating with other people in an organization. What makes it special is its ability to integrate with other services so well. Things like Google-based authentication for single-sign-in is simple to configure. It also has a very robust API, so you can integrate countless third-party tools into your Slack environment. Some of them are silly (like adding GIPHY so you can insert animated GIF files into your conversations), and some are incredibly useful (like triggering remote scripts to give you real-time feedback in a chat window).

Plus, if you're like me and you can't do without IRC, it's possible to integrate IRC into your Slack experience using Sameroom (<https://sameroom.io>). Truly it's the ability to integrate with other services that makes Slack so powerful, and its ease of use makes it popular for companies, even when the employees aren't all IT folks. Check it out today if you're looking for a great way to communicate with groups of people in your life that might not appreciate the nuances of IRC!—Shawn Powers

SUBSCRIPTIONS: *Linux Journal* is available in a variety of digital formats, including PDF, .epub, .mobi and an online digital edition, as well as apps for iOS and Android devices. 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*, PO Box 980985, Houston, TX 77098 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 various 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*, PO Box 980985, Houston, TX 77098 USA. Letters may be edited for space and clarity.

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>.

FREE e-NEWSLETTERS: *Linux Journal* editors publish newsletters on both a weekly and monthly basis. 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/enewsletters>.

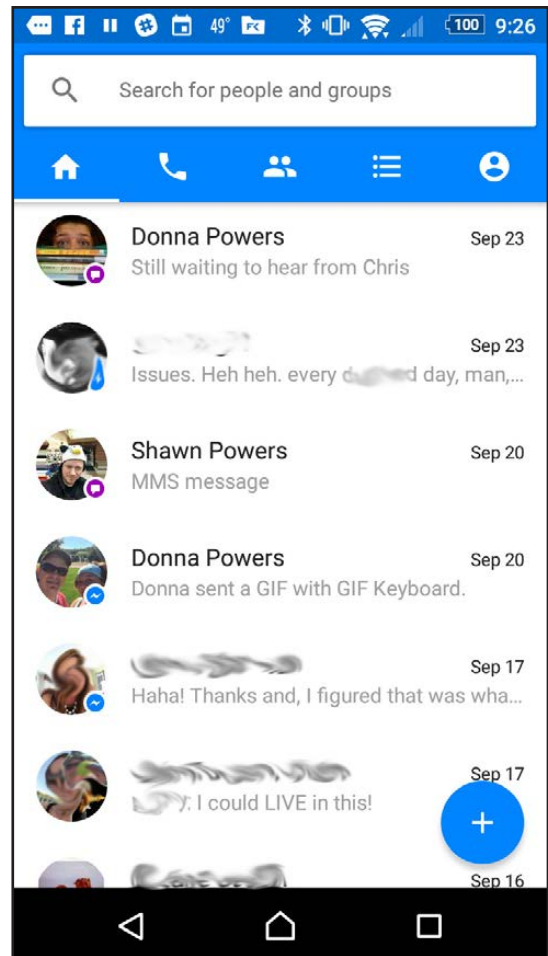
ADVERTISING: *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line: <http://www.linuxjournal.com/advertising>. Contact us directly for further information: ads@linuxjournal.com or +1 713-344-1956 ext. 2.

Android Candy: Facebook Everything?!?!

When Facebook decided its messenger app would be an entirely separate program from its regular app, I was ticked off. I didn't want to have a second application in order to send private messages. It seemed like a needless extra step. And, I stuck by that opinion until I realized I could integrate regular SMS and MMS messages into Facebook Messenger.

Yes, this makes me a bit of a sellout, but I've found it incredibly useful to have my Facebook messages and text (SMS/MMS) messages in one place. There are people I often communicate with via Facebook, and people I usually text. Thanks to Facebook Messenger acting as a client for both, I can communicate from one application. Because using multiple apps for Facebook was my initial beef with Facebook, allowing me to consolidate texting and messaging actually makes me happy.

I know the idea of using Facebook Messenger for texting isn't everyone's cup of tea. I didn't realize it was an option until recently, however, and I am really enjoying how it works! You can see in my (slightly blurred) screenshot, messages are differentiated by their icons. The purple icon means it was an SMS or MMS, and the blue icon shows a Facebook message. Even though it makes me feel a little like a sellout, if you use both texting and Facebook messaging for communicating often, I recommend giving it a try.—Shawn Powers





Where every interaction matters.

break down your innovation barriers

power your business to its full potential

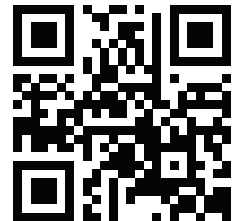
When you're presented with new opportunities, you want to focus on turning them into successes, not whether your IT solution can support them.

Peer 1 Hosting powers your business with our wholly owned FastFiber Network™, global footprint, and offers professionally managed public and private cloud solutions that are secure, scalable, and customized for your business.

Unsurpassed performance and reliability help build your business foundation to be rock-solid, ready for high growth, and deliver the fast user experience your customers expect.

Want more on cloud?

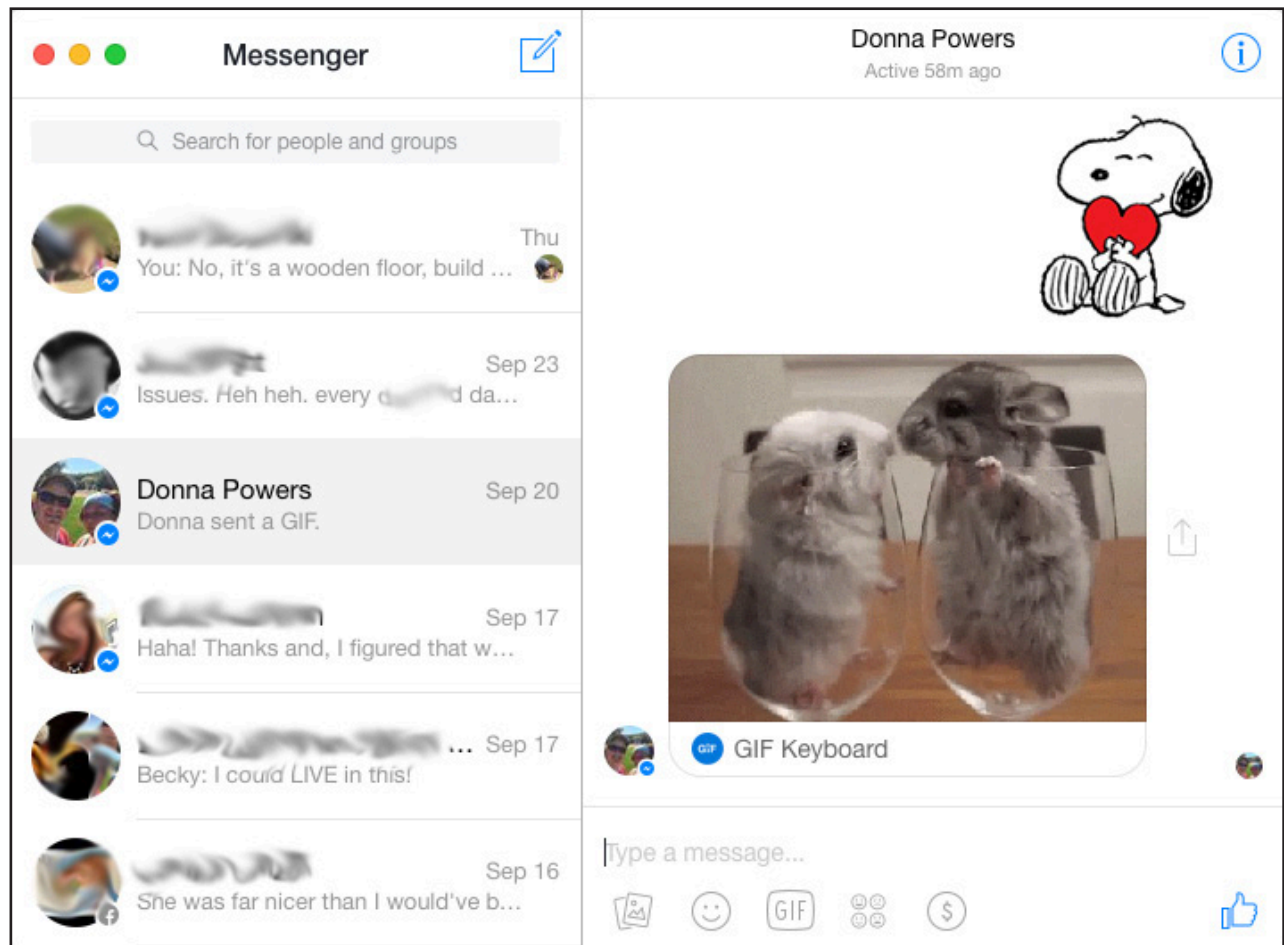
Call: 844.855.6655 | go.peer1.com/linux | [View Cloud Webinar:](#)



Public and Private Cloud | Managed Hosting | Dedicated Hosting | Colocation

Non-Linux FOSS: Facebook on OS X, sans Browser!

I wrote about using Facebook Messenger as an SMS client for my “Android Candy” piece this month. And because I’ll likely get lots of email about how horrible Facebook messenger is, I might as well go all in and share this open-source program: Messenger for Mac.



Over at <https://fbmacmessenger.rsms.me>, you'll find an OS X-native application that is a wrapper around Facebook Messenger. What makes it great is that it doesn't feel like a wrapper at all; it feels like an actual app. If you're using OS X and prefer applications rather than just web browser tabs, be sure to check it out.

I'll be honest; I don't do a ton of communication via Facebook Messenger. It's a great way to send a silly GIF sticker to my wife, however, and for that reason alone, I appreciate the protocol! You can download Messenger for Mac from the website above or from the GitHub page: <https://github.com/rsms/fb-mac-messenger>.

—Shawn Powers

THEY SAID IT

Getting there isn't half the fun—it's all the fun.

—Robert Townsend

The way you overcome shyness is to become so wrapped up in something that you forget to be afraid.

—Lady Bird Johnson

I can think of nothing less pleasurable than a life devoted to pleasure.

—John D. Rockefeller

Imagination is more important than knowledge.

—Albert Einstein

So you see, imagination needs moodling—long, inefficient, happy idling, dawdling and puttering.

—Brenda Ueland

Editing Your Own OpenStreet Maps

JOSM (Java OpenStreetMaps) editor is a tool you can use to create your own maps (<https://josm.openstreetmap.de>). This tool allows you to build your own maps based on data from OpenStreetMaps, other online sources or your own data. You can make edits, add annotations and upload your results back on to the OpenStreetMaps server.

There are two ways you can run JOSM. The first is to install it on your system. If you have it within your package management system, you may want to install that way so any required dependencies are installed automatically. For example, the following command will install it on

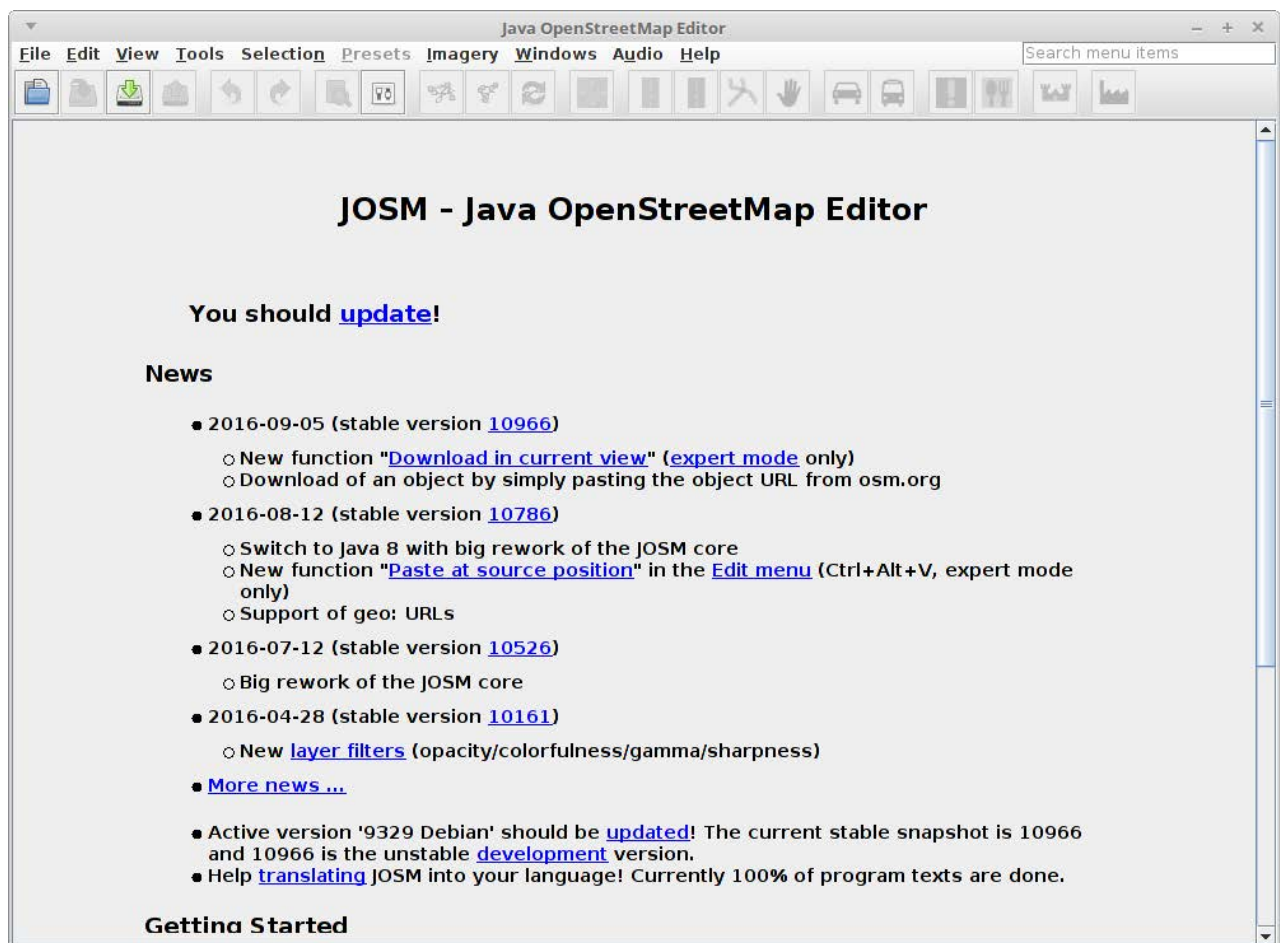


Figure 1. When you first start JOSM, you get an information panel.

Debian-based systems:

```
sudo apt-get install josm josm-plugins
```

Notice that the above command also installs the package of plugins for JOSM. A large number of plugins are available from the community on the main project's website. And, if you want the latest version of JOSM, you can download a jar file from the project's website.

The second way to run JOSM is actually using a Java Web Start package. You simply launch the JNLP file, again from the main project's website.

Once you have started JOSM, in whichever way you have chosen to launch it, you will get a window with an information panel about updates and startup information. There also are links to online help information and the community forums.

Now you can start creating your first map. The easiest way to get started is to download OpenStreetMap data as a starting point. Clicking on the File→Download from OSM menu item will pop up a new window where you can select an area to use as your map base.

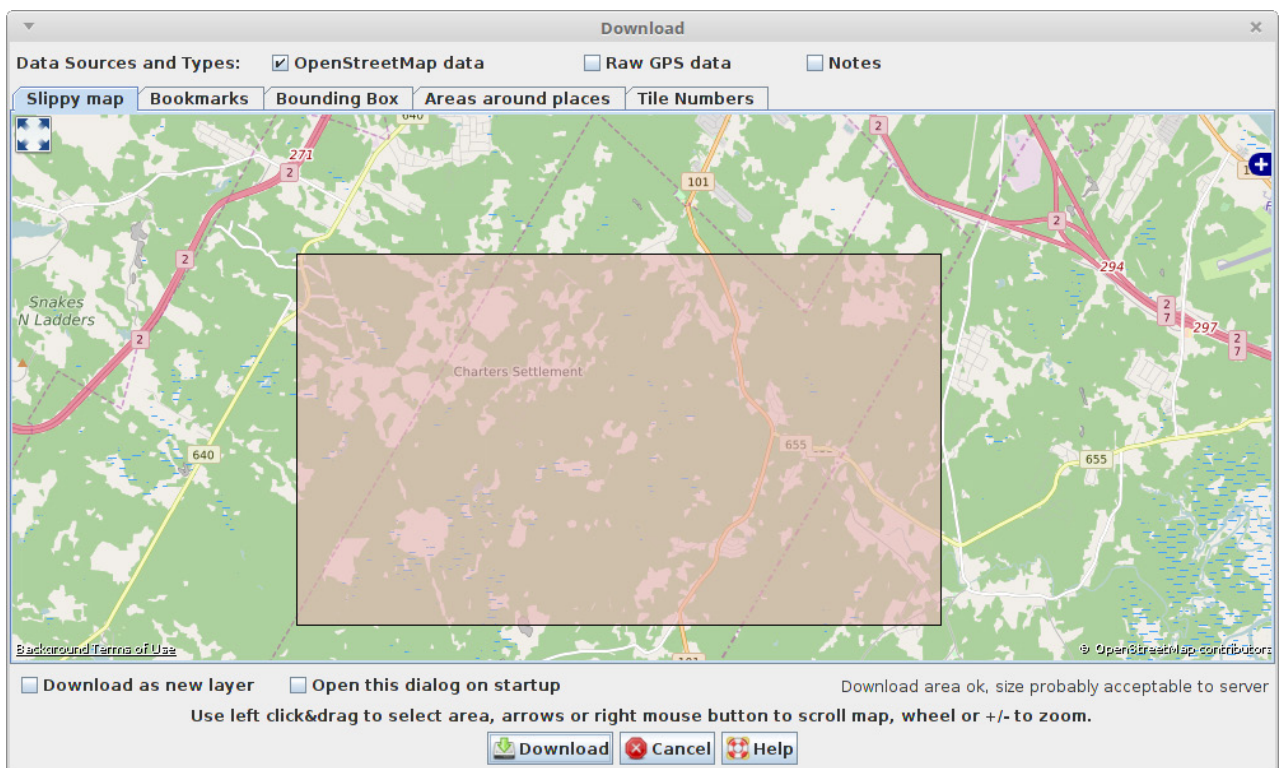


Figure 2. You can select and download an area of the Earth as your map base.

At the top of this window, you can choose what data to download. The choices are OSM data, raw GPS data and additional notes. Within the map window, you can pan around to the area of interest and zoom in to an appropriate level. You then click and drag with your mouse to select a box of the area that you want to use in your map. Be careful with how big a bounding box you select, because the data server limits how much data you can pull at once. Luckily, you will get a warning at the bottom of the window if the bounding box is too large.

Once you have made your selection, you either can click on the download button to download it as a new map, or you can click the option "Download as a new layer" at the bottom of the map pane first to add this data to an already existing map. For now, let's just click the download button to get a brand-new map started.

Looking at the new map, you can see that there is a lot of information available and a lot of functionality you can use to interact with that information. On the left-hand side, there is a strip of icon buttons you can choose from to interact with the map data. The right-hand side has several panes that can show the layers and detailed information of

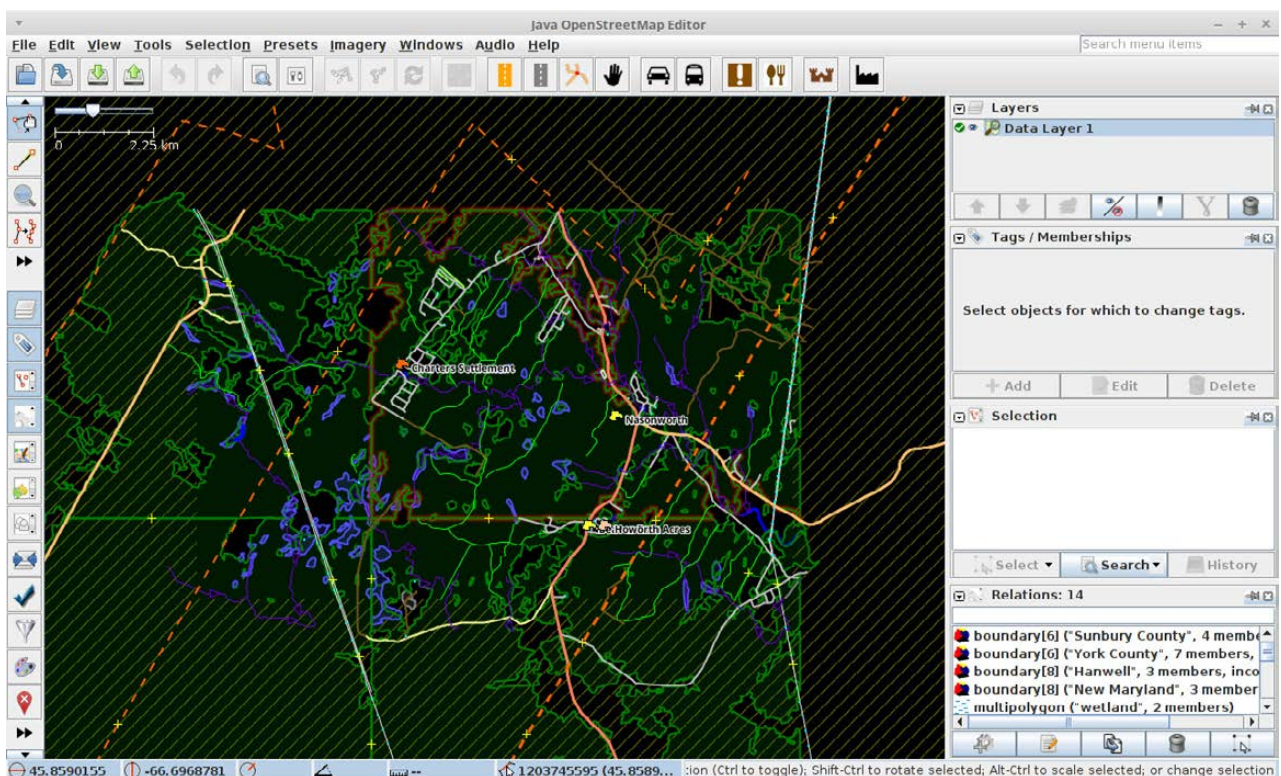


Figure 3. You can download OpenStreetMaps data to start a brand-new map.

selected objects on the map. You can change which panes are visible on the right-hand side by clicking the Windows menu item and selecting the panes that interest you. If you select an object on the map, such as a road section or water way, the details of that selected object will show up in the information panes on the right-hand side.

Raw geographical information is not the only data source that is

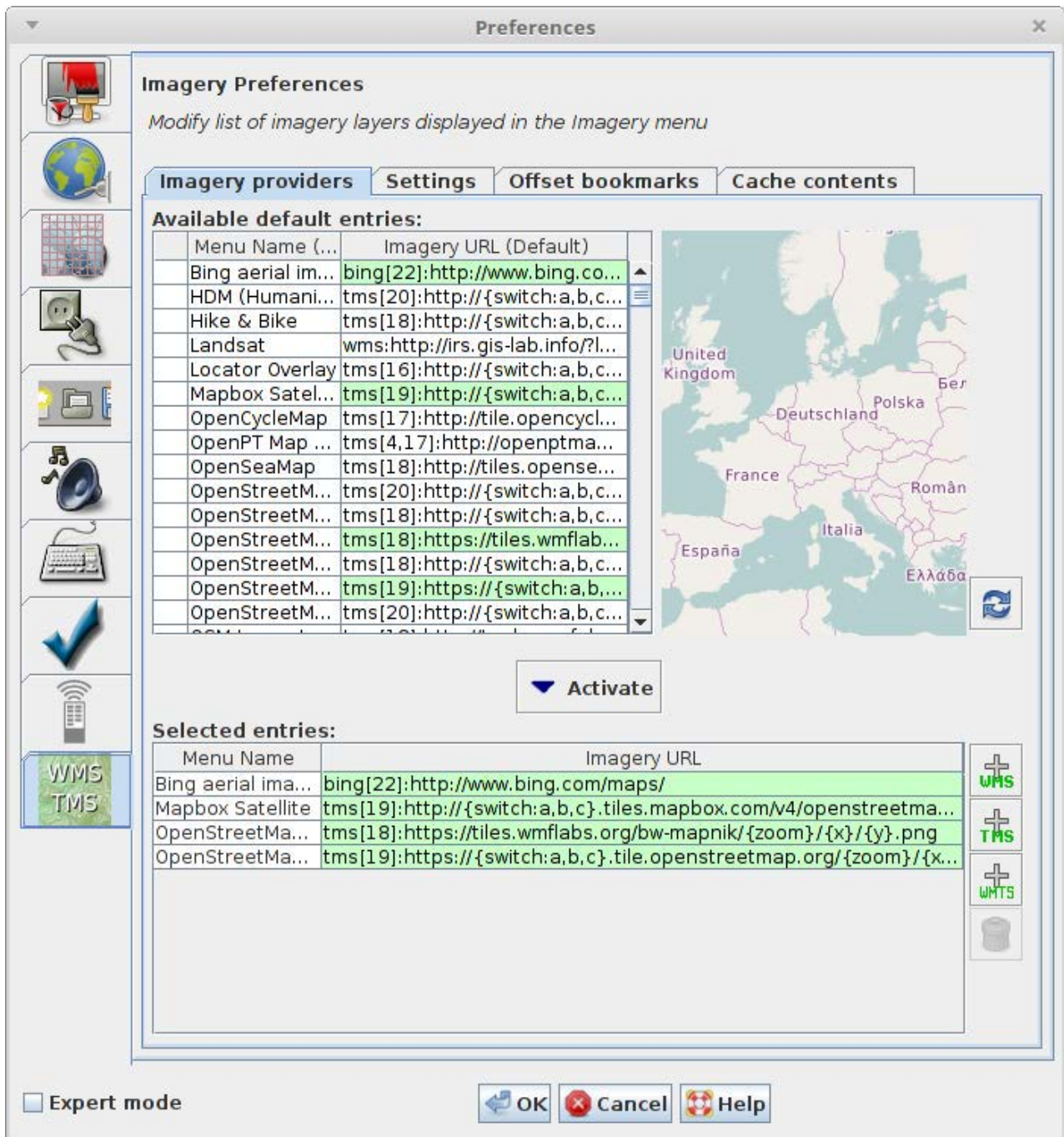


Figure 4. You can select the data sources used for downloading geographical imagery data.

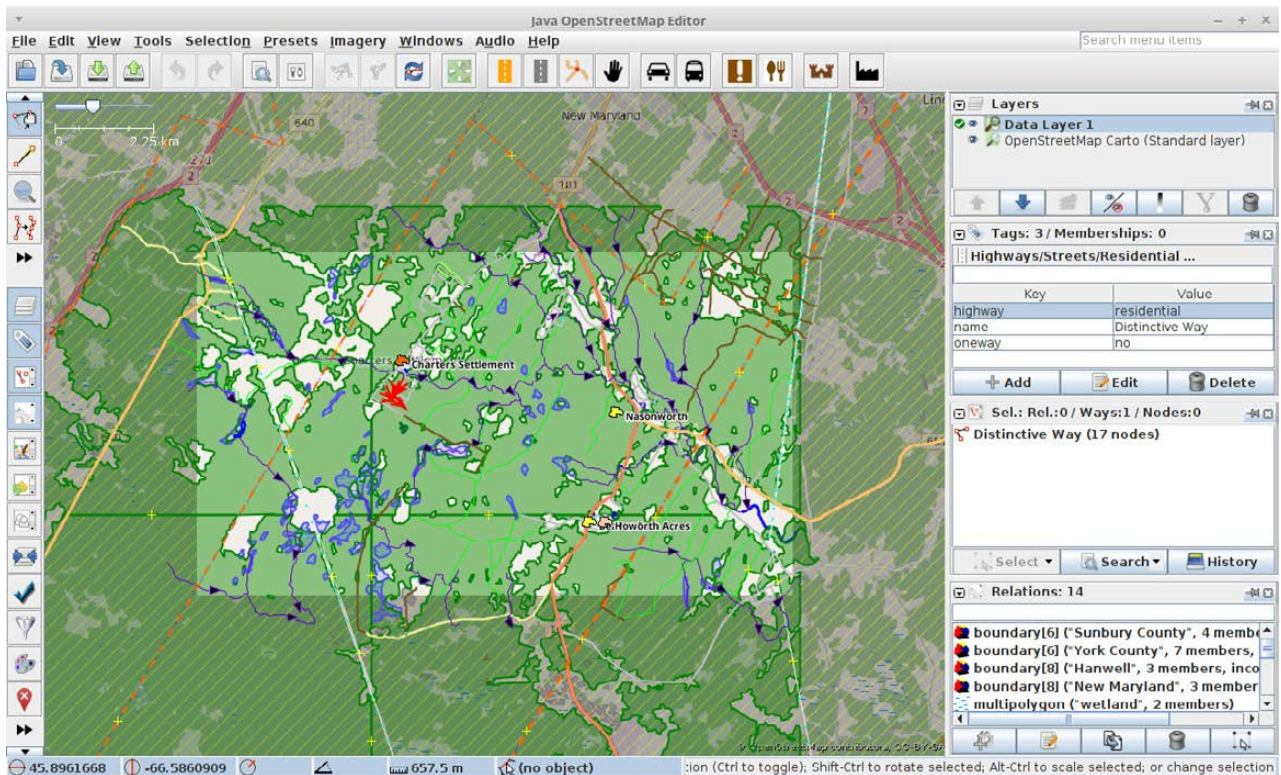


Figure 5. You can add multiple layers with satellite imagery or cartographic imagery.

available to you. Clicking the Imagery menu item gives you a menu of possible data sources where you can download geographical images. You can change this list by clicking the Imagery→Imagery Preferences menu item and selecting the sources you want to use.

In Figure 5, I selected the Imagery→OpenStreetMap Carto (Standard layer) menu item to get the cartographic imagery for my map area. You can choose the layer you want to work on by selecting it in the Layers pane at the top of the right-hand side. You also can change the order in which the layers are stacked or change whether they are visible. If you have your own imagery available, you can click on the Imagery→Rectified Image menu item to select the server from which your imagery is served.

What I have covered so far is fine if you just want static maps of pre-existing data, but that isn't very interesting. The last bit of functionality I want to cover here is how to add your own data to these maps.

You can add nodes, or a series of nodes, by selecting the appropriate tool from the list of icons at the top of the left-hand side. When you

add a node, you can tag it as a particular type of node or group of nodes. You can access several preset types by clicking the Presets menu item. These presets include natural objects, man-made objects and lots of geological features. This allows you to start adding tracks, objects and other features that are based on local knowledge, which means that you can create very specialized maps of your area.

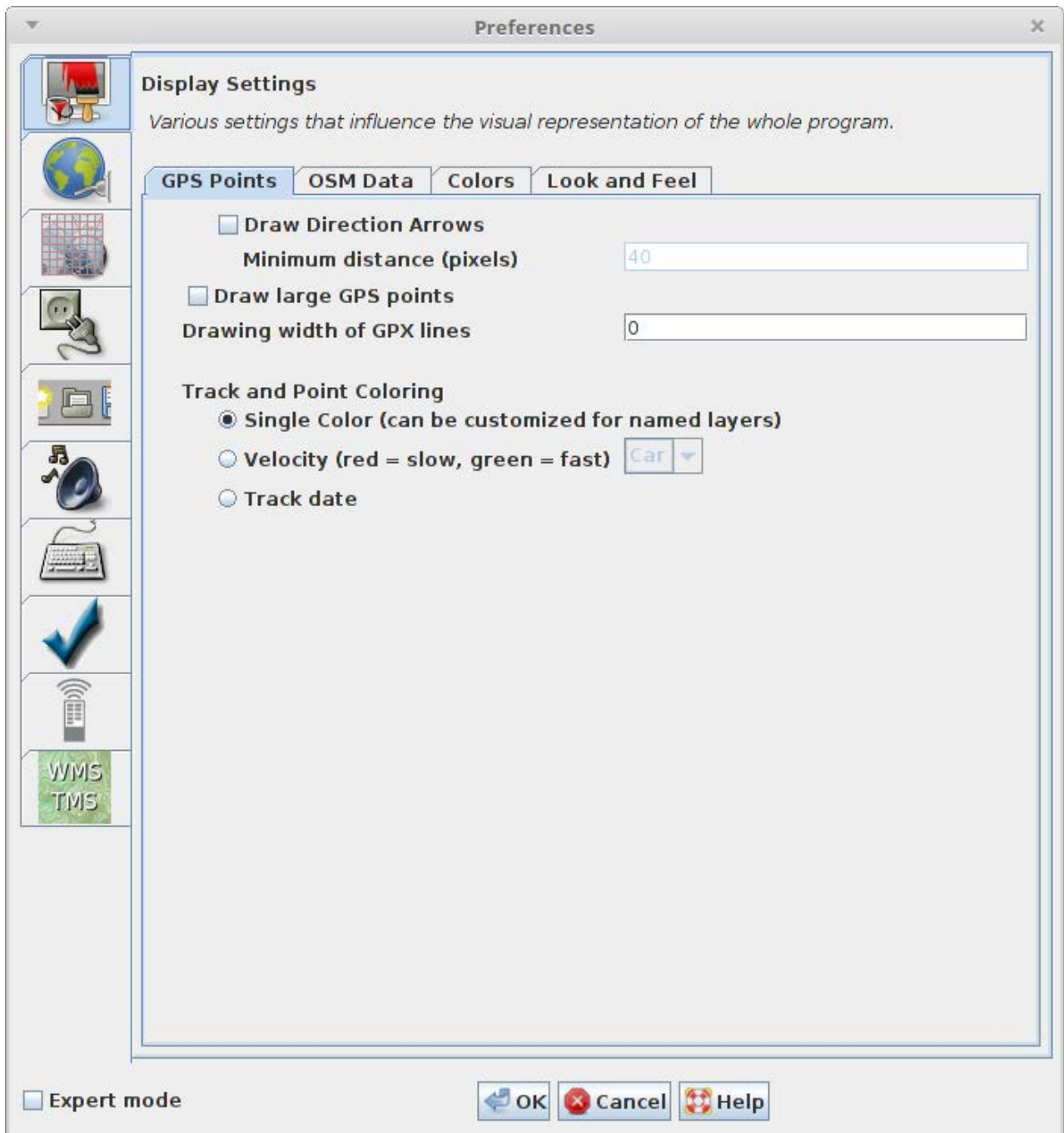


Figure 6. The preferences window lets you tweak a lot of the functionality in JOSM.

Clicking the Edit→Preferences menu item brings up the preferences window for JOSM. You can tweak several options to change how JOSM works. The Map Settings tab lets you change the main display options, such as the projection or the mapping styles available for use in JOSM.

As I mentioned earlier, a large selection of plugins is available from

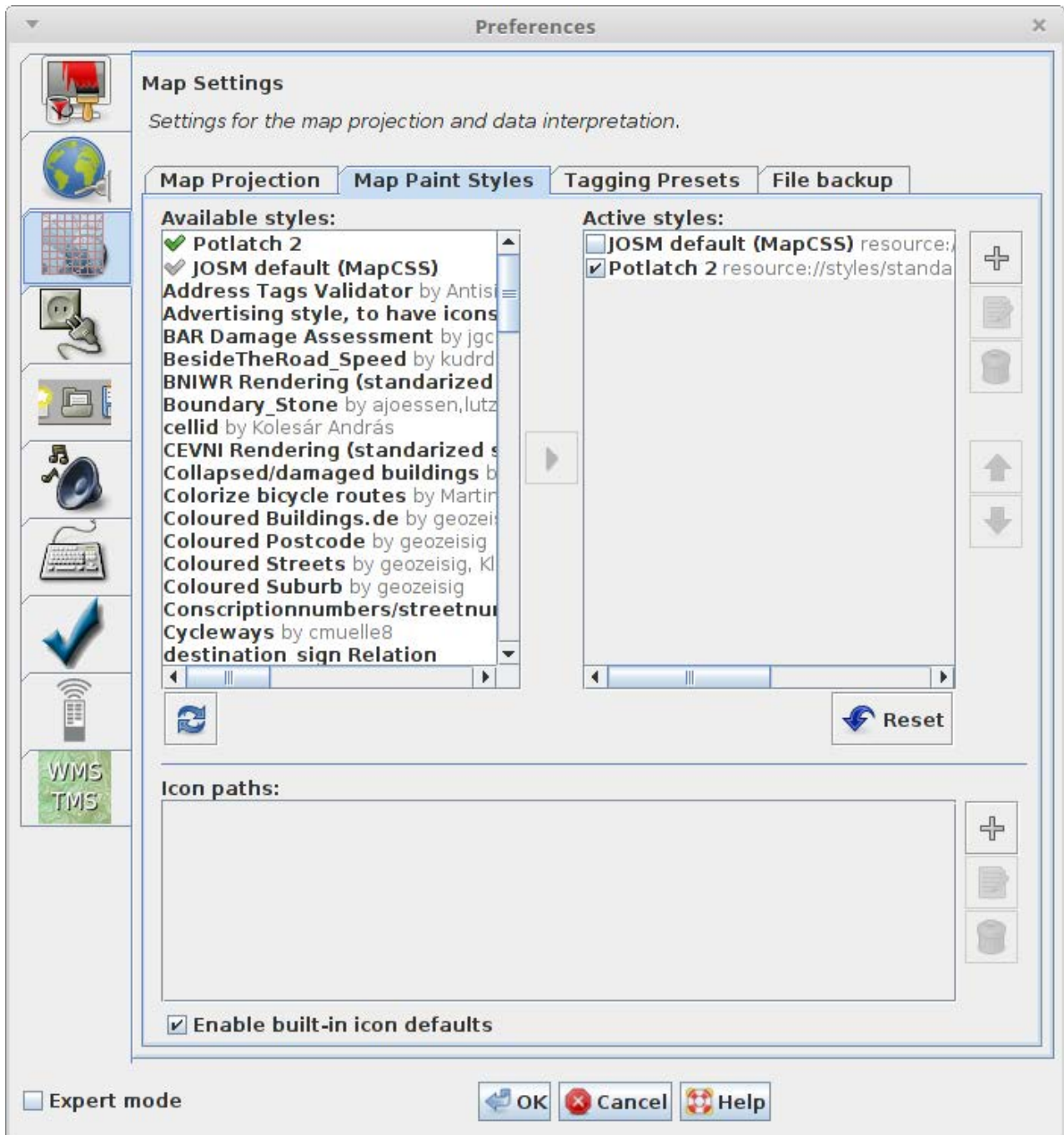


Figure 7. The Map Settings tab lets you select and download alternate mapping styles for your map display.

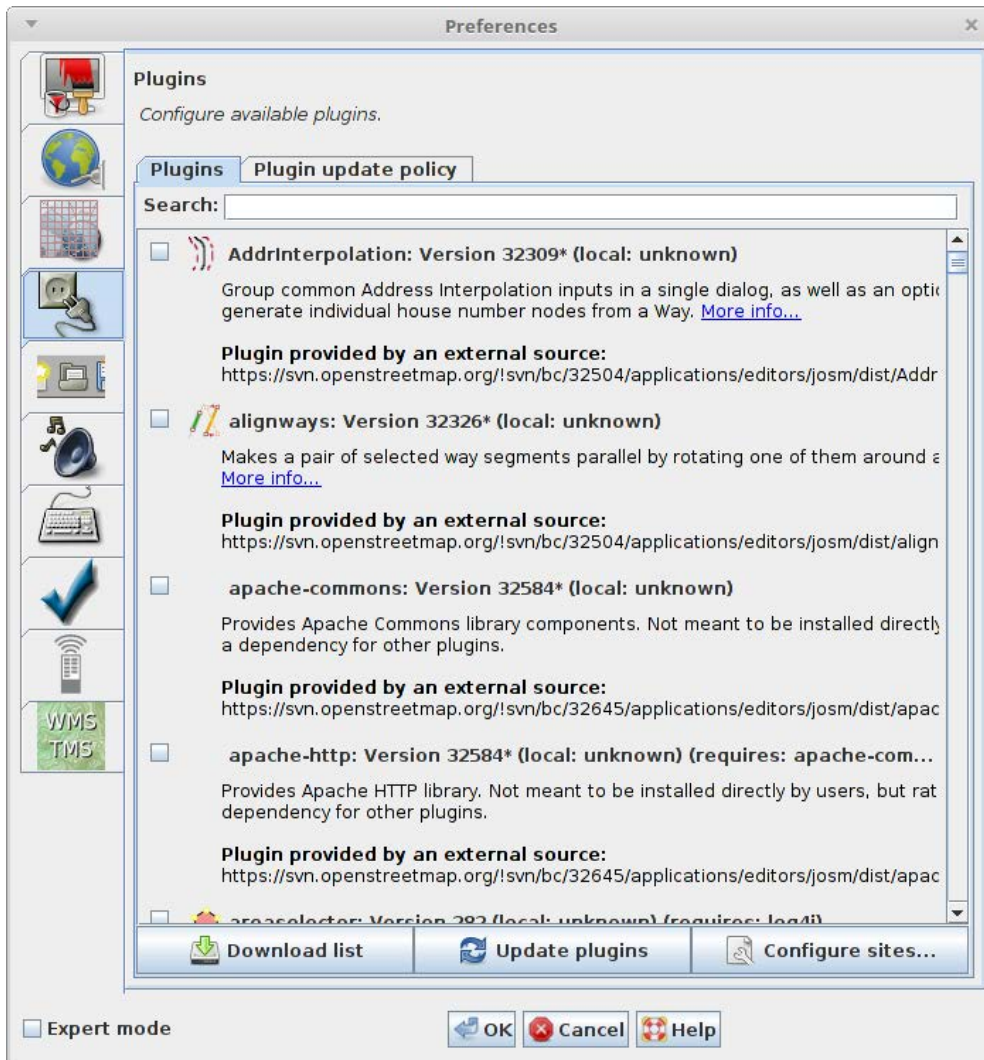


Figure 8. The Plugins tab lets you select from a large library of available plugins.

the community of JOSM users. You can access this library from the Plugins tab, where you can download and activate the plugins that interest you. Once you start looking at the available options, you'll see that there is a lot of extended functionality available for doing some serious work with your maps.

Once you are happy with a map, you'll want to save your work. JOSM can save all of the data related to your map in one of many different geographical file formats. Clicking on File→Save As pops up a save window where you can select from file formats such as GPX, GeoJSON or OSM Server Files. You also can upload changes back to the OpenStreetMaps servers if you are adding previously unknown information.—Joey Bernard



PREVIOUS
UpFront

NEXT

Reuven M. Lerner's
At the Forge



Be Kind, Buffer!

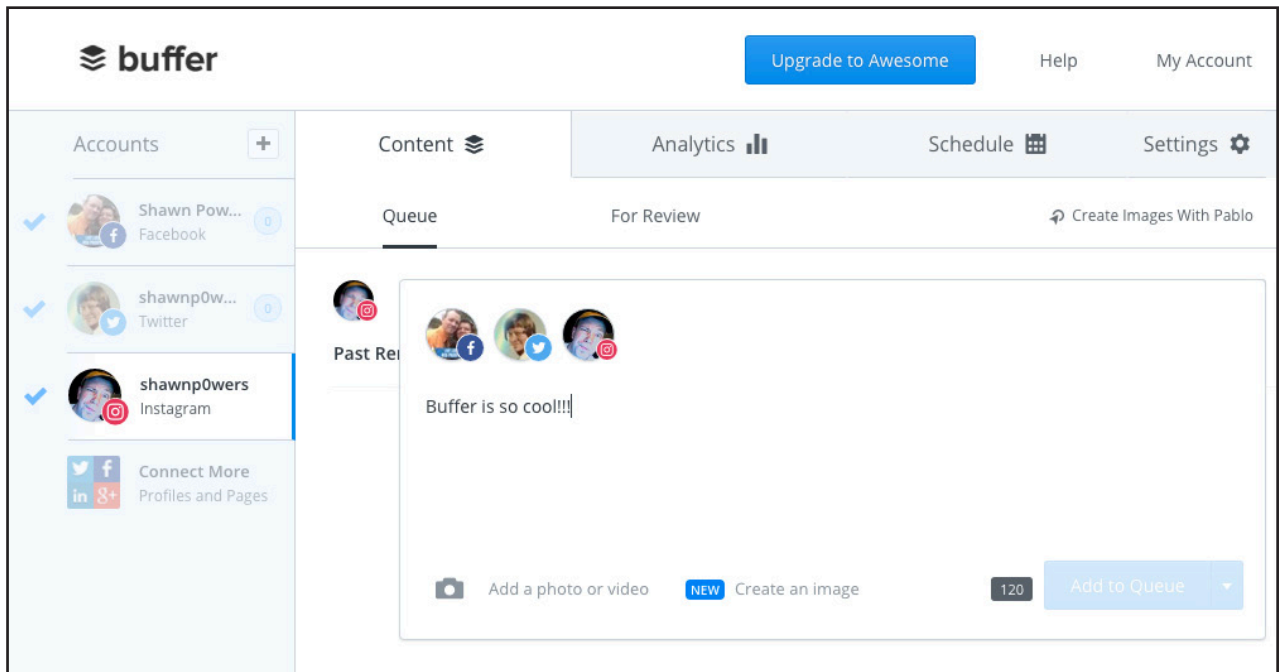


I like to tweet. Not like a bird (well, not usually), but tweeting on Twitter. I like to post silly pictures and say silly things. Unfortunately, a few things usually happen:

1. I take a bunch of photos within minutes of each other.
2. I want to post to Twitter and Facebook.
3. People don't want to see ten tweets from me in rapid succession.

I used to use a complicated combination of Instagram, If This Then That (<https://ifttt.com>) and a third-party Twitter client in order to post once and have it go to multiple social media sites. That didn't solve the problem of posting too often. It also made posting plain text vs. photos challenging.

Thankfully, Buffer solves all my problems with a cool app and website. Basically, you set up "times" throughout the day that you want to post your tweets/Facebook posts. You can "buffer" as many posts as you want, but the Buffer program will send them to the social-media sites only one at a time, at the appointed scheduled time. It also will accept plain text or photos, so I can use the same method for posting no matter what the media.



There are lots of interesting abilities too. For example, on the website, you can have Buffer analyze your social-media accounts to find the best time to post throughout the day. I actually prefer to set the times myself, but if you're looking for maximum "reach" for your posts, I imagine Buffer's algorithm is nice.

My favorite feature is that Buffer works on iOS, Android and the web. So no matter where I am, or what device I'm using, I can post to my social-media sites and know they'll be delivered in a non-annoying way. There also are paid features for folks who want more out of their social media, especially for companies that use it as a part of their marketing. For me though, the free features are absolutely perfect. Even if I had to pay, however, I think I'd still give Buffer this month's Editors' Choice award, because it has revolutionized the way I post to social media. And to my Twitter followers: you can thank Buffer for my apparently improved etiquette while posting!—Shawn Powers

[RETURN TO CONTENTS](#)

Preparing Data for Machine Learning

Before you can use machine-learning models, you need to clean the data.



**REUVEN M.
LERNER**

Reuven M. Lerner offers training in Python, Git and PostgreSQL to companies around the world. He blogs at <http://blog.lerner.co.il>, tweets at @reuvenmlerner and curates <http://DailyTechVideo.com>. Reuven lives in Modi'in, Israel, with his wife and three children.

◀ PREVIOUS
Editors' Choice

NEXT ▶
Dave Taylor's
Work the Shell

WHEN I GO TO AMAZON.COM, THE ONLINE STORE OFTEN RECOMMENDS PRODUCTS I SHOULD BUY.

I know I'm not alone in thinking that these recommendations can be rather spooky—often they're for products I've already bought elsewhere or that I was thinking of buying. How does Amazon do it? For that matter, how do Facebook and LinkedIn know to suggest that I connect with people whom I already know, but with whom I haven't yet connected online?

The answer, in short, is "data science", a relatively

new field that marries programming and statistics in order to make sense of the huge quantity of data we're creating in the modern world. Within the world of data science, machine learning uses software to create statistical models to find correlations in our data. Such correlations can help recommend products, predict highway traffic, personalize pricing, display appropriate advertising or identify images.

So in this article, I take a look at machine learning and some of the amazing things it can do. I increasingly feel that machine learning is sort of like the universe—already vast and expanding all of the time. By this, I mean that even if you think you've missed the boat on machine learning, it's never too late to start. Moreover, everyone else is struggling to keep up with all of the technologies, algorithms and applications of machine learning as well.

For this article, I'm looking at a simple application of categorization and "supervised learning", solving a problem that has vexed scientists and researchers for many years: just what makes the perfect burrito? Along the way, you'll hopefully start to understand some of the techniques and ideas in the world of machine learning.

The Problem

The problem, as stated above, is a relatively simple one to understand: burritos are a popular food, particularly in southern California. You can get burritos in many locations, typically with a combination of meat, cheese and vegetables. Burritos' prices vary widely, as do their sizes and quality. Scott Cole, a PhD student in neuroscience, argued with his friends not only over where they could get the best burritos, but which factors led to a burrito being better or worse. Clearly, the best way to solve this problem was by gathering data.

Now, you can imagine a simple burrito-quality rating system, as used by such services as Amazon: ask people to rate the burrito on a scale of 1–5. Given enough ratings, that would indicate which burritos were best and which were worst.

But Cole, being a good researcher, understood that a simple, one-dimensional rating was probably not sufficient. A multi-dimensional rating system would keep ratings closer together (since they would be more focused), but it also would allow him to understand which aspects of a

burrito were most essential to its high quality.

The result is documented on Cole's GitHub page (<https://srcole.github.io/100burritos>), in which he describes the meticulous and impressive work that he and his fellow researchers did, bringing tape measures and scales to lunch (in order to measure and weigh the burritos) and sacrificing themselves for the betterment of science.

Beyond the amusement factor—and I have to admit, it's hard for me to stop giggling whenever I read about this project—this can be seen as a serious project in data science. By creating a machine-learning model, you can not only describe burrito quality, but you also can determine, without any cooking or eating, the quality of a potential or theoretical burrito.

The Data

Once Cole established that he and his fellow researchers would rate burritos along more than one dimension, the next obvious question was: which dimensions should be measured?

This is a crucial question to ask in data science. If you measure the wrong questions, then even with the best analysis methods, your output and conclusions will be wrong. Indeed, a fantastic new book, *Weapons of Math Destruction* by Cathy O'Neil, shows how the collection and usage of the wrong inputs can lead to catastrophic results for people's jobs, health care and safety.

So, you want to measure the *right* things. But just as important is to measure *distinct* things. In order for statistical analysis to work, you have to ensure that each of your measures is independent. For example, let's assume that the size of the burrito will be factored in to the quality measurement. You don't want to measure both the volume and the length, because those two factors are related. It's often difficult or impossible to separate two related factors completely, but you can and should try to do so.

At the same time, consider how this research is being done. Researchers are going into the field (which is researcher-speak for "going out to lunch") and eating their burritos. They might have only one chance to collect data. This means it'll likely make sense to collect more data than necessary, and then use only some of it in creating the model. This is known as "feature selection" and is an important aspect of building a

machine-learning model.

Cole and his colleagues decided to measure ten different aspects of burrito quality, ranging from volume to temperature to salsa quality. They recorded the price as well to see whether price was a factor in quality. They also had two general measurements: an overall rating and a recommendation. All of these measurements were taken on a 0–5 scale, with 0 indicating that it was very bad and 5 indicating that it was very good.

It's important to point out the fact that they collected data on more than ten dimensions doesn't mean all of those measurements needed to be included in the model. However, this gave the researchers a chance to engage in feature selection, determining which factors most most affected the burrito quality.

I downloaded Cole's data, in which 30 people rated more than 100 burritos at 31 different restaurants, from a publicly viewable spreadsheet in Google Docs, into a CSV file (burritos.csv). The spreadsheet's URL is <https://docs.google.com/spreadsheets/d/18HkrkIYz1bKpDLeL-kaMrGjAhUM6LeJMIACwEljCgaw/edit#gid=1703829449>.

I then fired up the Jupyter (aka IPython) Notebook, a commonly used tool in the data science world. Within the notebook, I ran the following commands to set up my environment:

```
%pylab inline                # load NumPy, display
                               # Matplotlib graphics
import pandas as pd           # load pandas with an alias
from pandas import Series, DataFrame # load useful Pandas classes
df = pd.read_csv('burrito.csv') # read into a data frame
```

At this point, the Pandas data frame contains all the information about the burritos. Before I could continue, I needed to determine which fields were the inputs (the “independent variables”, also known as “predictors”) and which was the output (the “dependent variable”).

For example, let's assume that the burritos were measured using a single factor, namely the price. The price would be the input/independent variable, and the quality rating would be the output/dependent variable.

The model then would try to map from the input to the output.

Machine learning (and statistical models) works the same way, except it uses multiple independent variables. It also helps you determine just how much of an influence each input has on the output.

First, then, you'll need to examine your data, and identify which column is the dependent (output) variable. In the case of burritos, I went with the 0–5 overall rating, in column X of the spreadsheet. You can see the overall rating within Pandas with:

```
df['overall']
```

This returns a Pandas series, representing the average overall score from all of the samples at a particular restaurant.

Now that I have identified my output, which inputs should I choose? This is what I described earlier, namely feature selection. Not only do you want to choose a relatively small number of features (to make the model work faster), but you also want to choose those features that truly will influence the output and that aren't conflated with one another.

Let's start by removing everything but the feature columns. Instead of dropping the columns that I find uninteresting, I'll just create a new data frame whose values are taken from the interesting columns on this one. I'll want the columns with indexes of 11 through 23, which means that I can issue the following command in Pandas:

```
burrito_data = df[range(11,23)]
```

`range()` is a Python function that returns an iterator; in this case, the iterator will return 11 through 22 (that is, up to and not including 23). In this way, you can retrieve certain columns, in a smaller data frame. However, you still need to pare down your features.

Notice that my new data frame contains only the independent (input) variables; the overall score, which is our output variable, will remain on the side for now.

Feature Selection

Now that I have all of the input variables, which should I choose? Which

are more dependent on one another? I can create a “correlation matrix”, giving me a numeric value between 0 (uncorrelated) and 1 (totally correlated). If I invoke the “corr” method on the data frame, I’ll get a new data frame back, showing the correlations among all of them—with a correlation of 1.0 along the diagonal:

```
burrito_data.corr()
```

Now, it’s true that you can look through this and understand it to some degree. But it’s often easier for humans to understand images. Thus, you can use matplotlib, invoking the following:

```
plt.matshow(burrito_data.corr())
```

That produces a nice-looking, full-color correlation matrix in which the higher the correlation, the redder the color. The reddish squares show that (for example) there was a high correlation between the “length” and “volume” (not surprisingly), and also between the “meat” and the “synergy”.

Another consideration is this: how much does a particular input variable vary over time? If it’s always roughly the same, it’s of no use in the statistical model. For example, let’s assume that the price of a burrito is the same everywhere that the researchers ate. In such a case, there’s no use trying to see how much influence the price will have.

You can ask Pandas to tell you about this, using the “var” method on the data frame. When I execute `burrito_data.var()`, I get back a Pandas series object:

```
burrito_data.var()
```

```
Length          4.514376
Circum          2.617380
Volume          0.017385
Tortilla        0.630488
Temp            1.047119
Meat            0.797647
```

```
Fillings          0.765259
Meat:filling     1.084659
Uniformity       1.286631
Salsa            0.935552
Synergy          0.898952
Wrap             1.384554
dtype: float64
```

You can see that the burrito volume changes very little. So, you can consider ignoring it when it comes to building the model.

There's another consideration here, as well: is there enough input data from all of these features? It's normal to have some missing data; there are a few ways to handle this, but one of them is simply to try to work without the feature that's missing data. You can use the "count" method on the data frame to find which columns might have too much missing data to ignore:

```
burrito_data.count()
```

```
Length           127
Circum           125
Volume           121
Tortilla         237
Temp             224
Meat             229
Fillings         236
Meat:filling     231
Uniformity       235
Salsa            221
Synergy          235
Wrap             235
dtype: int64
```

As you can see, a large number of data points for the three inputs that have to do with burrito size are missing. This, according to Cole, is because the researchers didn't have a tape measure during many of their

outings. (This is but one of the reasons why I insist on bringing a tape measure with me whenever I go out to dinner with friends.)

Finally, you can ask scikit-learn to tell you which of these predictors contributed the most, or the least, to the outputs. You provide scikit-learn with inputs in a data frame and outputs in a series—for example:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
X = burrito_data
y = df[['23']]
```

In the above code, I import some objects I'll need in order to help with feature selection. I then use the names that are traditional in scikit-learn, X and y, for the input matrix and output series. I then ask to identify the most significant features:

```
sel = SelectKBest(chi2, k=7)
sel.fit_transform(X, y)
```

Notice that when invoking `SelectKBest`, you have to provide a value for “k” that indicates how many predictors you want to get back. In this way, you can try to reduce your large number of predictors to a small number. But if you try to run with the above, you'll encounter a problem. If there is missing data (NaN) in your input matrix, `SelectKBest` will refuse to run. So it's a good thing to discover which of your inputs are sometimes missing; if you remove those columns from the input matrix, you can use some feature reduction.

Cole and his colleagues did this sort of analysis and found that they could remove some of their input columns—the “flavor synergy”, as well as those having to do with burrito size. Having gone through the above process, I'm sure you can easily understand why.

Conclusion

Now that you have a good data set—with an input matrix and an output series—you can build a model. That involves choosing one or more

algorithms, feeding data into them and then testing the model to ensure that it's not overfit.

In my next article, I plan to do exactly that—take the data from here and see how to build a machine-learning model. I hope that you'll see just how easy Python and scikit-learn make the process of doing the actual development. However, I'll still have to spend time thinking about what I'm doing and how I'm going to do it, as well as which tools are most appropriate for the job. ■

RESOURCES

I used Python (<http://python.org>) and the many parts of the SciPy stack (NumPy, SciPy, Pandas, matplotlib and scikit-learn) in this article. All are available from PyPI (<http://PyPI.python.org>) or from <http://scipy.org>.

I recommend a number of resources for people interested in data science and machine learning.

One long-standing weekly email list is “KDNuggets” at <http://kdnuggets.com>. You also should consider the Data Science Weekly newsletter (<http://datascienceweekly.com>) and This Week in Data (<https://datarepublicblog.com/category/this-week-in-data>), describing the latest data sets available to the public.

I am a big fan of podcasts and particularly love “Partially Derivative”. Other good ones are “Data Stores” and “Linear Digressions”. I listen to all three on a regular basis and learn from them all.

If you're looking to get into data science and machine learning, I recommend Kevin Markham's Data School (<http://dataschool.org>) and Jason Brownlie's “Machine Learning Mastery” (<http://MachineLearningMaster.com>), where he sells a number of short, dense but high-quality ebooks on these subjects.

As I mentioned in the body of this article, Cathy O'Neil's new book, *Weapons of Math Destruction*, was thought-provoking and interesting, as well as disturbing. I highly recommend it.

Finally, thanks are due to Scott Cole, whose burrito-rating work is marvelously prepared, written and executed, and who shared his results with the online community for everyone's professional and culinary benefit.

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

[RETURN TO CONTENTS](#)

drupalize.me

Instant Access to Premium Online Drupal Training

- ✓ *Instant access to hundreds of hours of Drupal training with new videos added every week!*
- ✓ *Learn from industry experts with real world experience building high profile sites*
- ✓ *Learn on the go wherever you are with apps for iOS, Android & Roku*
- ✓ *We also offer group accounts. Give your whole team access at a discounted rate!*

Learn about our latest video releases and offers first by following us on Facebook and Twitter (@drupalizeme)!

Go to <http://drupalize.me> and get Drupalized today!



Wrapping Up the Mars Lander

Dave finishes the script for his Martian lander game and offers suggestions on how you can make improvements on your own.



DAVE TAYLOR

Dave Taylor has been hacking shell scripts on UNIX and Linux systems for a really long time. He's the author of *Learning Unix for Mac OS X* and the popular shell scripting book *Wicked Cool Shell Scripts*. He can be found on Twitter as @DaveTaylor, and you can reach him through his tech Q&A site: <http://www.AskDaveTaylor.com>.

PREVIOUS

◀ Reuven M. Lerner's
At the Forge

NEXT

Kyle Rankin's
Hack and / ▶

IN MY LAST FEW ARTICLES, I've been building a variant on the classic video game *Lunar Lander*, with a few simplifications and one big change: Martian gravity instead of lunar gravity. The moon is 1/6th of Earth's gravity; whereas Mars is about 1/3 of Earth's gravity, which makes flying a lander in for a soft descent a bit more exciting.

The tricky one might be to simulate a black hole, but that's easy to do by having a really, really big gravitational value, but not so easy to land safely. It's not hugely interesting, actually, unless you're working on the script to *Interstellar 2* perhaps.

WORK THE SHELL

The starting parameters of the game have Martian gravity set to 3.722 meters/sec/sec, and the spaceship enters the atmosphere at an altitude of 500 meters (about 1/3 mile). Do the math, and that means players have just more than 15 seconds to avoid crashing onto the Martian surface.

Creating the Interface

My last article (in the October 2016 issue of *LJ*) ended with a demonstration of code that offered second-by-second data on what was basically free fall through the Martian atmosphere:

```
1 seconds: speed: -3.722 m/s altitude: 496.278 meters.
2 seconds: speed: -7.444 m/s altitude: 488.834 meters.
3 seconds: speed: -11.166 m/s altitude: 477.668 meters.
4 seconds: speed: -14.888 m/s altitude: 462.780 meters.
5 seconds: speed: -18.610 m/s altitude: 444.170 meters.
```

That's not a great way to land unless you're in a really, really well padded couch. My first stab at adding an interesting interface is to stop each second and offer users the chance to specify whether they want to fire their retro-rockets and how much fuel to burn for the subsequent second.

Burn your fuel too early, and you could end up shooting off into space or level out just to plummet to the surface anyway. In this first version, however, the user will have unlimited fuel (though in real life it'd be limited, and the vessel would lighten up, decreasing gravitational pull, as the fuel was burned).

Here's the core of the code:

```
echo "$time seconds into flight: speed: $speed m/s \
    and altitude: $altitude meters."
echo -n "Fire retro rockets? (burn rate: 0-100): "
read thrust
if [ -z "$thrust" ] ; then
    thrust=0
fi
```

The last few lines allow the player simply to press Enter and have

WORK THE SHELL

that be the equivalent of a zero—easy enough. Now let's try to land the darn spaceship:

```
Time: 1: Speed: -3.722 m/s and altitude: 496.278 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 2: Speed: -7.444 m/s and altitude: 488.834 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 3: Speed: -11.166 m/s and altitude: 477.668 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 4: Speed: -14.888 m/s and altitude: 462.780 meters.  
Fire retro rockets? (burn rate: 0-100): 15
```

```
Time: 5: Speed: -3.610 m/s and altitude: 459.170 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 6: Speed: -7.332 m/s and altitude: 451.838 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 7: Speed: -11.054 m/s and altitude: 440.784 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 8: Speed: -14.776 m/s and altitude: 426.008 meters.  
Fire retro rockets? (burn rate: 0-100): 15
```

```
Time: 9: Speed: -3.498 m/s and altitude: 422.510 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 10: Speed: -7.220 m/s and altitude: 415.290 meters.  
Fire retro rockets? (burn rate: 0-100):
```

```
Time: 11: Speed: -10.942 m/s and altitude: 404.348 meters.  
Fire retro rockets? (burn rate: 0-100):
```

WORK THE SHELL

```
Time: 12: Speed: -14.664 m/s and altitude: 389.684 meters.  
Fire retro rockets? (burn rate: 0-100): 15
```

```
Time: 13: Speed: -3.386 m/s and altitude: 386.298 meters.  
Fire retro rockets? (burn rate: 0-100):
```

Notice here that I am being conservative with the fuel, firing the thrusters at 4, 8 and 12 seconds. This allows me to be 13 seconds into the descent and have a speed of only 3.386 m/s, while dropping from 500 meters to 386 meters. If the fuel holds out, this isn't a bad landing strategy!

Adding Some Limits and Constraints

Now, what if I add some basic constraints? What about limited fuel? And what about a cap on the maximum possible thrust so that you don't just decide to drop like a stone to the surface and apply huge amounts of thrust at the last second to pop up for a gentle landing?

In real life, of course, that'd likely produce more g-force than a human could survive, but I'm not going to worry about people falling unconscious while playing Martian Lander. It'd probably be bad for reviews anyway!

Still, starting with 100 fuel and a max thrust of 30 should make things interesting. This can be captured in a couple variables at the top of the script (and then could be changed with starting parameters, as desired).

The main mathematics of the script are captured in five lines, making it easy enough to understand:

```
speed=$( $bc <<< "scale=3; $speed + $gravity + $thrust" )  
thrust=0 # rocket fires second by second  
altitude=$( $bc <<< "scale=3; $altitude + $speed" )  
alt=$( echo "$altitude" | cut -d\. -f1 )  
time=$(( $time + 1 ))
```

Notice the `$alt` variable. That's the integer portion of the altitude for display. The script actually keeps a more accurate value as `$altitude`.

As a reminder, I'm using the `bc` binary calculator, and in the Linux

WORK THE SHELL

world, you need to specify how many digits of accuracy you want after the decimal point. The default is zero, making `bc` work like `expr`.

The math is straightforward once sufficiently simplified (one of the few times it's good not to work at NASA!), so most of the code deals with user interaction.

Before going there, however, let's start with a full listing of all starting parameters for the script:

```
speed=0           # > 0 is climbing, < 0 is falling
gravity="-3.722"  # gravity pulls ya down
accel=0           # start with zero acceleration
height=500        # Note that 1609 meters = 1 mile
fuel=100          # limited fuel
maxthrust=30      # the ship, she canna handle greater!
maxheight=$(( 2 * $height )) # above you shoot into space
altitude=$height  # current altitude above the surface
alt=$altitude     # integer value of altitude
thrust=0
outoffuel=0       # not yet true
```

With these values all specified, the main input loop of the game is captured in about 20 lines:

```
if [ $alt -gt $maxheight ] ; then
    echo "Well heck. You used too much thrust and have \
        escaped the gravitational pull of Mars. You're \
        doomed to float off into space and die. Bummer."
    exit 1
elif [ $alt -gt 0 ] ; then
    echo "Time: ${time}: Speed: $speed m/s and \
        altitude: $altitude meters."
    if [ $fuel -gt 0 ] ; then
        echo -n "Fire retro rockets? (burn: 0-{$maxthrust}): "
        read thrust
        echo ""
        if [ -z "$thrust" ] ; then
```


WORK THE SHELL

```
    thrust=0
elif [ $thrust -gt $maxthrust ] ; then
    echo "*** Ya canna handle that much thrust! \
        Reset to $maxthrust" ; echo ""
    thrust=$maxthrust
fi
fuel=$(( $fuel - $thrust ))      # burn, baby
else
    if [ $outoffuel -eq 0 ] ; then
        echo "( out of fuel )"
        outoffuel=1
    fi
fi
fi
```

Notice that if the user specifies too much thrust, the program just resets it to `$maxthrust`—safety, you know. Otherwise, the code above should be pretty easy to understand (and do note also that long, mnemonic variable names always make code more readable!)

And finally, let's give it a whirl:

```
Time: 1: Speed: -3.722 m/s and altitude: 496.278 meters.
Fire retro rockets? (burn rate: 0-30): 50
```

```
*** Ya canna handle that much thrust! Reset to 30
```

```
Time: 2: Speed: 22.556 m/s and altitude: 518.834 meters.
Fire retro rockets? (burn rate: 0-30): 30
```

```
Time: 3: Speed: 48.834 m/s and altitude: 567.668 meters.
Fire retro rockets? (burn rate: 0-30): 30
```

```
Time: 4: Speed: 75.112 m/s and altitude: 642.780 meters.
Fire retro rockets? (burn rate: 0-30): 30
```

```
Time: 5: Speed: 101.390 m/s and altitude: 744.170 meters.
```

WORK THE SHELL

```
( out of fuel )
```

```
Time: 6: Speed: 97.668 m/s and altitude: 841.838 meters.
```

```
Time: 7: Speed: 93.946 m/s and altitude: 935.784 meters.
```

```
Well heck. You used too much thrust and have escaped the  
gravitational pull of Mars. You're doomed to float off  
into space and die. Bummer.
```

Oops—way too much retro rocket, and I ran out of fuel only five seconds into the descent! Bummer indeed.

Mods and Improvements

I encourage you to work on this script yourself to see what you can do with it that's interesting. One possibility is a simple input script that lets users specify times and burns and have them all applied automatically (this could be as easy as `time:burn time:burn` as a starting parameter).

For realism, you also could go back and calculate gravitational pull as a function of the weight of the ship + fuel, so that as you burn fuel, the pull of the Martian surface diminishes. Or, that might be too much physics!

Another possibility: make gravity an optional starting parameter so that you can create Venusian Lander, Neptune Lander and so on too. While you're at it, you could let the player specify max thrust and fuel from the command line too.

In any case, good luck with your Martian Lander. In my next article, I'll move off in a completely new direction—which might possibly still involve the moon. ■

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

[RETURN TO CONTENTS](#)


O'REILLY®

Software Architecture

CONFERENCE

November 13-16, 2016
San Francisco, CA

Practical training in the tools, techniques, and leadership skills needed to build a solid foundation in the evolving world of software architecture.



“Architects are the decision makers on projects, from frameworks and libraries to Continuous Delivery pipelines. They make the decisions that are hard to change later.”

—Neal Ford, Software Architecture Conference Co-Chair

**Click here to
Save 20%**
with code **PCLinuxJournal**

Simple Server Hardening, Part II

Expand past specific hardening steps into more general practices you can apply to any environment.



KYLE RANKIN

Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

PREVIOUS

◀ Dave Taylor's Work the Shell

NEXT

▶ Shawn Powers' The Open-Source Classroom

IN MY LAST ARTICLE, I talked about the classic, complicated approach to server hardening you typically will find in many hardening documents and countered it with some specific, simple hardening steps that are much more effective and take a only few minutes. While discussing how best to harden SSH and sudo can be useful, in a real infrastructure, you also have any number of other services you rely on and also want to harden.

So instead of choosing specific databases, application servers or web servers, in this follow-up article, I'm going to extend the topic of simple

hardening past specific services and talk about more general approaches to hardening that you can apply to software you already have running as well as to your infrastructure as a whole. I start with some general security best practices, then talk about some things to avoid and finally finish up with looking at some areas where sysadmin and security best practices combine.

General Best Practices

I won't dwell too long on general security best practices, because I've discussed them in other articles in the past, and you likely have heard of them before. That said, it's still worth mentioning a few things as these are the principles you should apply when you evaluate what practices to put in place and which to avoid. As someone who likes running a tight ship when it comes to systems administration, it's nice that security best practices often correspond with general best practices. In both cases, you generally can't cut corners, and shortcuts have a tendency to bite you later on.

The first security best practice worth covering is the principle of least privilege. This principle states that people should have the minimum level of privileges to a system that they need and no more than that. So for instance, if you don't need to grant all engineers in your organization sudo root privileges on your servers, you shouldn't. Instead, just give them privileges to perform the tasks they need. If some classes of engineers don't really need accounts at all, it's better not to create accounts for them. Some environments are even able to get by without any developer accounts in production.

The simpler a system, the easier it should be to secure. Complexity not only makes troubleshooting more difficult, it also makes security difficult as you try to think through all of the different attack scenarios and ways to prevent them. Along with that simplicity, you should add layers of defense and not rely on any individual security measure. For instance, traditionally organizations would harden a network by adding a firewall in front of everything and call it a day. These days, security experts advise that the internal network also should be treated as a threat. Sometimes attackers can bypass a security measure due to a security bug, so if you have layers of defense, they may get past one security measure but then

have to deal with another.

On the subject of security bugs, keeping the software on your systems patched for security bugs is now more important than ever. The time between a security bug's discovery and being exploited actively on the internet keeps shrinking, so if you don't already have a system in place that makes upgrading software throughout your environment quick and easy, you should invest in it.

Finally, you should encrypt as much as possible. Encrypt data at rest via encrypted filesystems. Encrypt network traffic between systems. And when possible, encrypt secrets as they are stored on disk.

What to Avoid

Along with best practices, some security practices are best avoided. The first is security by obscurity. This means securing something merely by hiding it instead of hardening it. Obscurity should be avoided because it doesn't actually stop an attack; it just makes something harder to find and can give you a false sense of security.

A great example of this is the practice of moving your SSH port from the default (22) to something more obscure. Although moving SSH to port 60022 might lower the number of brute-force attempts in your logs, if you have a weak password, any halfway-decent attacker will be able to find your SSH port with a port scan and service discovery and be able to get in.

Port knocking (the practice of requiring a service to access random ports on the server in a sequence before the firewall allows the client through—think of it like a combination lock using ports) also falls into this category, because any router between the client and server can see what port the client uses—they aren't a secret—but will give you a false sense of security that your service is firewalled off from attack. If you are that concerned about SSH brute-force attacks, just follow my hardening steps from the first part of this series in the October 2016 issue of *LJ* to eliminate it as an attack completely.

Many of the other practices to avoid are essentially the opposites of the best practices. You should avoid complexity whenever possible, and avoid reliance on any individual security measure (they all end up having a security bug or failing eventually). In particular, when choosing

network software, you should avoid software that doesn't support encrypted communication. I treat network software that doesn't support encryption in this day and age as a sign that it's still a bit too immature for production use.

Where Sysadmin and Security Best Practices Collide

Earlier, I mentioned that general best practices and security best practices often are the same, and this first tip is a great example. Centralized configuration management tools like Puppet, Chef and SaltStack are tools systems administrators have used for quite some time to make it easier to deploy configuration files and other changes throughout their infrastructure. It turns out that configuration management also makes hardening simpler, because you can define your gold standard, hardened configuration files and have them enforced throughout your environment with ease.

For instance, if you use configuration management to control your web server configuration, you can define the set of approved, secure, modern TLS cipher suites and deploy them to all of your servers. If down the road one of those ciphers proves to be insecure, you can make the change in one place and know that it will go out to all relevant servers in your environment.

Another best practice with configuration management is checking your configuration management configuration files into a source control system like git. This "configuration as code" approach has all kinds of benefits for systems administrators, including the ability to roll back mistakes and the benefit of peer review. From a security standpoint, it also provides a nice auditing trail of all changes in your environment—especially if you make a point to change your systems only through configuration management.

Along with configuration management, another DevOps tool that also greatly aids security is an orchestration tool—whether it's MCollective, Ansible or an SSH for loop. Orchestration tools make it easy to launch commands from a central location that apply to particular hosts in your environment in a specific order and often are used to stage software updates. This ease of deploying software also provides a great security benefit because it's very important to stay up

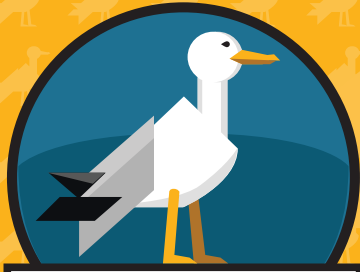
to date on security patches.

With an orchestration tool like MCollective, for instance, if you find out there's been a new bug in ImageMagick, you can get a report of the ImageMagick versions in your environment with one command, and with another command, you can update all of them. Regular security updates become simpler, which means you are more likely to stay up to date on them, and more involved security updates (like kernel updates that require a reboot) at least become more manageable, and you can use the orchestration software to tell for sure when all systems are patched.

Finally, set up some sort of centralized logging system. Although you can get really far with `grep`, it just doesn't scale when you have a large number of hosts generating a large number of logs. Centralized logging systems like Splunk and ELK (Elasticsearch, Logstash and Kibana) allow you to collect your logs in central place, index them and then search through them quickly. This provides great benefits to general sysadmin troubleshooting, but from a security perspective, centralized logging means attackers who compromise a system now have a much harder time covering up their tracks—they now have to compromise the logging systems too. With all of your logs searchable in one place, you also get the ability to build queries that highlight (and with many systems graph as well) suspicious log events for review. ■

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

[RETURN TO CONTENTS](#)



SeaGL

Seattle GNU/Linux Conference

Seattle's grassroots free software summit is on again!

Join speakers and participants from around the world for the fourth year of Seattle's free, as in freedom and beer, GNU/Linux Conference.

With over 50 talks and the inaugural Cascadia Community Builder Award, this year is sure to be a blast!

November 11th and 12th, 2016,
Seattle Central College campus
1701 Broadway Seattle, WA

→ Visit SeaGL.org for more information.

Graph Any Data with Cacti!

RRD and MRTG are confusing;
Cacti makes everything simple!



**SHAWN
POWERS**

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via email at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

PREVIOUS



Kyle Rankin's
Hack and /

NEXT

Susan Sons'
Under the Sink



FOR THE PAST FEW YEARS, I've been trying to understand how to make graphs using RRDtool (Round-Robin Database tool) after failing miserably to understand MRTG (Multi-Router Traffic Grapher) before that. The thing I like about RRDtool is that it's newer and supports a wider variety of data sources. It's still incredibly complicated though, and I've given up on learning how to use it on multiple occasions. That's when I discovered Cacti.

Cacti is not a new program. It's been around for a long time, and in its own way, it's a complicated beast itself. I finally really took the time to figure it out, however, and I realized that it's

not too difficult to use. The cool part is that Cacti makes RRDtool manipulation incredibly convenient. It did take me the better part of a day to understand Cacti fully, so hopefully this article will save you some time.

The Goal

I want to create a graph that graphs something automatically and does it using a bash script as the input as opposed to SNMP or anything like that. I've been using bash for years, and I'm comfortable using the command line to procure data. In fact, for this project, I'm going to adapt a script I use for BirdTopia (my continual birdcam project for the past few years) that will pull a temperature from the command line. I want to pull the temperature from two different cities and graph them together. For this example, I use Petoskey, Michigan (where I live), and Houston, Texas (where *Linux Journal* headquarters are located).

Here's the script:

```
#!/bin/bash
curl -s "http://api.wunderground.com/weatherstation/
↳WXCurrentObXML.asp?ID=$1" \
| grep temp_f | sed 's/./' | sed 's/./' | sed
↳'s/<temp_f>/' | sed 's/<\/temp_f>/'
```

It looks complex, but really it just downloads the API information from Weather Underground for the weather station given as an argument, and then uses `sed` (stream editor) to pare down the information to a simple number—specifically, the numerical degrees in Fahrenheit. If you prefer Celsius, I applaud your country for adopting the metric system, but sadly, my brain just can't relate Celsius to how warm the outdoor temperature feels.

One tricky part is figuring out what the proper weather station ID is for your city. I wish you could just use a ZIP code, but I've been unable to find a command-line weather API that will take a ZIP code. So if you're following along, just head over to <https://www.wunderground.com> and load the page for your locale. Once there, click on the link shown in

Figure 1 (your text will be different, but the location on the page should match). The next page will show the name of your local weather station. You can see mine in Figure 2.

To get the local temperature using the script, just type the name of the script (I named my “gettemp” and saved it as an executable



Figure 1. I assumed this was my weather station, but it’s not. You need to click through to find the code.

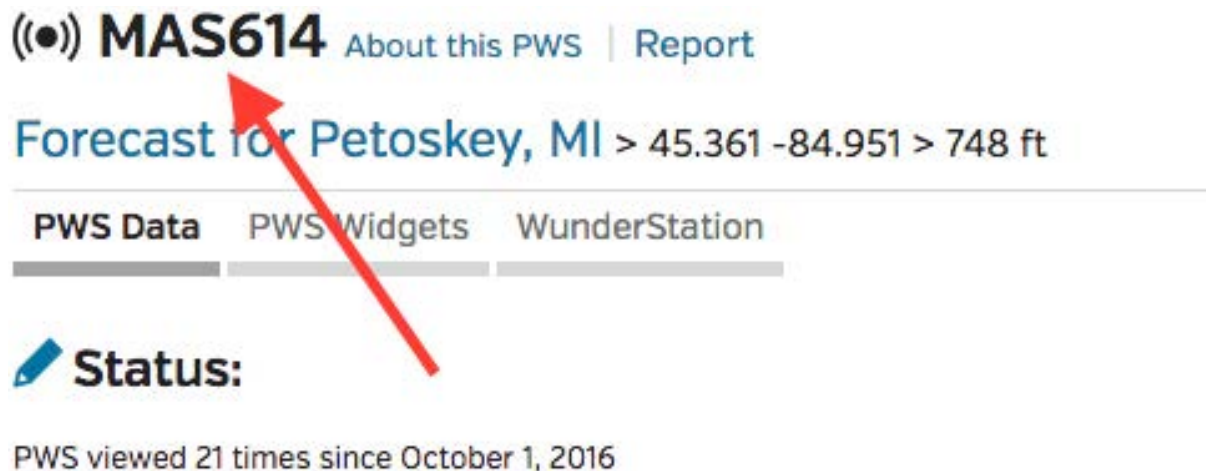


Figure 2. Here is the code for my local weather station. Be sure to try your script on the command line to see if you have the correct code.

in /usr/local/bin/) with the name of the weather station as an argument:

```
spowers@cacti:~$ gettemp MAS614
48.0
spowers@cacti:~$
```

The same script will work for Houston’s weather too. I looked up a weather station name in Houston and found “KTXGALEN6” as a name. Using that as the argument, I can get the current temp for Houston. And, those will be my two points of data.

How Cacti Works

This is honestly the most frustrating part of the process. There are so many different pieces to the Cacti puzzle, that it’s easy to give up. Go ahead and install Cacti on your system (it should be in the repository), and log in. The default login is usually “admin” for both login and password. You should change it immediately.

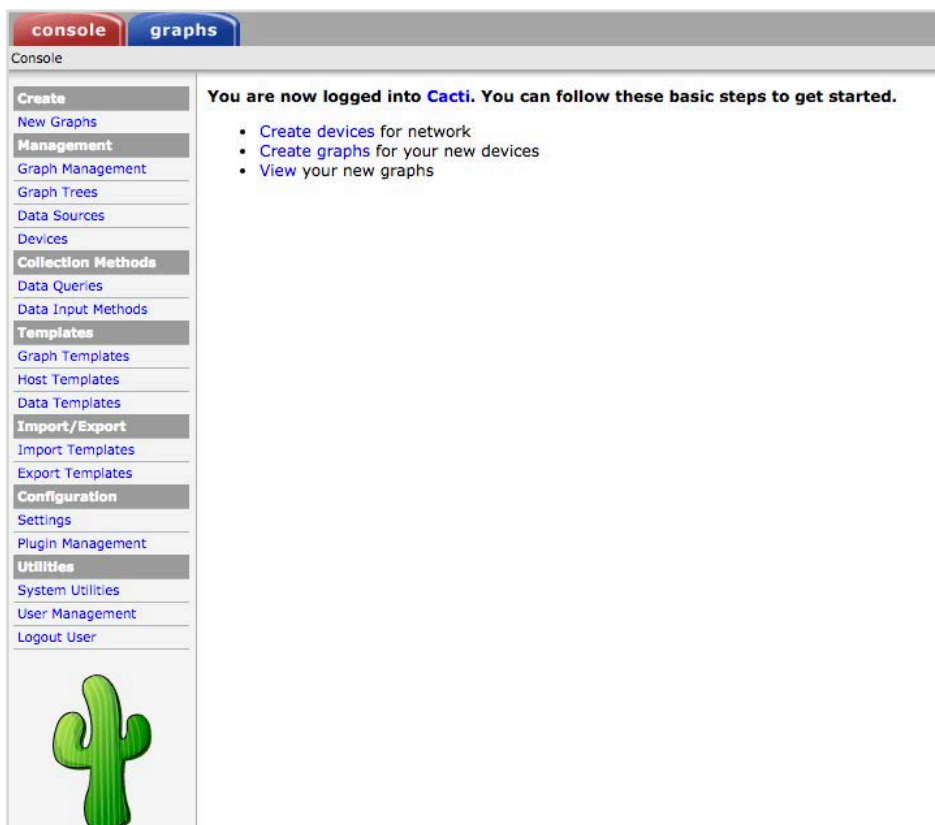


Figure 3. Don’t be overwhelmed; it’s not nearly as scary as it looks.

Once there, look along the left-hand side of the “console” page (Figure 3), which should load by default. The confusing part is that names like “devices”, “data sources”, “data queries” and such all sound like what you want to configure to get data. I’m listing the definitions below, so hopefully it’s less confusing:

- **Data Queries:** this generally refers to SNMP queries used to get data. I’m not using this here, even though it sounds like something I’d need to configure for doing data queries to a bash script.
- **Data Input Methods:** this is where you’ll configure your bash script if you’re following along. Cacti sees the script as a way to get data, not the data itself. This means you can use the script (or Data Input Method) for multiple data sources. In my case, I use the same Data Input Method for both Petoskey temps and Houston temps.
- **Devices:** Cacti allows you to categorize data by device. This makes sense if you’re going to monitor server data from a bunch of different servers. In this case, I’m not graphing different devices, so I won’t use the device categorization at all (I’ll use “none” as the device).
- **Data Sources:** a data source uses a “Data Input Method” to create a graphable piece of data. For this example, I have two data sources: the Petoskey temp and the Houston temp.
- **Data Templates:** this is a template that instructs Cacti on how to use a Data Input Method to create a Data Source. Basically, you set up a Data Template so Cacti knows what questions to ask when creating a Data Source. It’s possible to do without a Data Template, but if you don’t set one up, Cacti will give errors when creating a Data Source that you’ll need to go back and fix later. It’s a real pain, but the step to create a Data Template makes the process far less painful.

Hopefully, that clears up some of the strange terminology. Graphing is actually separate, so first you need to get your Data Sources configured. (You’ll make the graphs from your data sources, once they’re storing periodic data.)

Part I: Data Input Method

For this example, even though I have two data sources, I have only one Data Input Method. That means I need to configure the script in Cacti so that it will accept an argument (the weather station code) for each data source. So to do this, first click on “Data Input Methods” on the left, and then click “add” on the upper-right corner of the Data Input Methods page. Look at Figure 4 to see where the “add” link is, because it took me a long time to find it at first!

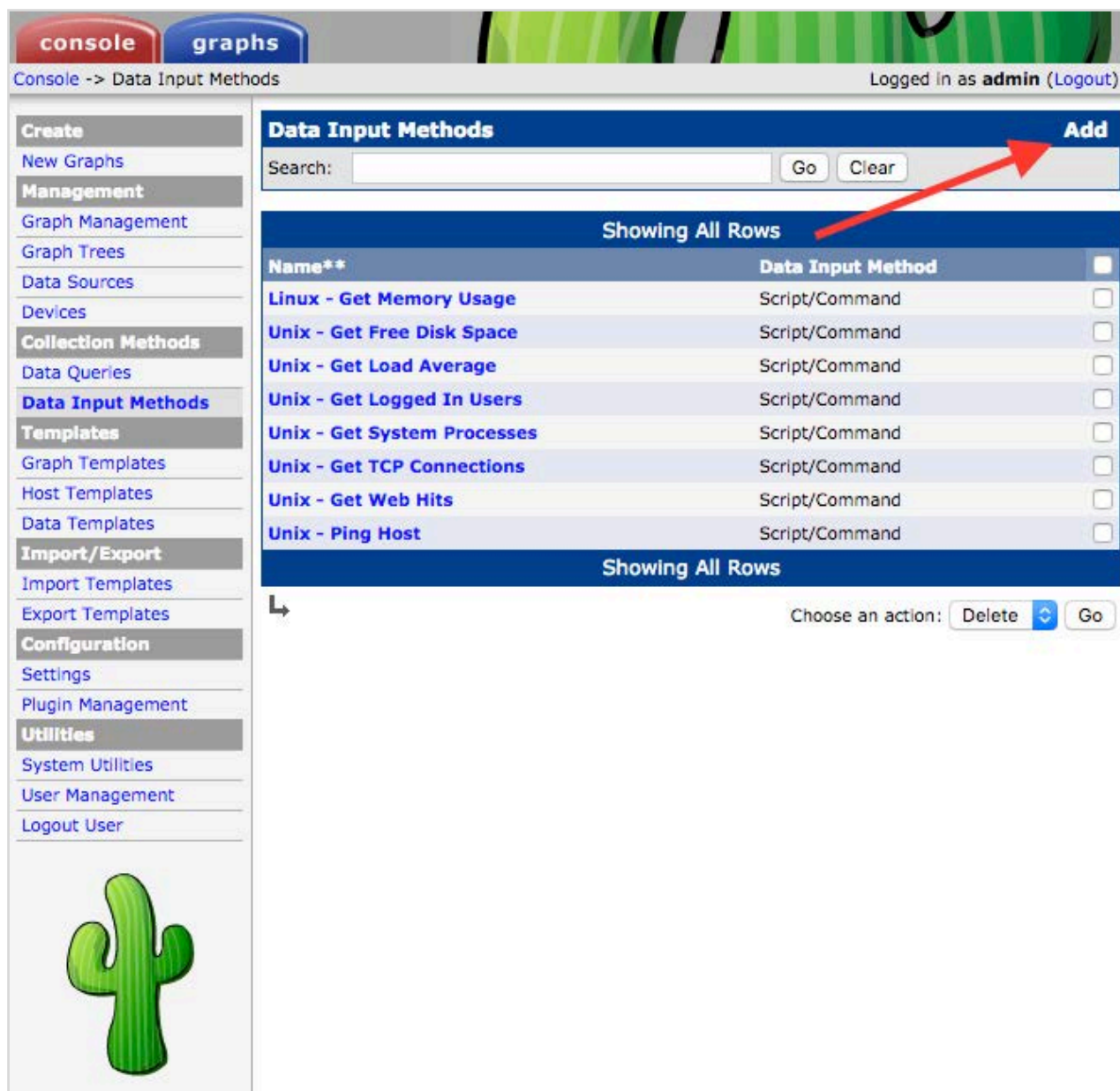


Figure 4. It took me forever to figure out how to add something. I felt very silly.

THE OPEN-SOURCE CLASSROOM

This is where you create the criteria for Cacti to “call” your script. Figure 5 shows how to set up the information. Note that you must use the “Script/Command” Input Type and that the placeholder for the argument goes in < > brackets. The name of the Data Input Method is just a friendly name, and the command-line argument in brackets gets a friendly name too. (Also note that Cacti refers to the command-line argument as an “input source”, which isn’t terribly descriptive in the case of a bash script.) Once filled in, click Create at the bottom.

The next page (Figure 6) looks similar, but you’ll see that there are now

The screenshot shows the 'Data Input Methods [new]' form. It has three main sections: 'Name' with a text input field containing 'Weather Underground Temp Script'; 'Input Type' with a dropdown menu set to 'Script/Command'; and 'Input String' with a text area containing '/usr/local/bin/gettemp <weatherstation>'. Each section has a brief instruction on what to enter.

Figure 5. Be sure to select Script/Command!

The screenshot shows the 'Data Input Methods [edit: Weather Underground Temp Script]' form after a successful save. At the top, it says 'Save Successful.'. Below the form fields, there are two tables: 'Input Fields' and 'Output Fields'. Both tables have columns for 'Name', 'Field Order', and 'Friendly Name'. The 'Input Fields' table has an 'Add' button and currently shows 'No Input Fields'. The 'Output Fields' table has an 'Add' button and columns for 'Name', 'Field Order', 'Friendly Name', and 'Update RRA', and currently shows 'No Output Fields'. At the bottom right, there are 'Return' and 'Save' buttons.

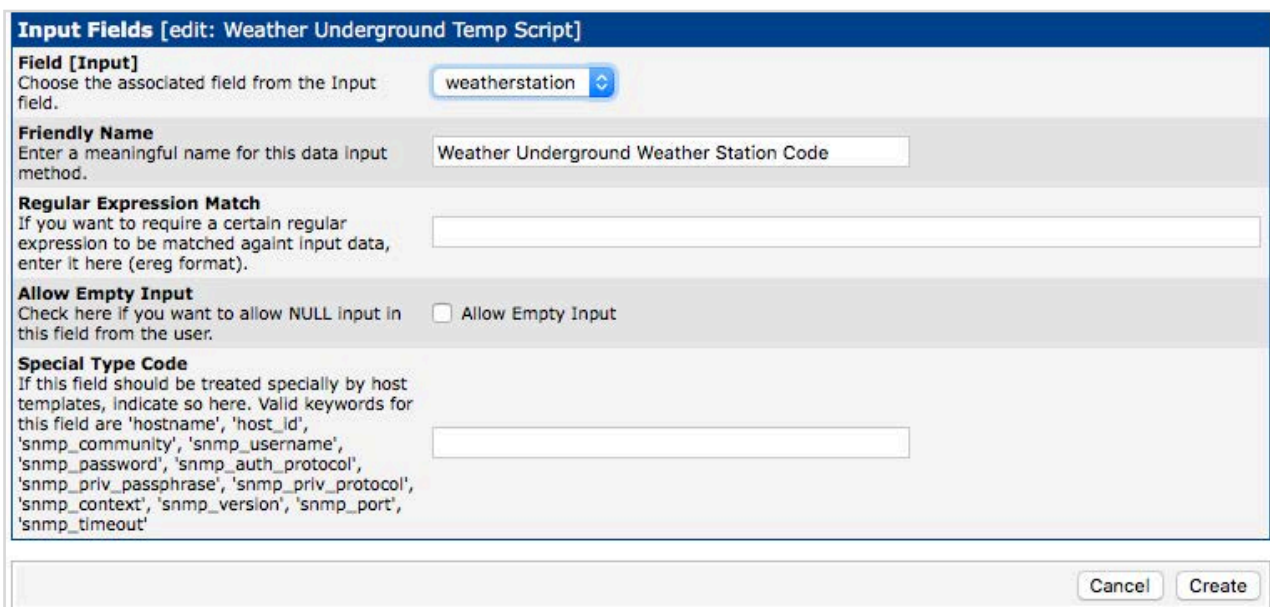
Figure 6. You need to configure an Input Field and an Output Field.

THE OPEN-SOURCE CLASSROOM

two more sections you need to complete. The first is the Input Fields section. Click Add in the upper-right corner of that section, and then tell the system what the script expects as input. In Figure 7, you can see the Input Field from the last step is already selected. My example script has only one argument, so it's the only option to choose. (Notice the name is the same that I put in <brackets> in the last step.)

I added a friendly name for the Input Field, so when I'm configuring the Data Sources later, it's clear what Cacti is looking for. If you want to get fancy, you can use regex to sanitize the input, but I'm leaving that blank. Also, since the script won't work without the weather station code, I made sure the check box to allow empty input is not checked. The last field also doesn't apply in this case, so I left it blank. Once you've chosen a friendly name, click Create.

Next, click the Add link in the upper-right corner of the "Output Fields" box. You'll be taken to a screen that looks like Figure 8. Here, you're basically applying a label to the output from your bash script. For this example, I know I'll be getting a number, so in the first field, I just called it "degrees", and then I added a friendly name that described the output. The "Update RRD File" is checked by default, and make sure to leave it checked. That's how Cacti knows this will be a graphable point of data!



The screenshot shows the 'Input Fields' configuration interface in Cacti. The title bar reads 'Input Fields [edit: Weather Underground Temp Script]'. The form contains several sections:

- Field [Input]**: A dropdown menu with 'weatherstation' selected.
- Friendly Name**: A text input field containing 'Weather Underground Weather Station Code'.
- Regular Expression Match**: An empty text input field.
- Allow Empty Input**: A checkbox labeled 'Allow Empty Input' which is unchecked.
- Special Type Code**: An empty text input field.

At the bottom right of the form are two buttons: 'Cancel' and 'Create'.

Figure 7. These error-checking features will come in handy for some scripts.

Output Fields [edit: Weather Underground Temp Script]

Field [Output]
Enter a name for this Output field.

Friendly Name
Enter a meaningful name for this data input method.

Update RRD File
Whether data from this output field is to be entered into the rrd file. Update RRD File

Figure 8. Just use a name that makes sense.

Save Successful.

Data Input Methods [edit: Weather Underground Temp Script]

Name
Enter a meaningful name for this data input method.

Input Type
Choose the method you wish to use to collect data for this Data Input method.

Input String
The data that is sent to the script, which includes the complete path to the script and input sources in <> brackets.

Input Fields			Add
Name	Field Order	Friendly Name	
weatherstation	1	Weather Underground Weather Station Code	✖

Output Fields				Add
Name	Field Order	Friendly Name	Update RRA	
degrees	0 (Not In Use)	Current Temp in Degrees Fahrenheit	Selected	✖

Figure 9. Be sure to save!

Once complete, click Create.

Your Data Input Method screen should now look like Figure 9. Be sure to press Save at the bottom right of the window; otherwise, you'll lose all of your work. If you click on "Data Input Methods" on the left column, you should see your script added as a new Script/Command. Next, you can use that Data Input Method to create your two data sources.

Part II: Data Templates

If you head over to "Data Templates" in the left column and click Add in the upper-right corner, you'll be presented with a screen that

Data Templates [new]

Name
The name given to this data template.

Data Source

Name
 Use Per-Data Source Value (Ignore this Value)

Data Input Method
This field is always templated.

Associated RRA's
This field is always templated.

Step
 Use Per-Data Source Value (Ignore this Value)

Data Source Active
 Use Per-Data Source Value (Ignore this Value) Data Source Active

Data Source Item []

Internal Data Source Name
 Use Per-Data Source Value (Ignore this Value)

Minimum Value ('U' for No Minimum)
 Use Per-Data Source Value (Ignore this Value)

Maximum Value ('U' for No Maximum)
 Use Per-Data Source Value (Ignore this Value)

Data Source Type
 Use Per-Data Source Value (Ignore this Value)

Heartbeat
 Use Per-Data Source Value (Ignore this Value)

Figure 10. Data Templates make it so much easier to create Data Sources.

looks similar to Figure 10. The idea behind a template is to answer as many questions as possible, while leaving blanks for those items that will be specific to each Data Source configured with the template. Looking at Figure 10, note that I gave the template a friendly name at the top. The next field, under Data Source, I left blank, but I checked the box that says “Use Per-Data Source Value”, because I want the two data sources to have different names. (It wouldn’t be very useful if they both had the same name.) Checking the box tells Cacti that when people use the data template, you want them to come up with their own name. The “Data Input Method” is the one you created if you were following along in Part I—just find it in the drop-down list. “Associated RRAs” tells Cacti which sets of data it should track. Be sure to select all of these, because you want to be able to create multiple graphs for historical data. Then, “Step” refers to how often it should poll the script for the temperature. The default is five minutes (300 seconds), and I recommend leaving it. Finally in that section, be

THE OPEN-SOURCE CLASSROOM

sure “Data Source Active” is checked so that it actually stores the data. Note that the last two fields in that section are *not* checked to use “Per-User Data Source”, because I want those values to be the same for any data sources created with this template.

The next section refers to the data items inside the RRD (round-robin database) file. Since I’m tracking the temperature, I just decided to call the field “temp” inside the file. I also don’t want a maximum or minimum value, so I put “U” in each of those fields. The “Data Source Type” field refers to the kind of data stored. In this case, it’s a value that fluctuates, and I want to compare the difference between them on a graph, so the “GAUGE” type is what I want. Feel free to explore

Save Successful.

Data Templates [edit: Weather Underground Temp Script]

Name
The name given to this data template.

Data Source

Name
 Use Per-Data Source Value (Ignore this Value)

Data Input Method
This field is always templated.

Associated RRA's
This field is always templated.

Step
 Use Per-Data Source Value (Ignore this Value)

Data Source Active
 Use Per-Data Source Value (Ignore this Value) Data Source Active

Data Source Item [temp] New

Internal Data Source Name
 Use Per-Data Source Value (Ignore this Value)

Minimum Value ('U' for No Minimum)
 Use Per-Data Source Value (Ignore this Value)

Maximum Value ('U' for No Maximum)
 Use Per-Data Source Value (Ignore this Value)

Data Source Type
 Use Per-Data Source Value (Ignore this Value)

Heartbeat
 Use Per-Data Source Value (Ignore this Value)

Output Field
 Use Per-Data Source Value (Ignore this Value)

Custom Data [data input: Weather Underground Temp Script]

Weather Underground Weather Station Code
 Use Per-Data Source Value (Ignore this Value)

Figure 11. Remember to fill in as much as you can, and check those boxes that should be unique to each data source.

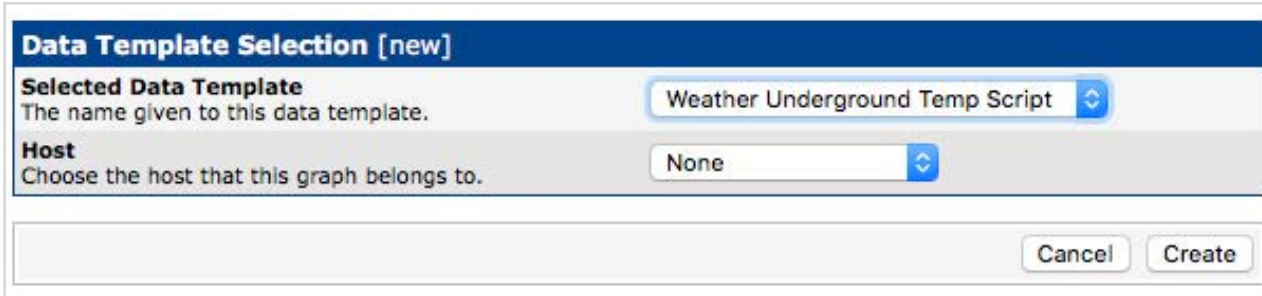
the other data types, but for monitoring and graphing something like temperature, GAUGE is what you should use. Finally, the “Heartbeat” field tells Cacti how long it can go between data entries before it needs to fill in data points with “unknown” (this defaults to ten minutes, and I suggest leaving it). Also note that none of the “Use Per-Data Source” check boxes are marked, because these values will be the same for any data source using my script. Click Create, and you should see Cacti provide another field to fill in (Figure 11).

The “Custom Data” section that appears at the bottom comes from the Data Input Method that requires an input (the weather station code). Since I want each data source to use its own code, I check the box to use “Per-Data Source” and leave the field blank. Click Save, and the data source should be configured and ready to use.

Part III: Data Sources

Now that you have the Data Input Method and a Data Template to tell Cacti how to use it, you need to create your two data sources. Head over to “Data Sources” on the left, and then click “Add” in the upper right in order to add your first source. The first page asks you to select a Data Template and Host. This data source isn’t related to a specific server, so leave Host as “none”, but you want to select your freshly created Data Template (Figure 12) and click Create.

Since the Data Template already answers most of the questions about the new data source, you’re asked only a few details. In Figure 13, you can see that I need to name this data source, choose a data source path and give it the weather station code. I just used the default Data



Data Template Selection [new]	
Selected Data Template The name given to this data template.	Weather Underground Temp Script
Host Choose the host that this graph belongs to.	None
<input type="button" value="Cancel"/> <input type="button" value="Create"/>	

Figure 12. Cool, the data template!

Creating the data sources is really the hardest part of creating graphs with Cacti.

***Turn On Data Source Debug Mode.**
***Edit Data Template.**

Data Template Selection [edit:]	
Selected Data Template The name given to this data template.	Weather Underground Temp Script
Host Choose the host that this graph belongs to.	None
Supplemental Data Template Data	
Data Source Fields	
Name Choose a name for this data source.	Petoskey Temperature
Data Source Path The full path to the RRD file.	<path_rra>/temp_14.rrd
Custom Data	
Weather Underground Weather Station Code	MAS614
<input type="button" value="Cancel"/> <input type="button" value="Save"/>	

Figure 13. See? It's much easier to create a data source when you have only two things to enter!

Source Path that Cacti created. Once you enter the name and weather station code, click Save.

If you're following along and want to do something similar to me, repeat the process for your second weather station. Since the Data Template is already in place, adding a second Data Source is very simple. I added Houston exactly the same way as I added Petoskey.

Part IV: the Graphs

Creating the Data Sources is really the hardest part of creating graphs with Cacti. Once they're created, the data will be polled every five minutes, and data will be added to the RRD files. You can't create a graph until you have a Data Source created, because graphs simply show the data inside the Data Sources graphically. So, now that you have Data

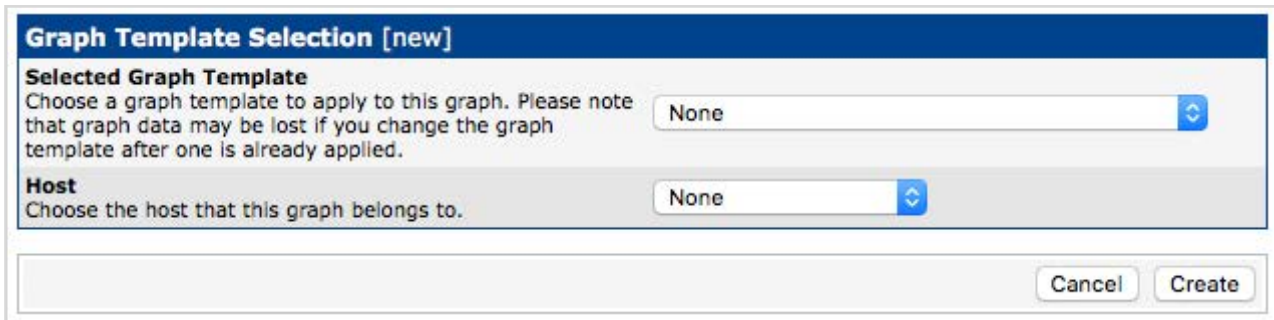


Figure 14. Graph Templates won't save you any time, but if you're doing lots of similar graphs, check them out.

Sources, you can create graphs to display them.

It's possible to set up Graph Templates, but unlike Data Templates, they're not really required. In my case, since I created only one graph, it would take longer to set up a template than simply to create a graph from scratch, so I'll just do it from scratch. In order to start, click on "Manage Graphs" on the left-hand side of the screen. (Note: don't click "New Graphs", because for some reason, Cacti will try to use

LINUX JOURNAL

on your
e-Reader

Customized
Kindle and Nook
editions
available

LEARN MORE



Graph Template Selection [new]

Selected Graph Template
Choose a graph template to apply to this graph. Please note that graph data may be lost if you change the graph template after one is already applied. None

Host
Choose the host that this graph belongs to. None

Graph Configuration

Title (--title)
The name that is printed on the graph. Petoskey & Houston

Image Format (--imgformat)
The type of graph that is generated; PNG, GIF or SVG. The selection of graph image type is very RRDtool dependent. PNG

Height (--height)
The height (in pixels) that the graph is. 120

Width (--width)
The width (in pixels) that the graph is. 500

Slope Mode (--slope-mode)
Using Slope Mode, in RRDtool 1.2.x and above, evens out the shape of the graphs at the expense of some on screen resolution. Slope Mode (--slope-mode)

Auto Scale
Auto scale the y-axis instead of defining an upper and lower limit. Note: If this is checked both the Upper and Lower limit will be ignored. Auto Scale

Auto Scale Options
Use Use --alt-autoscale (ignoring given limits)
 Use --alt-autoscale-max (accepting a lower limit)
 Use --alt-autoscale-min (accepting an upper limit, requires rrdtool 1.2.x)
 Use --alt-autoscale (accepting both limits, rrdtool default)

Logarithmic Scaling (--logarithmic)
Use Logarithmic y-axis scaling Logarithmic Scaling (--logarithmic)

SI Units for Logarithmic Scaling (--units=si)
Use SI Units for Logarithmic Scaling instead of using exponential notation (not available for rrdtool-1.0.x). Note: Linear graphs use SI notation by default. SI Units for Logarithmic Scaling (--units=si)

Rigid Boundaries Mode (--rigid)
Do not expand the lower and upper limit if the graph contains a value outside the valid range. Rigid Boundaries Mode (--rigid)

Auto Padding
Pad text so that legend and graph data always line up. Note: this could cause graphs to take longer to render because of the larger overhead. Also Auto Padding may not be accurate on all types of graphs, consistent labeling usually helps. Auto Padding

Allow Graph Export
Choose whether this graph will be included in the static html/png export if you use cacti's export feature. Allow Graph Export

Upper Limit (--upper-limit)
The maximum vertical value for the rrd graph. 100

Lower Limit (--lower-limit)
The minimum vertical value for the rrd graph. 0

Base Value (--base)
Should be set to 1024 for memory and 1000 for traffic measurements. 1000

Unit Grid Value (--unit/--y-grid)
Sets the exponent value on the Y-axis for numbers. Note: This option was added in rrdtool 1.0.36 and deprecated in 1.2.x. In RRDtool 1.2.x, this value is replaced by the --y-grid option. In this option, Y-axis grid lines appear at each grid step interval. Labels are placed every label factor lines.

Unit Exponent Value (--units-exponent)
What unit cacti should use on the Y-axis. Use 3 to display everything in 'k' or -6 to display everything in 'u' (micro).

Vertical Label (--vertical-label)
The label vertically printed to the left of the graph. Degrees Fahrenheit

Cancel Create

Figure 15. Thankfully, you want mostly all defaults.

a host and not give you the option for selecting a Data Source that doesn't have an associated host. Thankfully that isn't a problem in the "Graph Management" section.)

Once in Graph Management, click "Add" in the upper-right corner. Select "None" for both host and template (Figure 14), because you're not going to use a template for this graph. Then click Create.

The next page (Figure 15) looks overwhelming, but you're going to leave almost everything at its default. The only things I added were the Title of the graph (Petoskey & Houston) and, at the bottom, the Vertical Label (degrees Fahrenheit). For your first graph, I recommend

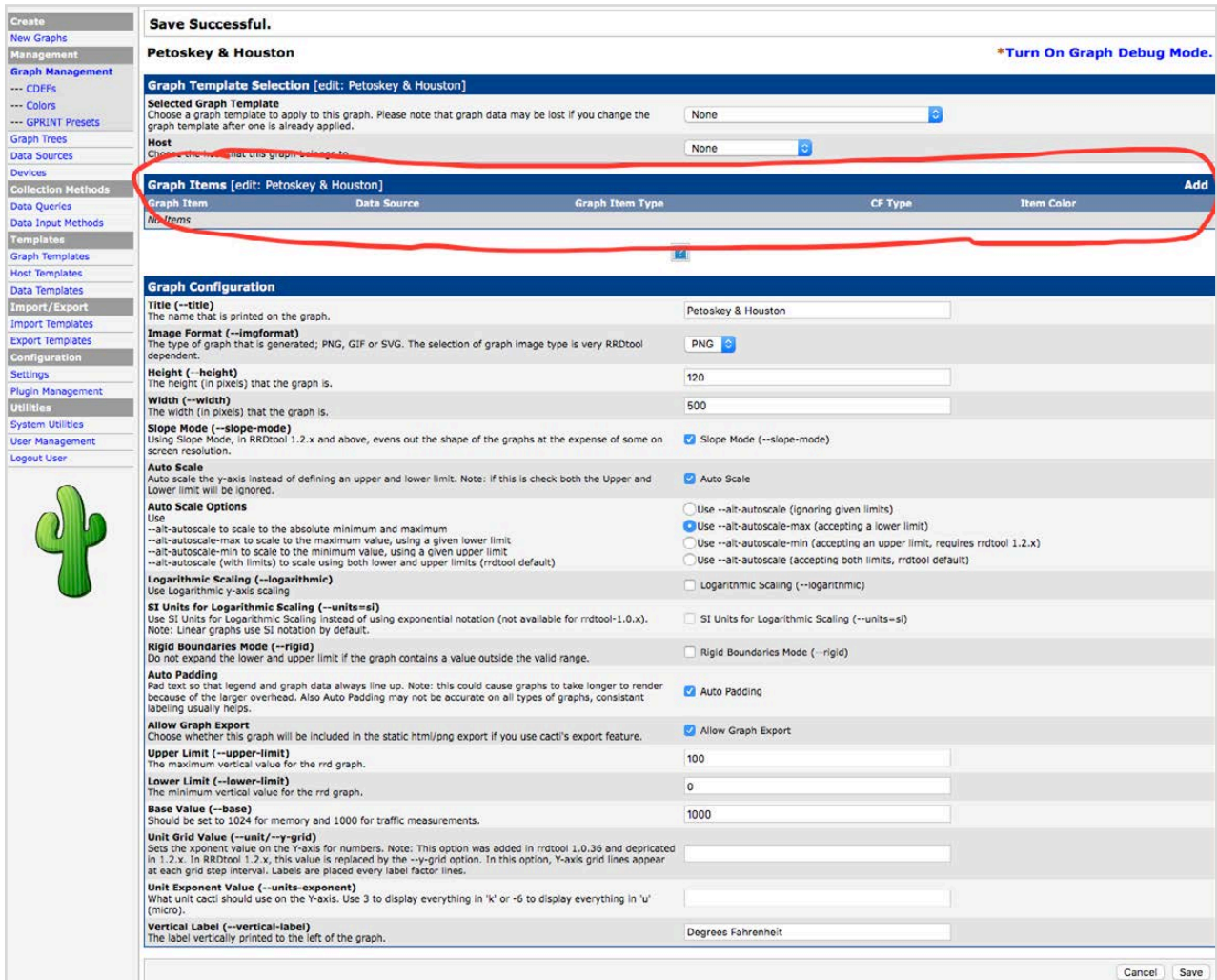


Figure 16. I missed this section at first and was confused why I didn't get any graphs.

leaving everything else at its default. Click Create.

The next page (Figure 16) looks similar, but notice the section I circled in red titled "Graph Items". You need to click "Add" on the right in that section, and add the Data Sources. You'll see I chose "Houston Temperature" as the Data Source to graph (Figure 17).

The second field is to choose a color for the graph. Irritatingly, the drop-down menu shows only HEX codes for colors, but after you select one, it displays the color for you. Houston is generally hot, so it seemed appropriate for it to be red. Next is opacity, and I chose 100%. The "Graph Item Type" does not default to "AREA", so be sure to select that for a traditional graph that looks like a rolling hill

THE OPEN-SOURCE CLASSROOM

Data Sources [host: No Host]

Host:

Data Template:

Graph Items [edit graph: Petoskey & Houston]

Data Source
The data source to use for this graph item.

Color
The color to use for the legend.

Opacity/Alpha Channel
The opacity/alpha channel of the color. Not available for rrdtool-1.0.x.

Graph Item Type
How data for this item is represented visually on the graph.

Consolidation Function
How data for this item is represented statistically on the graph.

CDEF Function
A CDEF (math) function to apply to this item on the graph.

Value
The value of an HRULE or VRULE graph item.

GPRINT Type
If this graph item is a GPRINT, you can optionally choose another format here. You can define additional types under "GPRINT Presets".

Text Format
Text that will be displayed on the legend for this graph item.

Insert Hard Return
Forces the legend to the next line after this item.

Sequence

Figure 17. Why on earth does the drop-down box show only HEX?

Graph Items [edit graph: Petoskey & Houston]

Data Source
The data source to use for this graph item.

Color
The color to use for the legend.

Opacity/Alpha Channel
The opacity/alpha channel of the color. Not available for rrdtool-1.0.x.

Graph Item Type
How data for this item is represented visually on the graph.

Consolidation Function
How data for this item is represented statistically on the graph.

CDEF Function
A CDEF (math) function to apply to this item on the graph.

Value
The value of an HRULE or VRULE graph item.

GPRINT Type
If this graph item is a GPRINT, you can optionally choose another format here. You can define additional types under "GPRINT Presets".

Text Format
Text that will be displayed on the legend for this graph item.

Insert Hard Return
Forces the legend to the next line after this item.

Sequence

Figure 18. Blue seemed appropriate for Petoskey.

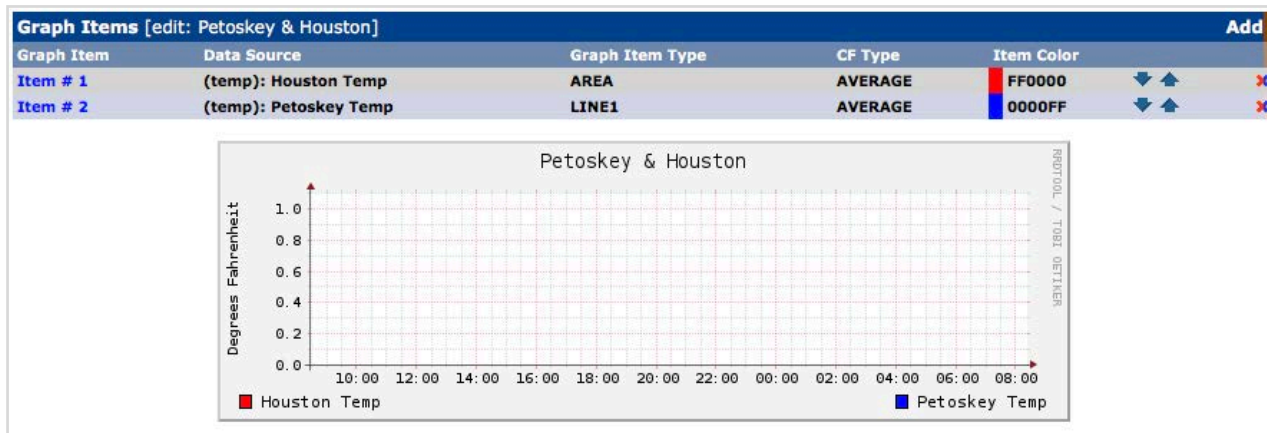


Figure 19. Looks like the graph is configured. (Don't forget to click Save.)

of data. It's safe to leave "Consolidation Function" to AVERAGE, and everything else the default. I did add "Houston Temp" to the Text Format field, so it shows which color is which Data Source on my graph. Once you're happy with the data, click Create. Do the same thing with your second Data Source. You'll notice in mine (Figure 18) that I chose blue for Petoskey, and instead of AREA, I chose LINE1. It's a different type of graph, so that instead of filling the page, it will draw a line with the temperature. I did that so Houston would fill the background, and Petoskey would draw a line over the top of it, so you can see both. Figure 19 shows the preview of what the graph will look like. Be sure to click Save at the bottom!

That's It!

The only thing left to do is wait. Every five minutes, there should be a data point added, and the graphs will update with a graphical representation of that data. Cacti allows you to export the graphs to a local path (like /var/www/html) or to a remote FTP server. If you don't have it export the graphs, you'll have to log in to see them. I won't go through the process for exporting the graphs, but click on the "settings" link on the left column, and then the "Graph Export" tab across the top. The setup is fairly self-explanatory. To see the graphs you've just created, wait 10–15 minutes, then follow the arrows as shown in Figure 20 to find the graphs. You should see your new graphs starting to populate themselves with data!

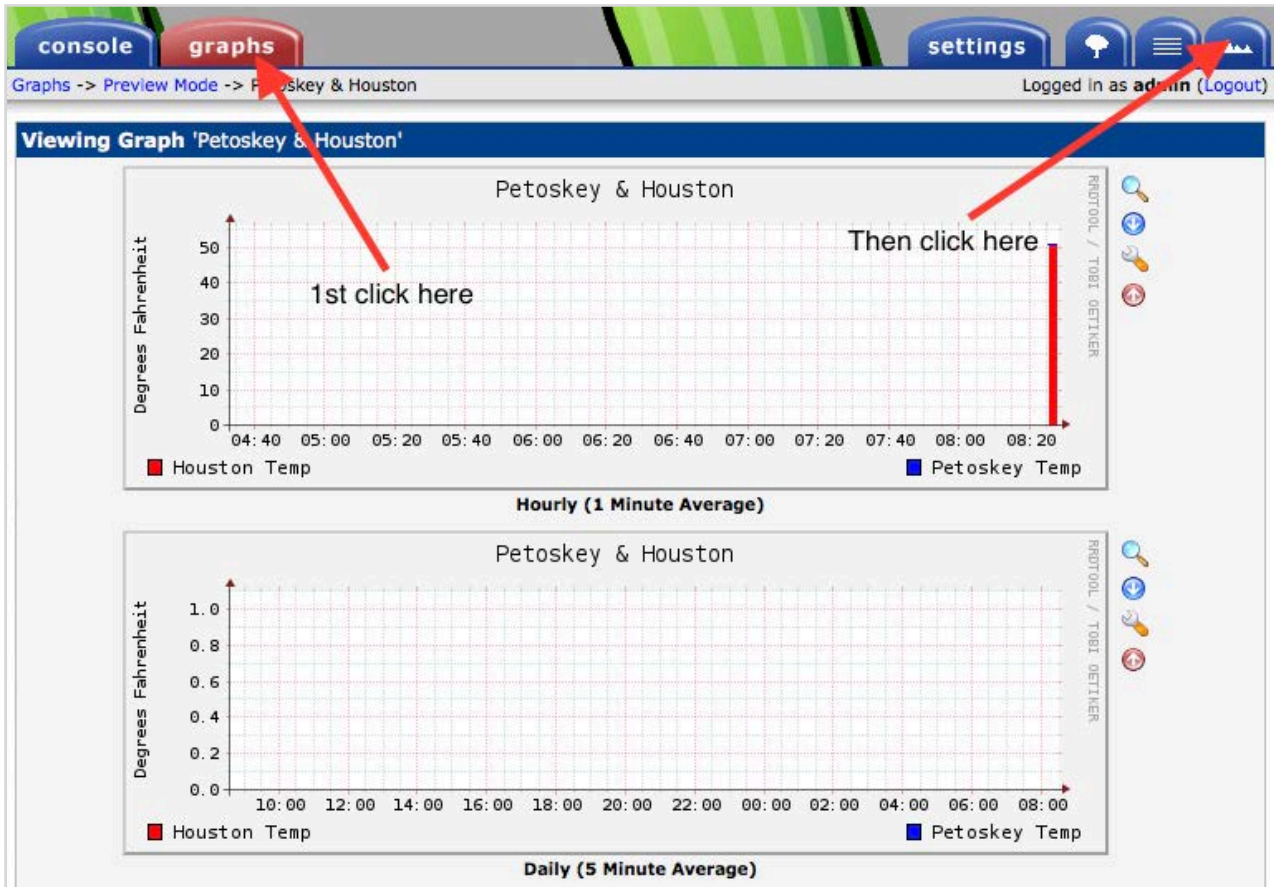


Figure 20. These are frustrating to find; hopefully the arrows help you.

Cacti has lots of other features, and it allows you to customize your graphs with min/max values displayed on the graphic, along with different types of graphs, data sources and so on. Once you become familiar with using it, Cacti is a very nice tool for automating the graphing process. I hope you have as much fun with it as I did! ■

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

[RETURN TO CONTENTS](#)

LINUX JOURNAL

on your
Android device

Download the
app now from
the **Google
Play Store.**



www.linuxjournal.com/android

For more information about advertising opportunities within *Linux Journal* iPhone, iPad and Android apps, contact John Grogan at +1-713-344-1956 x2 or ads@linuxjournal.com.

Holy Triage, Batman!

Code triage: it's a dirty job, but somebody's got to do it—quickly, well and often on very little notice.

PREVIOUS

◀ Shawn Powers' The Open-Source Classroom

NEXT ▶
New Products

ALMOST ONCE PER WEEK, SOMEONE COMES TO ME ASKING FOR A CODE AUDIT. Almost invariably, these people have no idea what they are asking for. In 99% of cases, what they really need is *code triage*, or perhaps a more in-depth review, not an audit.

A real code audit means *auditing to a standard*, a process in which code is checked to see that it complies fully with a specific, objective coding standard. It can be done, but it's resource-intensive for all but the most trivial of cases. Unless the standard is unusually well suited to ensuring something very much needed by the code's use case, this type of audit is usually more socially useful than it is technologically useful.

More commonly, what people requesting a code audit mean is *make me a one-page list of every way in which this software deviates from the ideal*. There are



SUSAN SONS

Susan Sons serves as a Senior Systems Analyst at Indiana University's Center for Applied Cybersecurity Research (<http://cacr.iu.edu>), where she divides her time between helping NSF-funded science and infrastructure projects improve their security, helping secure a DHS-funded static analysis project, and various attempts to save the world from poor information security practices in general. Susan also volunteers as Director of the Internet Civil Engineering Institute (<http://icei.org>), a nonprofit dedicated to supporting and securing the common software infrastructure on which we all depend. In her free time, she raises an amazing mini-hacker, writes, codes, researches, practices martial arts, lifts heavy things and volunteers as a search-and-rescue and disaster relief worker.

a number of things wrong with this:

1. You probably can't provide a specific and extensive enough definition of correct behavior for your software, taking into account all possible inputs, environments and eventualities, to provide a standard to evaluate its total correctness. To audit is to *compare code to an objective standard*, without such a standard, audit is impossible.
2. I cannot find *every* problem in your code while you are still developing it, and you probably aren't willing to undergo a one- to two-year code freeze while your code is analyzed.
3. You probably aren't willing to pay for one to four man-years of senior software security engineer time to come close to finding *every* potential problem with your code, which is typical for a medium-complexity project with mediocre code quality and a few hundred kloc.

I do sometimes say yes to *code reviews*, and I often find myself doing *code triage*. What's the difference?

Code review is systematic examination of computer source code intended to find mistakes overlooked in the initial development phase, improving the overall quality of software.

Code triage is a specific form of code review intended to identify the most critical targets for immediate improvement without a deep inspection. In other words, code triage answers the question, "Given limited time and resources, how do I address this code's worst deficiencies?"

The Role of Triage

One of the scariest moments in a developer's life—especially someone who works with infrastructure software or anything security-critical—is taking over someone else's mess. Even someone else's good code—in enough volume, on tight enough deadlines, with little enough documentation, tooling and familiarity—can seem like a mess. Code triage is the method for making sense of the mess, instead of saying "I can't touch this until I've had 12 solid man-months to spelunk its depths."

Let's assume you don't have 12 man-months.

Let's pretend you just discovered that a critical piece of infrastructure software has gone unmaintained, or badly mismanaged, for about a decade. You don't know how bad it is, but you suspect it is terrible. It's the kind of software that, if it breaks just so, people will die, or the world economy will be in ruins—whatever. This has somehow (through assignment or your unrelenting sense of duty) become your problem.

This may or may not have happened to me before. I can promise you, there's a way to win through, and once you've become practiced at it, having a few hundred-thousand lines of broken mystery code fall in your lap will become significantly less daunting. Note that I didn't say "sane" or "easy"—just "less daunting". There is method to the madness.

The People Phase

Never start with the code if you can help it. The code will tell you what someone programmed, but it never will tell you what someone intended to program or, for that matter, what they *should have* intended to program. I begin code triage on complex projects (anything that seems to deserve more than a half-day of my time) with a fountain pen and a notebook. Vary the tools if you must, but the process is important.

Keep excellent notes. Remember that triage is a process of gathering information to use in a decision-making process. If information is lost before it can be used, you've failed. Triage is often undertaken as a one-person activity, but it may be done by a small team if the team is tightly knit and well coordinated enough. If working with a team, notes must be kept in such a way that team members can collaborate in as close to real time as possible, and that the notes are clear to all of the team at all times. Getting to that point with a team is difficult but possible.

Your complete notes should be close at hand at all times during the triage process. Often, information you come across in one part of the process will conflict with, or relate to, something that came up earlier. These relationships are typically the most important discoveries and may escape notice if notes are disjointed or compiled only after the fact. Keep it together! Review the entire body of notes often.

Use varied sources. You will want to contact as many different stakeholders as possible, including at least one sample from any distinct

What problems have others identified already? What barriers have others found when they have tried to make improvements? What frustrates people? What changes are people afraid of? What functionality do they depend on?

group of stakeholders you can identify. For example, when triaging the NTP reference implementation, I spoke to the project's maintainer, to the project's funding coordinator, to community members who had contributed code or tried and failed to contribute code, to past contributors who had left the project, to a package maintainer who packaged the software for a major Linux distribution, and to end users in different sectors where the software was employed: commodity computer usage, data center, core internet routing, scientific applications and finance sector. I did not have the opportunity to talk to the software's original author, but I made an effort to get to know a bit about him through his writing.

Listen mindfully. When interviewing people as part of software triage, expect to get different perspectives from different people. Don't try to get consensus; it's not important at this stage. You are gathering information and not much else. What problems have others identified already? What barriers have others found when they have tried to make improvements? What frustrates people? What changes are people afraid of? What functionality do they depend on?

I tend to ask users most about their use cases. I ask developers most about developer experience, the purpose of the software, and so on. And, I ask everyone about the things they would like to change and not like to change. However, the most important thing that I look for is something that, by nature, I cannot ask:

What assumptions do I hear when people talk about the software?

Unspoken assumptions can be dangerous, and until I've gone through the documentation in detail, I do not know which assumptions are or are not explicitly documented. So, I assume all assumptions have been left

unspoken until proven otherwise. Document assumptions regardless of their correctness. Spotting them takes practice.

“What time would you like to go to dinner?” assumes that you would like to go to dinner, and that you have an opinion about what time you would like to go.

“I set my pencil on the desk a moment ago; if it isn’t still there, it must have rolled off” assumes that no one has picked up the pencil, that the pencil isn’t capable of walking or flying, that the pencil is capable of rolling, that pencils don’t evaporate, and that nothing on your desk ate the pencil.

Try This Exercise: Someone says, “This program tells you how many files are in your home directory.” What are some questions you could ask about such a simple program to root out unspoken assumptions about the program’s expected behavior? See the sidebar at the end of this article for a list of possible questions.

Put others at ease. This is the most difficult part, especially for many software engineers for whom interviewing is not part of their core skill set. If you are new at this or unsure of yourself, start with the consumers of the software: they are the easiest, because you often can deflect from issues of the software itself by focusing on their workflow and use case, and they usually don’t see themselves as responsible for the current state of the software. If they aren’t very technical, get the workflow and use-case information from them directly, then see if one of the consumers can get you in touch with someone in their IT department who supports the software for them: that’s an important stakeholder too, with potentially crucial information on factors such as operating environment.

I tend to start with the current or former developer(s) of a project I’m trying to triage. Regardless of whether I’m triaging in order to take over or triaging in order to assist the standing team, these will be the most delicate interviews. They also are potentially the most fruitful. Who better knows the assumptions with which the code has been developed so far than the people developing it? Who knows how it got to where it is today? Who knows what users or potential users, contributors or potential contributors ask the development team the most? Who knows where the tooling is falling down or a struggle to work with? Who has the most ego involved with the current state of

the software? Yep. Be prepared to tread lightly.

First and foremost: *do not take an adversarial mindset into any interview*, no matter how badly you think anyone may have done his or her job. It will come through in your speech and behavior, and it will make people shut down on you in self defense. Your job is to fix the software, and you will be most effective at that if you can find some empathy for the people with whom you are speaking. This is another reason I start with the people before the code. I want to approach those people with empathy and understanding, but I'm still a cynical, grumpy engineer—at least on the days when I've been slogging through 200,000 lines of code trying to find the race condition that ate Manhattan. Don't meet people the day you wrestled with their bad code.

I keep interviews about software triage as informal as possible. Formality causes most people to expect an adversarial experience, which is exactly what I do not want to provide. I want to make people feel comfortable. Usually, this means dropping a short, informal email to set up a time convenient to them and then doing a face-to-face meeting (if possible) over tea or coffee, or a chat by phone or video conference. If face to face, make the effort to incorporate snacks/beverages: eating and drinking is a natural signal to the body that we are not in combat, and it may have a calming effect.

Bad code is bad for a reason. Let the developers tell their side of the story. Sometimes it's self-inflicted, sometimes it's not. You can judge later. Your job during the interview is to be empathetic and let them talk. Nobody gets up in the morning and decides to write terrible code just for fun. It tends to result from developers working in a toxic environment, or being in over their heads, or lacking resources, or being burnt out or some horrible dysfunction—something went wrong. Chances are, the developers can't or won't identify this and tell you directly, but if you get them talking about the software long enough, let them ramble a bit, and ask pointed questions here and there in an empathetic manner, you will get it eventually. What is most important will vary a great deal from case to case, so this stage takes patience; follow threads and see where they go.

Assume that people do not want their comments and reflections attributed to them unless they have specifically given you permission to

UNDER THE SINK

quote them. Fears of causing drama or feeding a rumor mill will cause people to self-censor.

In one case, I found that the most relevant tale was when the software lost a major funding source and the team ended up adding features that didn't fit their vision for the project in order to keep funding flowing from other sources, rather than let the software die. This was very helpful information, because it told me I needed to understand the project's current funding strategies when I made triage recommendations.

In another, the project lead retired and hadn't planned well enough for the project's longevity. Not having another plan, he handed the software off to his former assistant who wasn't ready for the responsibility. Years of being in over his head made the assistant a paranoid and dysfunctional project lead who chased off developers and drove the software into crisis. I burned a great deal of time and energy trying to get this project lead to cooperate with saving his software and never did succeed. I don't regret trying. Over many long, late-night phone calls, I got an inside view of how he'd struggled to balance the interests of those he saw as his most important stakeholders. Most of his views on managing the code were off enough not to be very helpful, but coming to understand how he'd gone about interacting with people made a big difference, even when he hadn't been interacting effectively.

In yet another case, the software had been written by company A to manage a specific hardware platform purchased by several companies, including B. B became so dependent on the software that when A stopped maintaining the software, B made a deal to buy it. B, however, wasn't a software development firm and had no in-house resources for software development, so it hired a series of contractors for one-off improvements or feature-adds to the software. After eight years and more than 20 contractors, the software was a security and reliability nightmare with no design integrity whatsoever, no documentation and a brittle build system. When I got to the code, I was prepared to deal with the huge variation in coding styles I found and the lack of design integrity. I also was able to get in touch with someone in the accounting department who could help me reconstruct which contractors I would want to speak with based on dates of various code changes, and I actually was able reach some of them.

Reliable, reproducible builds are necessary to the development process, and little improvement to the code can happen without them.

Don't be afraid to go back. Often, after talking to more people or during a later stage of the triage process, you will trip on something that leaves you wanting to speak with someone you've already spoken to. I end all interviews by asking permission to contact the people again should such a need arise and ask how they prefer to be contacted so that I may be as respectful of their time as possible.

The Proxy Phase

Before I jump headlong into a big code base, but after I've spoken to whatever relevant people I can reach, I still have work to do around the code. I spend some time looking for things that can give me red flags of likely problems, or signs that certain other things may be well handled, or tools that may exist to make my work with the code easier. I call this the *proxy stage* because many of the things I look at aren't actually direct evidence of what's broken; they're just strongly correlated enough to be useful in a quick triage.

The proxy stage will direct the effort you put in for the rest of your triage process. It's triage for triage. When you find out that the software cannot be built without the one machine in a former developer's apartment that no one has root on, which has a black-box script no one knows the contents of, the build system becomes a priority far higher than the contents of the code base. Reliable, reproducible builds are necessary to the development process, and little improvement to the code can happen without them.

On the other hand, discovering that the source control, build system and so on are in good shape tells you that spending time digging through SCM logs probably will give you useful information, because someone took the time to use those tools properly.

Begin with documentation. Hopefully you already know how useful good documentation can be; however, don't discount the potential

treasures to be found in bad documentation:

- Who authored what parts of the documentation can tell you about who cares about what components, other than the developers.
- Relative ages of different parts of the documentation can give you an idea of the relative neglect of different parts of the code, in the absence of revision control or other, better data.
- Insight into the mental models that developers were operating on while writing code.

Spelunk issue queue contents (current and historical).

- Find out how issues have been handled in the past; this will tell you a lot about the development team's workflow.
- Find out what big issues have been churned on for a long time but not solved.
- Find out what security issues have cropped up in the past and how they were dealt with.
- Find out how active the community/team is in general.

Look at tests.

- Are there tests?
- What is the coverage like?
- What is the overall sophistication like? For example, is it a unit-test-only setup, a functional-test-only setup, or are both in use? Has this project begun using fuzz testing? Is there scaffolding for mocked interfaces?
- Do all tests currently run and pass?

UNDER THE SINK

- Can you tell anything about the testing strategies in use? Were tests committed with every patch? Was adversarial testing employed? Was only one person writing tests? Was anyone a testing specialist?

Examine tooling (build system, CI infrastructure, source control setup and so on).

- How much automation is/was in place?
- How reliable is the automation? How much is still available/usable?
- Is a modern SCM (git or Mercurial) in use?
- Do the tools seem to be reducing the dev team's overhead or increasing it? (That is, is it doing its job, which is making developers' lives easier and their work better?)

Use commit messages, tags and branch structure within the SCM.

- Are commit messages, tags and branching used effectively—that is, can you follow them?
- What can you learn from reading the commit messages?

Look at general style and code quality.

- Don't get sucked into a deep read of the code yet, skim only.
- How is the overall comment density? Are the comments literate?
- Does the indentation, overall structure and so on suggest the absence or presence of a style guide?
- Does this feel "clean" or "messy" in general?
- Is semantic versioning used?

- How many red flag comments do you see in a quick skim? Look for things that indicate cut-and-paste coding—for example, “got this from <url>” or anything mentioning Stack Overflow. Also, look for “WTF”, “IDK why, but if I remove this, it breaks”, and things like that. These are areas you may want to look at in the code stage if you have time.

The Code Phase

You may have run out of time for triage by now, or you may have found so many problems that triage is done. I have had projects like that. When I stepped into NTP, the code wasn't C99-compliant, the build system was unusable, the code was locked up in an inaccessible and antiquated SCM, and the documentation was mostly more than seven years out of date—all of those issues took precedence over specific code improvements, because fixing them was a prerequisite to enabling developers to fix the code. We needed to be able to onboard new developers by giving them access to code they could actually build and documentation on how it all worked.

Do not try to read, let alone understand, all or most of the code. Your job is not to find every problem—90% of problems are irrelevant to your search, unless the code is shockingly well written. Remember, you are doing triage: you are a field medic, you are not performing an autopsy. You are figuring out how to do the most to improve the life of the patient in limited time with limited resources: where do I get the most bang for my buck *right now*, and what do I look at next once that is done? Nothing more. Your proxy stage, above, will give you a clue about how much of the code stage to bother with. In most cases, you will skip some or most of it.

You will get your biggest gains by improving development process (because then fixing code becomes faster/easier, and all development after that point gives greater returns), by fixing extremely high-impact vulnerabilities and by making changes that remove entire classes of vulnerabilities (rather than trying to squash them one at a time). To that end, see below.

Evaluate program architecture. Think about the code's overall architecture. Are you having trouble navigating it? How good or bad is

the separation of concerns? How well is minimization being practiced? Don't spend too much time trying to learn it all, just skim for major red flags—for example, crypto algorithms housed in the same place as web interface code or piles of theoretically unreachable code.

Eliminating code eliminates attack surface, eliminating entire classes of vulnerabilities. It also reduces complexity, reducing opportunities for developer mistakes. If you can safely remove code, do so.

Refactoring to make code more navigable and more logically compartmentalized makes it easier for developers to understand, makes bug fixing easier, reduces the rate of defects introduced by developer error and increases the speed at which developers can introduce high-quality, atomic patches. It isn't always highest priority in a disastrous code base though, as brittle code bases are difficult and time-consuming to refactor. Other changes likely will take priority if the code resembles spaghetti.

Catalog interfaces. Find and list all of the software's internal and external interfaces—or all of the ones you can find. Try to figure out which ones are well defined and controlled, and which aren't, and figure out which are used and necessary, and which aren't.

Catalog data stores and data sources. Most software deals with data at some point. Look at where external data comes from, what assumptions are made and how the software copes with nonconforming (accidentally or maliciously) or missing data. Now do it again for any data the software stores.

Remain mindful of assumptions. As you go through the code, keep in mind all the assumptions you noted earlier. Note anything in the code that confirms or conflicts with those assumptions. Note any new assumptions you find.

Putting It All Together

Don't go down rabbit holes. A good software engineer will be tempted, at some point, to dive in to an interesting problem. Six hours later, your triage will be shot. You *do not* have time to understand any one problem fully; you are trying to understand the breadth of the problems the software has. This is not an exercise in depth. Don't be afraid to make generalizations and intuitive leaps, as long as you note

them as such and jot down a rough estimate of the time it would take to investigate fully the issues involved.

Triage is more complicated when you do not understand the problem space, but that complication largely can be conquered by carefully compartmentalizing the problem. Be methodical, and don't get distracted by the esoteric bits of domain knowledge you don't have. If the software is not properly segmented so that very domain-specific algorithms are separate from interfaces, crypto and so on, that is a fault in itself. Note it, and move on. If it is well segmented, you should have no problem checking out the build system, data stores, interfaces and so on, leaving the domain-specific code segments for deep dives with a domain expert by your side.

The first thing you will want to do is to use the information you just gathered to aid in decision-making and communicate that process to others. Lay out a plan, and describe what led you to choose that plan. Keep your notes, and ensure that the references to specific code in those notes will be find-able later, after the code has evolved and/or been moved to another SCM.

If you can, follow the software through at least the first stages of its refactor or rejuvenation following the triage you just did. That experience provides a crucial feedback loop that will enable you to improve your triage skills much faster than you could without it. You'll inevitably see things you missed: some that you had no chance of finding without a deep dive, and some that you'll soon realize were staring you in the face all along. The more of these experiences you have, the better you will become at triage.

Practice is *the* way to improve your code triage skills. Good practice is frequent and in volume. Long breaks make it hard to build on and reinforce previous learning. Working with only small code samples will not teach you the skills needed to find big-picture problems and trends in a sea that you cannot read line by line. Additionally, try to work with a variety of code, in terms of language, domain and quality. If you can, also stretch your assessment muscles in other domains. I've learned many triage skills in volunteer search-and-rescue work that transferred to software engineering and information security. ■

SOME ANSWERS TO THE "TRY THIS EXERCISE"

- How does it determine the scope of "my home directory"?
- Will it span multiple devices?
- Will it follow symlinks?
- If the same file is symlinked or hardlinked many times, how will it be counted?
- What if there is a hardlink or symlink in my home directory to something outside my home directory?
- Is any type of deduplication attempted?
- What if I, for some reason, had filesystem objects in my home directory that aren't really files per se, such as broken filesystem pointers or half of /proc?
- How does the program determine what my home directory is?
- If \$HOME differs from what is listed as my home in /etc/passwd will that have any effect?

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

[RETURN TO CONTENTS](#)

NEW PRODUCTS

◀ PREVIOUS Susan Sons' Under the Sink	NEXT Feature: Low Power Wireless ▶
---	--

Synacor, Inc.'s Zimbra Open Source Support and Zimbra Suite Plus



Zimbra Collaboration Suite is a successful open-source collaboration application that includes email, calendaring, file sharing, chat and video chat. Zimbra's developer, Synacor, Inc., recently released two new Zimbra-related offerings, namely Zimbra Open Source Support (ZOSS) and Zimbra Suite Plus. The first offering, ZOSS, is a new global program for the 400+ million Zimbra users that takes Zimbra Open Source Edition deployments to the business-ready level. Private and secure, ZOSS offers support to users worldwide in their local language, during their business hours and by experts who understand their business needs and culture. The second offering, Zimbra Suite Plus, has been upgraded with a palette of new features. This modular add-on that extends Zimbra Server's capabilities on Zimbra Open Source or Network Edition now offers new and innovative tools, such as Zimbra Backup Plus, Admin Plus, HSM Plus and Mobile Plus.

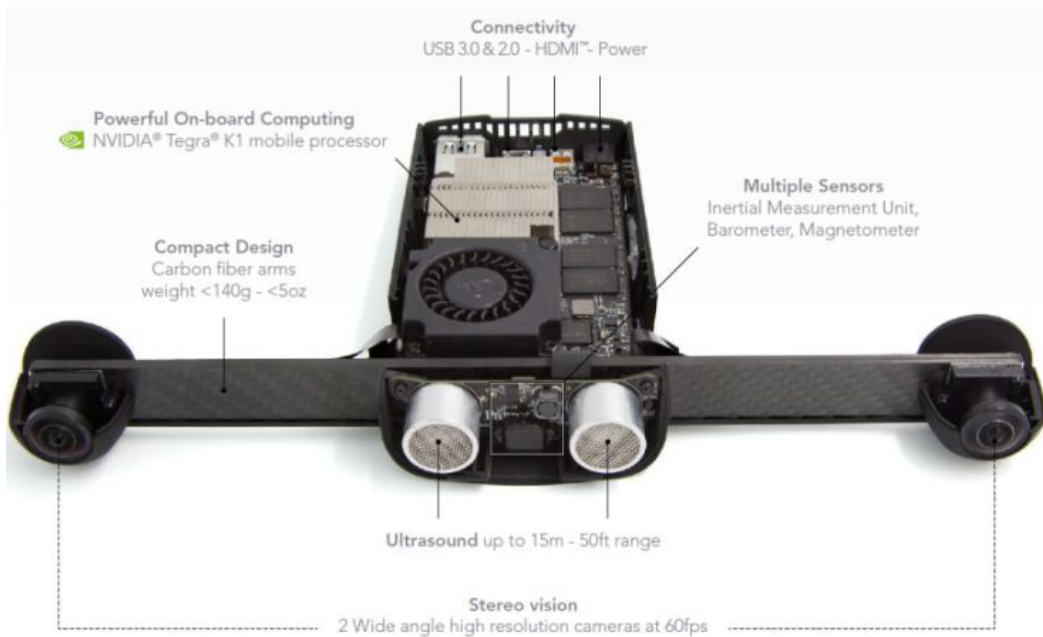
<http://synacor.com> and <http://zimbra.com>



Equus Computer Systems, Inc.'s 1.5U Server

IT operations seeking to optimize valuable data-center rack space while improving efficiency are the target customers sought by Equus Computer Systems, Inc., for its "unique" new 1.5U server with data transfer rates of 12GiB/s. With two drive drawers each supporting eight 3.5" hot-swappable SAS/SATA hard drives and delivering up to 160TB of drive storage in a compact form factor, Equus' server also supports two internal 2.5" SSD drives, meaning it can scale out to 16 data drives while still providing a separate redundant OS volume. Equus observes that the 1.5U server can accommodate 33% more drives than a typical 2U server while using 25% less rack space. The availability of two drive drawers means drive failure will not bring down the server. Users simply can pull out the drive drawer and remove the failed drive while the rest of the drives in the drawer remain online. The flexible 1.5U server is a great fit for academia, medical, enterprise or content delivery applications, and it functions well for object-oriented storage or as database, file or analytic servers.

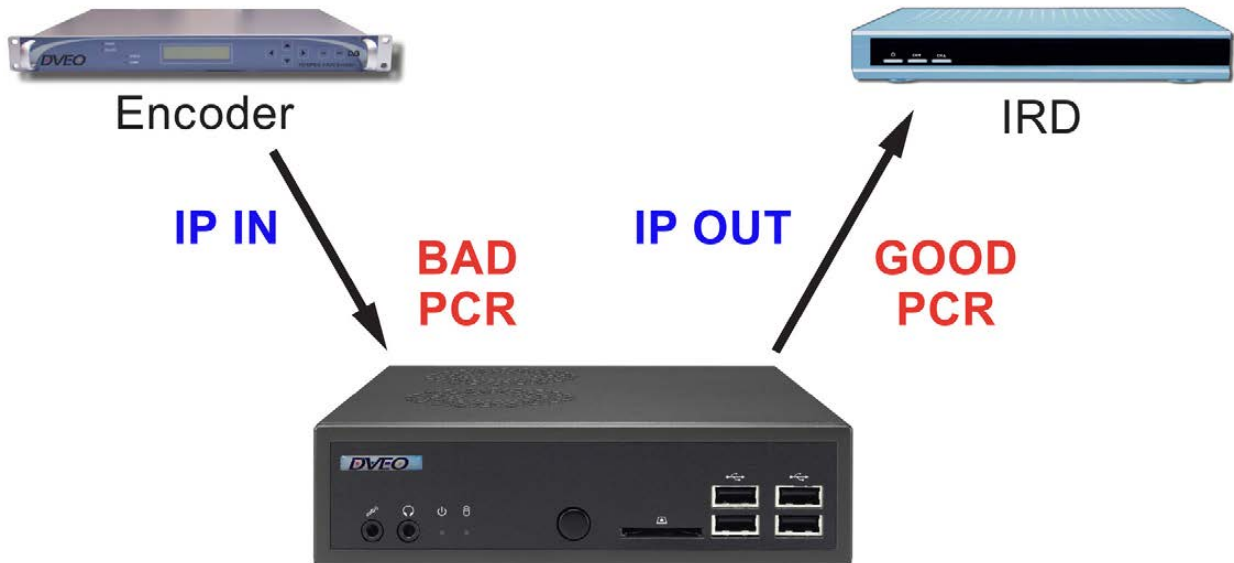
<http://equuscs.com>



Parrot S.L.A.M.dunk

Powered by Ubuntu and Robot Operating System (ROS), the Parrot S.L.A.M.dunk open development hardware and software kit enables drones to transform into smart robots. Parrot's S.L.A.M.dunk (Simultaneous Localization and Mapping) technology enables the design of advanced applications that enable the drone to understand and map its surroundings in 3D and self-navigate in environments with multiple barriers or lack of a GPS signal. Because Parrot S.L.A.M.dunk exploits ROS, the leading Linux-based versatile robotic development environment, it can be used not only for standard drone applications but also for a much wider set of "robots"—that is, flying wings, articulated arms and roving robots. Parrot S.L.A.M.dunk applications include autonomous navigation for drones and robots, 3D mapping, obstacle avoidance, prototyping of drones and robotic solutions and data gathering via the stereo camera and sensor array (IMU, barometer, magnetometer, ultrasound). Parrot asserts that it developed Parrot S.L.A.M.dunk to be as easy and user-friendly as possible for developers, researchers, integrators and academics.

<http://parrot.com>



DVEO's Jitter Box IP/IP

Telco TV/OTT and IPTV operators must deal with the fact that many IP transport streams are asynchronous. This makes the streams prone to poor video quality due to jitter if they are sent to Program Clock Reference (PCR)-compliant devices. A new corrective solution for this challenge is DVEO's Jitter Box IP/IP, a low-power, Linux-based IP PCR jitter-correction appliance. Designed for telco TV/OTT and IPTV operators, the Jitter Box IP/IP corrects the PCR in IP video transport streams so IP streams can be jitter-free and interoperate with PCR-sensitive devices, such as modulators, muxers, decoders and encapsulators. If an IP stream is not compliant, the Jitter Box IP/IP can make it so. Jitter Box IP/IP also features a web-based GUI that is manageable from anywhere, adds DVEO.

<http://dveo.com>



*FutureVault*TM
Pioneering the Digital Collaborative Vault

FutureVault Inc.'s FutureVault

Though short of Mr Torvalds' aim of world domination, FutureVault, Inc., has set the ambitious goal to "change the way business is done" with its FutureVault digital collaborative vault application. Described by its developer as "at the epicenter of a brand new disruptive category in the financial services world", FutureVault allows users to deposit, store and manage important financial, legal and personal documents digitally by means of a white-label, cloud-based, SaaS platform. FutureVault is a solution for banks and other financial institutions used to acquire, retain and reward their customers while yielding valuable data and analytics. With robust features around secure sharing, collaboration and organization, FutureVault, Inc., calls its solution "an incredibly 'sticky' tool to connect and stay connected with clients in order to increase both trust and opportunity between all parties". The company predicts that "Within 15 years, it will be unheard of not to have all of one's personal, financial and legal documents deposited in a product like FutureVault."

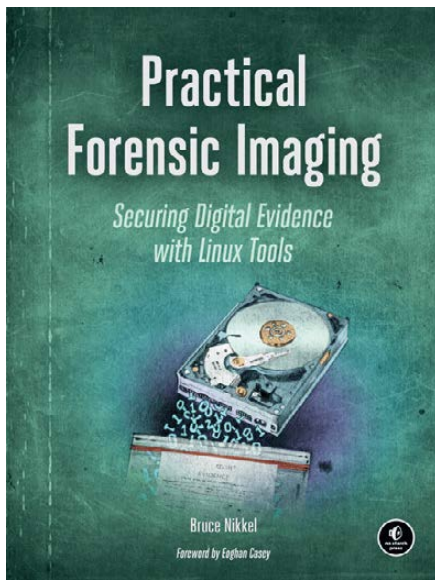
<http://futurevault.com>



CORSAIR's Carbide Air 740

Part of the joy of owning a custom-built PC is building it yourself, notes CORSAIR. (Oh, do we agree!) In an effort to promote endless PC-making joy, CORSAIR developed the new Carbide Air 740 PC case that “offers remarkable cooling performance and the flexibility to handle even the most ambitious enthusiast system builds”. The Carbide Air 740 features CORSAIR’s innovative dual-chamber Direct Airflow Path design, innovative cooling capabilities, unique bold design and a stunningly tinted, fully removable side-windowed panel. An evolution of the well received CORSAIR Carbide Air 540, the Carbide Air 740’s Direct Airflow Path design fully utilizes the space within the case by splitting the system’s hardware between two compartments. The main chamber houses the core heat generating system components—motherboard, CPU, graphics cards and memory—while the rear chamber mounts the 3.5in/2.5in drive bays and the PSU. This arrangement, claims CORSAIR, maximizes the airflow from the three pre-installed 140mm fans, delivering unparalleled, unimpeded airflow through the main compartment. Drive cages and cables are neatly tucked away and don’t interfere with airflow, all without compromising storage capacity. To maximize the coolness of running so cool, the Carbide Air 740’s enormous, hinged and fully removable side panel is perfect for showing off those amazing builds and setups.

<http://corsair.com>



Bruce Nikkel's *Practical Forensic Imaging* (No Starch Press)

Forensic image acquisition is an important part of the process of after-the-fact incident response and evidence collection. Digital forensic investigators acquire, preserve and manage digital evidence

as part of criminal and civil court cases; they examine violations of organizational policy; and they analyze cyber attacks. Author Bruce Nikkel, in his new book *Practical Forensic Imaging*, takes an in-depth look into how to secure and manage digital evidence using Linux command-line tools. This essential guide walks readers through the entire forensic acquisition process and covers a wide range of practical scenarios and situations related to the imaging of storage media. Readers learn how to perform critical tasks, such as performing forensic imaging of modern and legacy storage technologies; protecting evidence media from accidental modification; managing large forensic image files; preserving and verifying evidence integrity with cryptographic and other tools; working with newer drive and interface technologies; managing drive security and acquiring usable images from more complex or challenging situations, such as RAID systems, virtual machine images and damaged media. With its unique focus on digital forensic acquisition and evidence preservation, *Practical Forensic Imaging* is a valuable resource for experienced digital forensic investigators wanting to advance their Linux skills and experienced Linux administrators wanting to learn digital forensics.

<http://nostarch.com>



Red Hat OpenStack Platform

The adoption of OpenStack in production environments has burgeoned, necessitating increased requirements for enhanced management and seamlessly integrated enterprise capabilities. Numerous enterprises worldwide rely on Red Hat's offerings in the OpenStack space—that is, Red Hat OpenStack Platform, a highly scalable, open Infrastructure-as-a-Service (IaaS) platform designed to deploy, scale and manage private cloud, public cloud and Network Functions Virtualization (NFV) environments. The updated Red Hat OpenStack Platform 9, based on the "Mitaka" release from the upstream OpenStack community, brings technical updates across the board, encompassing nearly all of the major OpenStack projects, and features integrated management for OpenStack through Red Hat CloudForms. Red Hat OpenStack Platform 9 builds on the proven, trusted foundation of Red Hat Enterprise Linux to provide critical dependencies needed in production OpenStack environments centered around service functionality, third-party drivers, and system performance and security.

<http://redhat.com>

Please send information about releases of Linux-related products to newproducts@linuxjournal.com or New Products c/o Linux Journal, PO Box 980985, Houston, TX 77098. Submissions are edited for length and content.

[RETURN TO CONTENTS](#)

LOW POWER WIRELESS: 6LoWPAN, IEEE802.15.4 and the Raspberry Pi

The Internet of Things (IoT) is gathering momentum, with predictions of 20 billion devices connected to the internet in just a few years. Many of these devices will be running on batteries and communicating using wireless. One of the emerging standards is 6LoWPAN, IPv6 over low power wireless personal-area networks. This is the first article in a series looking at how to use 6LoWPAN on Linux, using OpenLabs 6LoWPAN modules running on Raspberry Pis.

JAN NEWMARCH



PREVIOUS
New Products

NEXT

Feature: GCC Inline
Assembly and Its Usage
in the Linux Kernel



The Internet of Things (IoT) is one of the new kids on the block. It promises connection of sensors and actuators to the internet, for data to flow both ways, and once on the internet, to become part of new and exciting business systems, reaching up into the realms of big data and artificial intelligence.

IoT applications will rely on a large and complex system. One of the components in this will be the connections between sensors and actuators and the internet. This will most likely be wireless, and it will have to be low power. If you have a thousand sensors, they will most likely be running off batteries, and you will want those batteries to last years, not days.

Low power wireless is heading in two directions right now: personal-area networks (LoWPAN) spanning up to 20–30 meters and wide-area networking (LPWAN) of up to 20 or more kilometers. The technologies at the physical layer are completely different and lead to different Linux solutions. This article deals only with LoWPAN.

The physical layer for LoWPAN is specified by IEEE802.15.4. This defines communication using various wireless bands, such as 2.4GHz, with a range of about 10 meters and data transfer rates of 250kb/s—good enough for most sensors, but not good enough to stream MP3s!

On top of IEEE802.15.4 is a variety of protocols: Zigbee, Z-Wave, Thread and so on. Of these, only the IETF 6LoWPAN is an open standard, and this is where the Linux development community has settled. This article covers only 6LoWPAN. I also ignore other wireless systems, such as Bluetooth LE.

6LoWPAN and Linux

6LoWPAN is IPv6 over IEEE802.15.4 wireless. That isn't easy. IPv6 is designed for the current internet, while IEEE802.15.4 is designed for a different environment. You don't need to worry about how this mismatch has been overcome, but it does mean you need to be aware that two different levels are dealt with here: getting two wireless *devices* to talk to each other and getting a *networking* layer talking over these devices.

The device layer is where physical hardware choices come into play.

For the IoT, it (and the Arduino) form an excellent bridge between the physical and ICT worlds. But, there are now IEEE802.15.4 modules available, and they can be used to turn an RPi into a “full-function 6LoWPAN device”.

Linux supports several devices, such as the AT86RF230 series, the MRF24J40 and several others. The kernel needs to have those device drivers compiled in or available as dynamically loadable modules.

The networking layer requires 6LoWPAN support. Again, the kernel needs to have this compiled in or available as modules. These modules are the `ieee802154_6LoWPAN`, `ieee802154` and `mac802154` modules.

6LoWPAN Devices and the Raspberry Pi

The Raspberry Pi is a wonderful toy or a full-blown Linux computer, depending on your viewpoint. With its GPIO pins, it can act as a connection into the realm of sensors and actuators, while with Ethernet (and on the RPi3, Wi-Fi), it can be a part of LANs and WANs. For the IoT, it (and the Arduino) form an excellent bridge between the physical and ICT worlds. But, there are now IEEE802.15.4 modules available, and they can be used to turn an RPi into a “full-function 6LoWPAN device”.

I used the RPi with the OpenLabs “Raspberry Pi 802.15.4 radio”. This is an Atmel AT86RF233 radio on a small board with a header that allows it to be plugged straight onto pins 15–26 of the RPi. It can be plugged in facing out or facing in—facing in the right way to do it.

I started off using the standard Raspbian distro (dated May 27, 2016). This can be set up to recognize the radio, but—oh dear!—the 4.4 Linux kernel it uses has 6LoWPAN modules, but they don’t work properly in that kernel. The IPv6 packets get corrupted even for pinging itself, so this Raspbian distro won’t support 6LoWPAN.

The hunt is on then for a setup that allows the RPi to support 6LoWPAN with an AT86RF233 radio. This is painful: there are many helpful sites that are outdated or with instructions that I just couldn’t get to work. I finally was pointed by Sebastian Meiling to his page “Create a generic Raspbian image with 6LoWPAN support” (<https://github.com/RIOT-Makers/wpan-raspbian/wiki/Create-a-generic-Raspbian-image-with-6LoWPAN-support#4-new-linux-kernels-for-the-pi>). In summary, what is needed is an upstream Linux kernel, 4.7 or 4.8, recent firmware and suitable configuration of the `/boot/config.txt` file. At the time of this writing, these instructions work only for the RPi 1 and 2. The RPi 3 isn’t working yet, but it may be by the time this article is published.

Installing a 6LoWPAN Kernel

For this article I’m using the OpenLabs module on the RPi 2B. For other modules and RPis, see Sebastian’s page. I’m also going to assume a reasonable amount of Linux savvy in installing software and building from source.

Start by installing the latest Raspbian image. If that runs a 4.7 (or later) kernel, you may be okay already; otherwise, you need to build and install an upstream 4.7 kernel. You probably will need extra tools for this, such as `rpi-update`, `git`, `libncurses5-dev`, `bc` and maybe development tools that you can install using `apt-get`.

Before you do anything else, make sure your system is up to date by running:

```
rpi-update
```

This will install the latest firmware bootloader.

Download a 4.7 kernel into the linux-rpi2 directory with:

```
git clone --depth 1 https://github.com/raspberrypi/linux.git \  
    --branch rpi-4.7.y --single-branch linux-rpi2
```

Building a kernel means compiling a *lot* of files and is *very slow* on the RPi. Most people recommend cross-compiling, but that's more complex, and I like things simple. So, I prefer to build on the RPi itself. It takes only about 5 hours, so start it up, and either go to bed or go out, listen to some jazz and stay out late.

In the linux-rpi2 directory, set up a configuration file for the RPi 2B with:

```
make bcm2709_defconfig
```

Then run `menuconfig` to do two things:

1) Install the device driver as a module from the menu entry:

Device Drivers

```
--> Network device support
```

```
--> IEEE 802.15.4 drivers
```

2) Install 6LoWPAN support as a module from the menu entry:

Networking support

```
--> Networking options
```

```
--> IEEE Std 802.15.4 Low-Rate Wireless Personal Area  
    Networks support
```

Build the kernel and associated files with:

```
make zImage modules dtbs -j4
```

Five hours later, install the modules and dtbs files:

```
sudo make modules_install dtbs_install
```


The safest way to install the kernel is to copy it to an appropriate location. When I run `make kernelversion` in the source tree, it tells me I have built 4.7.2. So I use that number in copying the kernel:

```
sudo cp arch/arm/boot/zImage /boot/kernel.4.7.2.img
```

That way I don't destroy any existing images, so I have a safe fallback to the previous system.

Finally, you need to tell the RPi to boot into the new kernel. As root, edit `/boot/config.txt` and add these lines at the end:

```
kernel=kernel.4.7.2.img  
device_tree=bcm2709-rpi-2-b.dtb  
dtoverlay=at86rf233
```

What does that do? First, it tells the RPi to use the new boot image `kernel.4.7.2.img`. Second—and this is currently ARM-specific—it tells the RPi to pick up hardware default values using the *device tree* system from `bcm2709-rpi-2-b.dtb`. And third—and this is RPi-specific—it says to add in the `at86rf233` device in an additional file to the device tree file.

Finally...reboot. If all went well, you should have the new kernel running. Check this with:

```
uname -a
```

It should show something like this:

```
Linux raspberrypi 4.7.2-v7+ #1 SMP Fri Aug 26 15:45:29 UTC 2016  
➡armv7l GNU/Linux
```

If it didn't boot or showed the wrong kernel, take your SD card back to somewhere else so you can comment out the lines you added to `/boot/config.txt`. Back on the RPi, reboot back into the default kernel, and try to figure out which step went wrong. I skipped some steps from Sebastian's guide because I didn't need them, but if your system isn't working, pay very close attention to his guide. He seems to be pretty diligent about updating it.

Setting Up 6LoWPAN

Are you there yet? Sorry, no. You've built and installed an upstream kernel with 6LoWPAN support. You're more than half-way there though. To configure the 6LoWPAN stack, you need another tool, `wpan-tools`. Get this from GitHub:

```
git clone --depth 1 https://github.com/linux-wpan/wpan-tools.git
↳wpan-tools
```

Before you can build this though, you need `autoreconf`:

```
sudo apt-get install dh-autoreconf
```

Then in the `wpan-tools` directory, you can run:

```
./autogen.sh
./configure CFLAGS='-g -O0' --prefix=/usr --sysconfdir=/etc
↳--libdir=/usr/lib
make
sudo make install
```

What's going on here? Linux is part of the UNIX family of operating systems (including BSD, among many others). They all have quirks, and source code authors have to deal with those. There have been many tools to make this management easier, and `wpan-tools` uses `autoreconf` to build a configuration file, then `configure` to work out the specifics of your RPi system so that when you `make` your application, all of the correct pieces are in place.

The result of this is that the application `iwpan` is now in the `/usr/bin` directory for use.

You're nearly there! Remember in the kernel configuration you set the 6LoWPAN and device drivers to be dynamic modules. They won't have been installed by default like you would expect modules to be. That's what all this device tree stuff is about—bringing devices into the system when it can't detect them normally. So the next step is to load the modules:

```
sudo modprobe at86rf230
```

Then `lsmod` should include something like this:

Module	Size	Used by
<code>ieee802154_6LoWPAN</code>	19335	0
<code>6LoWPAN</code>	13191	8 <code>nhc_fragment</code> , <code>ieee802154_6LoWPAN</code>
<code>at86rf230</code>	22211	0
<code>mac802154</code>	49035	1 <code>at86rf230</code>
<code>ieee802154</code>	55698	2 <code>ieee802154_6LoWPAN</code> , <code>mac802154</code>
<code>crc_ccitt</code>	1278	1 <code>mac802154</code>

And now—ta-da!—`iwpan list` shows something like this:

```
wpan_phy phy0
supported channels:
page 0: 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26
current_page: 0
current_channel: 13, 2415 MHz
cca_mode: (1) Energy above threshold
cca_ed_level: -77
tx_power: 4
capabilities:
ifotypes: node,monitor
channels:
page 0:
[11] 2405 MHz, [12] 2410 MHz, [13] 2415 MHz,
[14] 2420 MHz, [15] 2425 MHz, [16] 2430 MHz,
[17] 2435 MHz, [18] 2440 MHz, [19] 2445 MHz,
[20] 2450 MHz, [21] 2455 MHz, [22] 2460 MHz,
[23] 2465 MHz, [24] 2470 MHz, [25] 2475 MHz,
[26] 2480 MHz
tx_powers: 4,3.7,3.4,3,2.5,2,1,0,-1,-2,-3,-4,-6,-8,-12,-17
cca_ed_levels: -91,-89,-87,-85,-83,-81,-79,-77,-75,-73,-71,
➡-69,-67,-65,-63,-61
cca_modes:
(1) Energy above threshold
(2) Carrier sense only
```

```
(3, cca_opt: 0) Carrier sense with energy above threshold
↳(logical operator is 'and')
(3, cca_opt: 1) Carrier sense with energy above threshold
↳(logical operator is 'or')
min_be: 0,1,2,3,4,5,6,7,8
max_be: 3,4,5,6,7,8
csma_backoffs: 0,1,2,3,4,5
frame_retries: 0,1,2,3,4,5,6,7
lbt: false
Supported commands:
...
```

Your 6LoWPAN device is now known to the Linux system.

Configuring 6LoWPAN

So now you have a new kernel, you have the at86rf230 device recognized, and the 6LoWPAN networking stack is in place. The final steps are to configure networking and bring the device up. You likely are used to Wi-Fi networks having an SSID. IEEE802.15.4 networks have a similar concept, a PAN ID. Two devices will be on the same network only if they have the same PAN ID. You use `iwpan` to set this:

```
iwpan dev wpan0 set pan_id 0xbeef
```

The ID of `0xbeef` isn't fixed, but every example seems to use it! Then, you bring up the interface using normal networking tools:

```
ip link add link wpan0 name lowpan0 type lowpan
ifconfig wpan0 up
ifconfig lowpan0 up
```

What have you got now? `ifconfig` returns something like this:

```
lowpan0  Link encap:UNSPEC  HWaddr
↳EE-0B-FB-0F-76-B9-F3-93-00-00-00-00-00-00-00
inet6 addr: fe80::ec0b:fb0f:76b9:f393/64 Scope:Link
```

```
UP BROADCAST RUNNING MULTICAST  MTU:1280  Metric:1
RX packets:38 errors:0 dropped:0 overruns:0 frame:0
TX packets:39 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:5205 (5.0 KiB)  TX bytes:5205 (5.0 KiB)
```

```
wpan0      Link encap:UNSPEC  HWaddr
↳EE-0B-FB-0F-76-B9-F3-93-00-00-00-00-00-00-00
UP BROADCAST RUNNING NOARP  MTU:123  Metric:1
RX packets:58 errors:0 dropped:0 overruns:0 frame:0
TX packets:55 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:300
RX bytes:4111 (4.0 KiB)  TX bytes:4904 (4.7 KiB)
```

The interface `wpan0` is the wireless device. The interface `lowpan0` is the 6LoWPAN network device, just like `eth0`, the loopback device and so on. Note how it has an IPv6 address, but no IPv4 address—this is next-generation IP only!

Ping!

You are done! Well, almost. There is an old *B.C.* comic strip where one character invents the telephone. “Who do we ring?” asks his friend. “I only invented one” is the reply. You need someone to talk to. So, do all this over again with another RPi. You did buy two RPi and two wireless modules, didn’t you?

The `ifconfig` command tells you the IPv6 address of the 6LoWPAN device. From the *other* device, once you have it set up, do:

```
ping6 -I lowpan0 fe80::ec0b:fb0f:76b9:f393 # IPv6 address of
                                             # the other device
```

or:

```
ping6 fe80::ec0b:fb0f:76b9:f393%lowpan0
```

The `ping6` command is the IPv6 version of `ping`. The IPv6 address

of each network interface is assigned automatically and is a *link local* address.

If you have multiple interfaces, each of them can be on a network segment with non-routable link local addresses. Hosts on these different network segments *can have the same address*. These are like IPv4 link local addresses 169.254.0.0/16, and they can't be routed across different network segments. So in Linux, you need to specify the interface to use (`lowpan0`) to avoid possible confusion. There are two ways of doing this: either use the `-I lowpan0` option or append `%lowpan0` to the IPv6 address.

On my system, this produces:

```
$ping6 -I lowpan0 fe80::ec0b:fb0f:76b9:f393
PING fe80::ec0b:fb0f:76b9:f393(fe80::ec0b:fb0f:76b9:f393) from
↳fe80::f0f9:a4ed:3cad:d1de lowpan0: 56 data bytes
64 bytes from fe80::ec0b:fb0f:76b9:f393: icmp_seq=1 ttl=64
↳time=11.6 ms
64 bytes from fe80::ec0b:fb0f:76b9:f393: icmp_seq=2 ttl=64
↳time=11.1 ms
64 bytes from fe80::ec0b:fb0f:76b9:f393: icmp_seq=3 ttl=64
↳time=10.5 ms
```

Success! The two devices can ping each other across 6LoWPAN. What if it doesn't work? Well, it didn't work for me for a long time, and working out where the failure occurred was painful. It turned out to be a wrong kernel for 6LoWPAN. To troubleshoot, first keep running `ifconfig`. This tells you which interfaces are getting and sending packets. It told me that the wireless layer (`wpan0`) was getting and receiving packets, but the networking layer wasn't. Then I ran `wireshark` using selector `ip6` on packets, and it showed me errors at the network layer. The command `dmesg` gave gold, telling me the IPv6 packets were corrupted, even when pinging myself.

In desperation, I turned to Sebastian, giving him as much information as I could (`uname`, firmware version using `/opt/vc/bin/vcgencmd`, contents of `/boot/config.txt`, decompiling the device tree using `dtc -I fs /proc/device-tree`, and then `wireshark` and `dmesg`

reports). He needed only the first line: wrong kernel. But, spending time working out a detailed report at least shows you are serious. “Duh, it doesn’t work” isn’t helpful to a maintainer!

A Sensor and a Receiver

You don’t really need 6LoWPAN to communicate between Raspberry Pis. Wi-Fi and Ethernet are better. But now suppose one of them is a sensor running off a battery or solar panel. Wi-Fi is estimated to drain a battery within a fortnight; whereas 6LoWPAN on batteries can be expected to run for several years. I’m simulating this here by using one of the RPi as sensor for convenience.

To follow along, you will need to set up a client-server system. Usually, people think of servers as big grunty machines somewhere, but in the IoT world, the sensors will be the servers, handling requests for values from clients elsewhere in the network.

The server is just like a normal IPv6 server as described in the Python documentation: 18.1. `socket` —Low-level networking interface (<https://docs.python.org/3/library/socket.html>). But note that just as with the `ping6` command above, you need to specify the network interface to be used. This means you have to use Python 3 rather than Python 2, as this has the socket function `socket.if_nametoindex()` that allows you to specify the IPv6 “scope id”, which is the interface you use.

I don’t want to complicate this article with how to add sensors to an RPi. Instead, I’ll just measure the temperature of the RPi’s CPU, as this can be found really easily by running this command from a shell:

```
vcgencmd measure_temp
```

This will return a string like:

```
temp=36.9'C
```

Within Python, you create a process to run this command using `Popen` and read from the `stdout` pipeline.

Here’s an IPv6 TCP server that waits for connections, sends the

temperature and then closes the connection:

```
#!/usr/bin/python3

import socket
from subprocess import PIPE, Popen

HOST = ''      # Symbolic name meaning all available interfaces
PORT = 2016    # Arbitrary non-privileged port

def get_cpu_temperature():
    process = Popen(['vcgencmd', 'measure_temp'], stdout=PIPE)
    output, _error = process.communicate()
    return output

def main():
    s6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    scope_id = socket.if_nametoindex('lowpan0')
    s6.bind((HOST, PORT, 0, scope_id))
    s6.listen(1)

    while True:
        conn, addr = s6.accept()
        conn.send(get_cpu_temperature())
        conn.close()

if __name__ == '__main__':
    main()
```

And, here's a client that opens a connection and reads the temperature every ten seconds:

```
#!/usr/bin/python3

import socket
import time
```



```
ADDR = 'fd28:e5e1:86:0:e40c:932d:df85:4be9' # the other RPi
PORT = 2016

def main():
    # scope_id = socket.if_nametoindex('lowpan0')
    while True:
        s6 = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
        s6.connect((ADDR, PORT, 0, 0))
        data = s6.recv(1024)
        print(data.decode('utf-8'), end='')

        # get it again after 10 seconds
        time.sleep(10)

if __name__ == '__main__':
    main()
```

The output looks like this:

```
temp=37.4'C
temp=37.4'C
temp=37.9'C
```

What's My Server's Address?

So imagine you've now got 1,000 of these sensors scattered out in the wild somewhere, and they are all running IPv6 servers. What are their addresses? How do you talk to them? Unfortunately, the OpenLabs module generates a new MAC address each time it is booted, so it generates a new IPv6 address each time. Running multi-cast discovery is not recommended for these low power networks as it is a power drain. I will cheat a bit in the next article, but show better ways in the third article.

Conclusion

The scenario presented in the last section is still a bit unrealistic. If you have enough power to drive an RPi as a sensor, you probably

have enough power for it to use Wi-Fi or Ethernet. But soon there will be genuine low power sensors using 6LoWPAN, and this article has shown you how to bring them into one particular Linux system. It's been pretty heavy going, but right now this is cutting-edge stuff, so expect to bleed a bit!

In my next article, I'll describe how to bring a 6LoWPAN network into the standard IPv6 world, and in the third article, I plan to look at CoAP, the equivalent of HTTP for low power networks. ■

Jan Newmarch has been using Linux since kernel 0.96. He has written many books and papers about software engineering, network programming, user interfaces and artificial intelligence, and he is currently digging into the IoT. He is in charge of ICT degrees at Box Hill Institute.

RESOURCES

OpenLabs Raspberry Pi 802.15.4 Radio:
<http://openlabs.co/OSHW/Raspberry-Pi-802.15.4-radio>

Python API: socket — Low-level networking interface:
<https://docs.python.org/3/library/socket.html>

IETF RFC4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks (6LoWPAN): <https://tools.ietf.org/html/rfc4944>

6LoWPAN: The Wireless Embedded Internet by Zach Shelby, Wiley 2009

Create a generic Raspbian image with 6LoWPAN support by Sebastian Meiling:
<https://github.com/RIOT-Makers/wpan-raspbian/wiki/Create-a-generic-Raspbian-image-with-6LoWPAN-support>

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

[RETURN TO CONTENTS](#)

Linux Journal eBook Series

GEEK GUIDES

Practical books for the most technical people on the planet.

FREE
Download
NOW!



**BotFactory:
Automating
the End
of Cloud
Sprawl**

Author:
John S. Tonello
Sponsor:
BotFactory.io



**Containers
101**

Author:
Sol Lederman
Sponsor:
Puppet



**An API
Marketplace
Primer
for Mobile,
Web and
IoT**

Author:
Ted Schmidt
Sponsor:
IBM



**Public
Cloud
Scalability
for
Enterprise
Applications**

Author:
Petros Koutoupis
Sponsor:
SUSE

Go to <http://geekguide.linuxjournal.com>

GCC Inline Assembly

and Its Usage in the Linux Kernel

Learning GCC inline assembly just got one more benefit. Now let's dive in to the kernel to see how a few things actually work.

DIBYENDU ROY



PREVIOUS
Feature:
Low Power Wireless

NEXT
Doc Searls' EOF



The GNU C compiler allows you to embed assembly language code into C programs. This tutorial explains how you can do that on the ARM architecture. As GNU assembler is similar for different architectures, including assembler syntax and most assembler directives, the general concepts of inline assembly remain same for other architectures as well.

Why should you embed assembly code into C? There are at least two reasons:

- **Optimization:** the compiler tends to optimize unless specified otherwise. For some applications, however, hand-written assembly replaces the most performance-sensitive parts. Because the inline assembler does not require separate assembling and linking, it is more convenient than a separately written assembly module. Inline assembly code can use any C variable or function name that is in scope, so it is easy to integrate it with your C code.
- **Access to processor-specific instructions:** C does not support saturated math operations, co-processor instructions or accessing the Current Program Status Registers (CPSR). C code also does not support ARM LDREX/STREX instructions. ARM implements its atomic operations and locking primitives with LDREX/STREX. Inline assembly is the easiest way to access instructions not supported by the C compiler.

Getting Started

Let's start with the simple example shown in Listing 1.

The part from the example program in Listing 1 that needs explanation is this:

```
asm volatile("add %[Rd], %[Rm], %[Rn]"
             : [Rd] "=r" (result)
             : [Rm] "r" (x), [Rn] "r" (y)
             );
```

Before explaining the code, let's start with the basics. The `asm` keyword enables you to embed assembler instructions within C code.

Listing 1. Example Program

```
#include<stdio.h>

int add(int x, int y)
{
    int result;
    asm volatile("add %[Rd], %[Rm], %[Rn]"
                 : [Rd] "=r" (result)
                 : [Rm] "r" (x), [Rn] "r" (y)
                 );
    return result;
}

int main(void)
{
    int ret;
    ret = add(5, 7);
    printf("the result is = %d\n", ret);
    return 0;
}
```

GCC has two forms of inline `asm` statements: basic `asm` and extended `asm`. A basic `asm` is one with no operands, while an extended `asm` includes one or more operands. Basic `asm` enables you to include assembly language outside any function. The extended form is preferred for mixing C and assembly languages within a function.

Basic `asm` and Extended `asm`

A basic `asm` statement has the following format:

```
asm [volatile] (Assembly code)
```

The `volatile` qualifier is optional here. All basic `asm` statements are implicitly `volatile`.

Assembly code is a string that can contain any assembly instruction(s) recognized by the GNU assembler, including directives. A C compiler does not parse or check the validity of the assembly instructions. Assembly code parsing and syntax checking is done at the assembling stage. A single `asm` string may contain multiple assembler instructions. You can use a newline followed by a tab (`\n\t`) to break and indent the code in the next line.

Below is an example of basic `asm` in the kernel (`arch/arm/include/asm/barrier.h`):

```
#define nop() __asm__ __volatile__ ("mov\t r0,r0\t@ nop\n\t");
```

This is simply:

```
asm volatile("mov r0,r0");
```

The above inline assembly copies the `r0` register content to itself. `nop()` ends up only introducing some delay.

Note that the `asm` keyword is a GNU extension. Use `__asm__` instead of `asm` when your code is compiled with `-ansi` and the various `-std` options. The Linux kernel uses both `__asm__` and `asm` for compatibility.

An extended asm statement has the following syntax:

```
asm [volatile] (Assembly code
                : OutputOperands /* optional */
                : InputOperands  /* optional */
                : Clobbers       /* optional */
                )
```

The `volatile` qualifier is optional here. However, `asm` statements may produce side effects while operating on inputs and generating outputs. You may need to use the `volatile` qualifier to disable certain optimizations in that case.

Assembly code is a string literal that is a combination of fixed text and tokens that refer to the input and output parameters. `OutputOperands` and `InputOperands` are optional comma-separated lists of C variables. `Clobbers` are also an optional comma-separated list of registers or other special values. Read on for more about these.

Coming Back to the Example

The example program from Listing 1 includes an extended `asm` statement. Colons delimit each operand parameter after the assembly code:

```
"add %[Rd], %[Rm], %[Rn]"
```

This is the string literal containing the assembly code:

```
[Rd] "=r" (result)
```

Output operands consist of a symbolic name enclosed in a square bracket, followed by a constraint string and a C variable name enclosed in parentheses:

```
[Rm] "r" (x), [Rn] "r" (y)
```

The list of input operands uses similar syntax as output operands.

More on Output, Input and Clobbers

Output Operands `outputOperands` has the following format:

```
[asmSymbolicName] constraint (cvariablename)
```

An `asm` statement has zero or more output operands indicating the names of C variables modified by the assembler code. `asmSymbolicName` specifies a symbolic name for the operand. Square brackets are used to reference this inside the `asm` statement. The scope of the name is the `asm` statement that contains the definition.

You also can use the position of the operands in the assembler template (for example, if there are three operands, `%0` to the first, `%1` for the second and `%2` for the third). You can re-write the example code as:

```
asm volatile("add %0, %1, %2"
             : "=r" (result)
             : "r" (x), "r" (y)
             );
```

A constraint is a string constant specifying restrictions on the placement of the operand. Refer to the GCC documentation for a full list of supported constraints for ARM and other architectures. The most commonly used constraints are `"r"`, used as general-purpose registers (`r0` to `r15`); `"m"`, which refers to any valid memory location, and `"l"` for immediate integer. Constraint character may be prefixed with constraint modifiers:

- `=` — write-only operand, used for output operands.
- `+` — read-write operand, must be listed as an output operand.
- `&` — register used for output only.

Output operators must be write-only, and input operands are read-only. Constraints without any modifiers are read-only. So, it should be clear why the output operand in the example program has `"=r"` and input

operands "r".

But, what if your input and output operands are the same? "+r" can be used as a constraint and must be listed as output operands:

```
asm volatile("mov %[Rd], %[Rd], lsl #2"
             : [Rd] "+r" (x)
             );
```

The assembly code goes here:

```
#APP
@ 5 "inline_shift.c" 1
    mov r3, r3, lsl #2
@ 0 "" 2
```

Sometimes a compiler may choose the same register for input and output, even if you do not instruct it to do so. If your code explicitly requires different registers for input and output operands, use the "&" constraint modifier.

Constraints in an output operand should follow a `cvariable` name that must be an `lvalue` expression for output operands.

Input Operands Input operands have a similar syntax as output operands. But, constraints should not start with "=" or "+". Input operands' constraints for registers do not have any modifiers, as they are read-only operands. You should never try to modify the contents of input-only operands. Use "+r" when input and output operands are the same, as explained above.

Clobbers Sometimes inline assembly may modify additional registers, as side effects, apart from those listed in the output operands. In order to make the compiler aware of this additional change, you need to list them in a clobber list. Clobber list items are either register names or the special clobbers. Each clobber list item is a string constant and is separated by commas. When the compiler allocates registers for input and output operands, it does not use any of the clobbered registers. Clobbered registers are available for any use in the assembler code. Let's take a closer look at an inline add program that does not have a clobber list.

The inline assembly code may look like this:

```
#APP
@ 6 "inline_add.c" 1
    add r3, r3, r2
@ 0 "" 2
```

Here the code uses register r3 and r2. Now let's modify it and list the r2 and r3 registers in a clobber list:

```
asm volatile("add %[Rd], %[Rm], %[Rn]"
             : [Rd] "=r" (result)
             : [Rm] "r" (x), [Rn] "r" (y)
             : "r2", "r3"
             );
```

The assembly code:

```
#APP
@ 6 "inline_add2.c" 1
    add r4, r1, r0
@ 0 "" 2
```

Notice that the compiler did not use the r2 and r3 registers as they were mentioned in the clobber list. The processor can use r2 and r3 for any other work in the assembly code.

There are also two special clobbers available apart from registers: "cc" and "memory". The "cc" clobber indicates that the assembler code modifies the CPSR (Current Program Status Register) flag register. The "memory" clobber tells the compiler that the inline assembly code performs memory reads or writes on items apart from input and output operands. The compiler flushes the register contents to memory so that memory contains the correct value before executing the inline asm. Moreover, the compiler reloads all memory access after the inline asm statement so that it gets a fresh value. This way, the "memory" clobbers form a read-write compiler barrier across the inline asm statement.

In Linux, a compiler barrier is defined as a macro `barrier()` that is nothing but a memory clobber:

```
#define barrier() __asm__ __volatile__("" : : "memory")
```

Important:

- Use `__asm__` instead of `asm` when your code is compiled with `-ansi` and the various `-std` options.
- The difference between basic and extended `asm` is the latter has optional output, input and clobber lists separated by colons (:).
- Extended `asm` statements must be inside a function. Only basic `asm` statements may be outside functions.
- Inside a function, extended `asm` statements typically produce more efficient and robust code.

Inline Assembly in the Linux Kernel

Now that I've gone through the basics of GCC inline assembly, let's move on to a more interesting topic—its usage in the Linux kernel. The rest of this article is architecture-dependent and is discussed with respect to ARMv7-A. Basic knowledge of ARM and assembly language will be helpful in understanding the rest of the material covered here.

A Little Background In multitasking computers, shared resource accesses must be restricted to only one modifier at a time. This shared resource can be a shared memory location or a peripheral device. Mutual exclusion, a property of concurrency control, protects such shared resources. In a single processor system, disabling interrupts could be a way of achieving mutual exclusion inside critical sections (although user mode cannot disable interrupts), but this solution fails in SMP systems as disabling interrupts on one processor will not prevent others from entering the critical section. Atomic operations and locks are used to enforce mutual exclusion.

Mutual exclusion enforces atomicity. Let's consider the definition

of atomicity first. Any operation is atomic if the operation is entirely successful and its result is visible to all CPUs in the system instantaneously, or it's not successful at all. Atomicity is the basis of all mutual exclusion methods.

All modern computer architectures, including ARM, provide hardware mechanisms for atomically modifying the memory locations.

The ARMv6 architecture introduced the concept of exclusive accesses to memory locations for atomically updating memory. The ARM architecture provides instructions to support exclusive access.

LDREX (Load Exclusive) loads the value of a given memory location into a register and tags that memory location as reserved.

STREX (Store Exclusive) stores an updated value from a register back to a given memory location, provided that no other processor has modified the physical address since its last load. It returns 0 for success, and 1 otherwise, to a register indicating whether the store operation completed successfully. By checking this return value, you can confirm whether any other processor has updated the same location in between.

These instructions need hardware support to tag a physical address as "exclusive" by that specific processor.

Note: ARM says:

If a context switch schedules out a process after the process has performed a Load-Exclusive but before it performs the Store-Exclusive, the Store-Exclusive returns a false negative result when the process resumes, and memory is not updated. This does not affect program functionality, because the process can retry the operation immediately.

The concept of exclusive accesses also is related to the concepts of local and global monitors, memory types, memory access ordering rules and barrier instructions. See the Resources section of this article for more information.

Implementation of Atomic Operations Atomic integer operations are generally required to implement counters. As protecting a counter with a complex locking scheme is overkill, `atomic_inc()` and `atomic_dec()` are preferable. All the atomic functions in the Linux kernel are implemented using LDREX and STREX.

Take a look at `atomic_t` defined in `include/linux/types.h` as the following:

```
typedef struct {
    int counter;
} atomic_t;
```

After simplifying the macro definitions, the `atomic_add()` function definition in kernel-4.6.2 (`arch/arm/include/asm/atomic.h`) looks like Listing 2.

Let's take a closer look at the code shown in Listing 2.

The function below uses PLD (Preload Data), PLDW (Preload Data with intent to write) instructions that are typically memory system hints to bring the data into caches for faster access:

```
prefetchw(&v->counter);
```

Listing 2. `atomic_add()` Implementation

```
static inline void atomic_add(int i, atomic_t *v)
{
    unsigned long tmp;
    int result;

    prefetchw(&v->counter);
    __asm__ __volatile__("@ atomic_add\n"
"1: ldrex  %0, [%3]\n"
"   add   %0, %0, %4\n"
"   strex %1, %0, [%3]\n"
"   teq   %1, #0\n"
"   bne   1b"
: "=&r" (result), "=&r" (tmp), "+Qo" (v->counter)
: "r" (&v->counter), "Ir" (i)
: "cc");
}
```

`ldrex` loads the “counter” value to “result” and tags that memory location as reserved:

```
ldrex    %0, [%3]
```

The following adds `i` to the “result” and stores that to “result”:

```
add     %0, %0, %4
```

Two scenarios are possible here:

```
strex   %1, %0, [%3]
```

In first scenario, `strex` successfully stores the value of “result” into the memory location and returns 0 at “tmp”. This happens only when no other processor has modified the location in between the last load and store by the current processor. However, if any other processor has modified the same physical memory in between, the current processor’s store fails. In this case, it returns 1 at “tmp”.

This instruction tests equivalence and sets the Z (zero) flag of CPSR if “tmp” is 0 or clears it if “tmp” is 1:

```
teq     %1, #0
```

For a successful store scenario, the Z flag is set. So, the branch condition does not satisfy. However, if store fails, the branch takes place and execution starts again from the `ldrex` instruction. The loop continues until store is successful:

```
bne     1b
```

All other atomic operations are similar and use `LDREX` and `STREX`.

Barriers If a sequence of memory operations is independent, the compiler or CPU performs it in a random fashion to achieve optimization—for example:

```
a = 1;  
b = 5;
```

However, to synchronize with other CPUs or with hardware devices, it is sometimes a requirement that memory-reads (loads) and memory-writes (stores) issue in the order specified in your program code. To enforce this ordering, you need barriers. Barriers are commonly included in kernel locking, scheduling primitives and device driver implementations.

Compiler Barrier The compiler barrier does not allow the compiler to re-order any memory access across the instruction. As discussed before, the `barrier()` macro is used as a compiler barrier in Linux:

```
#define barrier() __asm__ __volatile__("" : : "memory")
```

Processor Barriers Processor optimizations, such as caches, write buffers and out-of-order execution, can result in memory operations occurring in a different sequence from the program order. A processor barrier is an implied compiler barrier as well. ARM has three hardware barrier instructions:

1. Data Memory Barrier (DMB) ensures that all memory accesses (in program order) before the barrier are visible in the system before any explicit memory accesses after the barrier. It does not affect instruction prefetch or execution of the next non-memory data access.
2. Data Synchronization Barrier (DSB) ensures that all pending explicit data accesses complete before any additional instructions execute after the barrier. It does not affect prefetching of instructions.
3. Instruction Synchronization Barrier (ISB) flushes the pipeline and prefetch buffer(s) so that once ISB has completed, the processor can fetch the next instructions from cache or memory.

Listing 3. Implementation of the Memory Barrier

```
#define dmb(option) __asm__ __volatile__ ("dmb " #option : : : "memory")  
#define dsb(option) __asm__ __volatile__ ("dsb " #option : : : "memory")  
#define isb(option) __asm__ __volatile__ ("isb " #option : : : "memory")
```


To execute any critical section code atomically, you need to ensure that no two threads of execution should execute critical sections concurrently.

SY is the default. It applies to the full system, including all processors and peripherals. Refer to the ARM manual for other options. Linux provides various memory barrier macros that are mapped to the ARM hardware barrier instructions: read memory barrier, `rmb()`; write memory barrier, `wmb()`; and full memory barrier, `mb()`. There also are corresponding SMP versions: `smp_rmb()`, `smp_wmb()` and `smp_mb()`. When the kernel is compiled without `CONFIG_SMP`, `smp_*` are simply `barrier()` macros.

Spinlock To execute any critical section code atomically, you need to ensure that no two threads of execution should execute critical sections concurrently. As described in Robert Love's *Linux Kernel Development*, "The term threads of execution implies any instance of executing code. This includes, for example, a task in the kernel, an interrupt handler, a bottom half, or a kernel thread."

For uniprocessor systems, spinlock implementation boils down to disabling preemption or local interrupts. `spin_lock()` disables preemption. `spin_lock_irq()` and `spin_lock_irqsave()` disable local interrupts. But, this is not sufficient for SMP, as other processors are free to execute the critical section code simultaneously.

Linux uses an improved version of the ticket lock algorithm to implement spinlock. Like atomic instructions, the spinlock implementation uses LDREX/STREX.

Listing 4. Spinlock Implementation

```

static inline void arch_spin_lock(arch_spinlock_t *lock)
{
    unsigned long tmp;
    u32 newval;
    arch_spinlock_t lockval;

    prefetchw(&lock->slock);
    __asm__ __volatile__(
"1: ldrex    %0, [%3]\n"
"   add %1, %0, %4\n"
"   strex   %2, %1, [%3]\n"
"   teq %2, #0\n"
"   bne 1b"
        : "=&r" (lockval), "=&r" (newval), "=&r" (tmp)
        : "r" (&lock->slock), "I" (1 << TICKET_SHIFT)
        : "cc");

    while (lockval.tickets.next != lockval.tickets.owner) {
        wfe();
        lockval.tickets.owner = ACCESS_ONCE(lock->tickets.owner);
    }

    smp_mb();
}

static inline void arch_spin_unlock(arch_spinlock_t *lock)
{
    smp_mb();
    lock->tickets.owner++;
    dsb_sev();
}

#define wfe()    __asm__ __volatile__ ("wfe" : : : "memory")

#define sev()    __asm__ __volatile__ ("sev" : : : "memory")

```

The `wfe` (Wait For Event) and `sev` (Send Event) ARM instructions need some introduction here. `wfe` puts the ARM processor into a lower power state until a wake-up event occurs. The wake-up events for `wfe` include the execution of an `sev` instruction on any processor on an SMP system, an interrupt, an asynchronous abort or a debug event. While contending for a spinlock, the processor goes to a low power state instead of being busy waiting, hence saving power. The `ACCESS_ONCE` macro prevents the compiler from an optimization that forces it to fetch the `lock->tickets.owner` value each time through the loop. A memory barrier `smp_mb()` is required after you get a lock and before you release it, so that other processors can be updated on time with whatever is happening on the current processor.

Note: acquiring and releasing a lock should be atomic. Otherwise, more than one thread of execution may acquire the same lock in parallel causing a race condition.

Semaphore Semaphores and mutexes can sleep, unlike a spinlock. When a task is holding a semaphore and another task attempts to acquire it, the semaphore places the contended task onto a wait queue and puts it to sleep. When the semaphore becomes

Listing 5. Semaphore Implementation

```
int down_interruptable(struct semaphore *sem)
{
    unsigned long flags;
    int result = 0;

    raw_spin_lock_irqsave(&sem->lock, flags);
    if (likely(sem->count > 0))
        sem->count--;
    else
        result = __down_interruptable(sem);
    raw_spin_unlock_irqrestore(&sem->lock, flags);

    return result;
}
```

Listing 6. Mutex Implementation

```

void __sched mutex_lock(struct mutex *lock)
{
    might_sleep();
    /*
     * The locking fastpath is the 1->0 transition from
     * 'unlocked' into 'locked' state.
     */
    __mutex_fastpath_lock(&lock->count, __mutex_lock_slowpath);
    mutex_set_owner(lock);
}

```

available, the scheduler wakes one of the tasks on the wait queue to acquire the semaphore. As you can see in Listing 5, the semaphore implementation uses `raw_spin_lock_irqsave()` and `raw_spin_unlock_irqrestore()` to acquire the lock. If another task is holding the semaphore, the current task releases the spinlock and goes to sleep (as sleeping is not an option while holding the spinlock), and after waking up, it re-acquires the spinlock. `up()` is used to release the semaphore that also uses the spinlock. `up()` may be called from any context and even by tasks that have never called `down()`, unlike mutexes.

Mutex A call to a mutex may take two different paths. First, it calls `__mutex_fastpath_lock()` to acquire the mutex. Then it falls back to `__mutex_lock_slowpath()` if it fails to acquire the lock. In the latter case, the task is added to the wait queue and sleeps until woken up by the unlock path.

`__mutex_fastpath_lock` is a call to `atomic_sub_return_relaxed()` that is an atomic operation—atomically subtract `i` from `v` and return the result. Similarly, `mutex_unlock()` uses `atomic_add_return_relaxed` for incrementing the counter atomically.

Wrapping It All Up

This article neither aims to provide algorithmic details of kernel

implementation of locks and barriers nor does it provide ARM architecture details. The goal is to provide the basics of GCC inline assembly and show how it can help you better understand the Linux kernel. ■

Dibyendu Roy is a Linux fundamentalist and works as an embedded Linux developer in Hyderabad, India. Being an open-source activist, he uses Linux for every silly little thing. You can reach him at diby.roy@gmail.com.

RESOURCES

Using the GNU Compiler Collection: <https://gcc.gnu.org>

ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition: <http://infocenter.arm.com>

ARM Synchronization Primitives Development Article: <https://developer.arm.com>

Cortex-A Series Programmer's Guide: <http://infocenter.arm.com>

Linux Kernel Development 3rd Edition by Robert Love:

<https://www.amazon.com/Linux-Kernel-Development-Robert-Love/dp/0672329468>

Inline assembler (Wikipedia): https://en.wikipedia.org/wiki/Inline_assembler

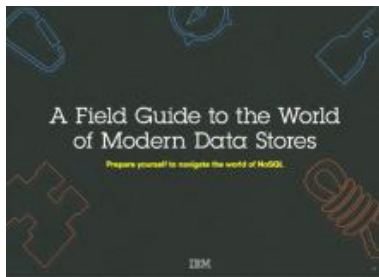
ARM GCC Inline Assembler Cookbook: <http://www.ethernut.de/en/documents/arm-inline-asm.html>

See also the kernel documentation on memory barriers, spinlock and mutex design.

Send comments or feedback via

<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

[RETURN TO CONTENTS](#)



A Field Guide to the World of Modern Data Stores

There are many types of databases and data analysis tools to choose from when building your application. Should you use a relational database? How about a key-value store? Maybe a document database? Is a graph database the right fit? What about polyglot persistence and the need for advanced analytics?

If you feel a bit overwhelmed, don't worry. This guide lays out the various database options and analytic solutions available to meet your app's unique needs.

You'll see how data can move across databases and development languages, so you can work in your favorite environment without the friction and productivity loss of the past.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/field-guide-world-modern-data-stores>



Why NoSQL? Your database options in the new non-relational world

The continual increase in web, mobile and IoT applications, alongside emerging trends shifting online consumer behavior and new classes of data, is causing developers to reevaluate how their data is stored and managed. Today's applications require a database that is capable of providing a scalable, flexible solution to efficiently and safely manage the massive flow of data to and from a global user base.

Developers and IT alike are finding it difficult, and sometimes even impossible, to quickly incorporate all of this data into the relational model while dynamically scaling to maintain the performance levels users demand. This is causing many to look at NoSQL databases for the flexibility they offer, and is a big reason why the global NoSQL market is forecasted to nearly double and reach USD3.4 billion in 2020.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/why-nosql-your-database-options-new-non-relational-world>



RunKeeper Case Study

Boston-based fitness start-up RunKeeper was struggling with its database, and could not keep pace with the company's expansion. With new users joining every day, this limitation threatened to halt the company's operations. With a database of 30 million users and growing fast, scaling up also became an issue.

RunKeeper's initial database PostgreSQL failed to provide the required speed and scale. Partnering with IBM, RunKeeper transformed using IBM Cloudant's Dedicated Cluster as its new data layer.

"We were impressed by the wealth of experience that the IBM team was able to draw on to adapt the solution to meet our business needs," says Joe Bondi, CTO and Co-founder of RunKeeper.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/run-keeper-case-study>



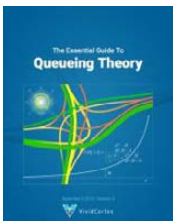
The 2016 State of DBaaS Report: How managed services are transforming database administration

If you didn't have to manage your database, what would you do with your free time? All those hours you previously spent micromanaging your data layer—ensuring it keeps your application running 24/7 and is able to scale up or down based on demand— would suddenly reappear in your day. You could spend more time building your applications, from adding key features to improving the experience of your users, and you would even get some hours back in your personal life.

The 2016 State of DBaaS Report, commissioned by IBM, assessed the business and technical impact of database-as-a-service (DBaaS), as identified by 680 executive and technical enterprise users, and found that developers are saving a substantial amount of time after adopting DBaaS. All of those surveyed were using a managed, NoSQL database service across a variety of industries, including insurance, healthcare, gaming, retail and finance.

Sponsor: IBM

> <https://geekguide.linuxjournal.com/content/2016-state-dbaas-report-how-managed-services-are-transforming-database-administration>



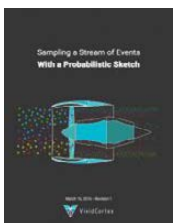
The Essential Guide To Queueing Theory

Whether you're an entrepreneur, engineer, or manager, learning about queueing theory is a great way to be more effective. Queueing theory is fundamental to getting good return on your efforts. That's because the results your systems and teams produce are heavily influenced by how much waiting takes place, and waiting is waste. Minimizing this waste is extremely important. It's one of the biggest levers you will find for improving the cost and performance of your teams and systems.

Author: Baron Schwartz

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/essential-guide-queueing-theory>



Sampling a Stream of Events With a Probabilistic Sketch

Stream processing is a hot topic today. As modern Big Data processing systems have evolved, stream processing has become recognized as a first-class citizen in the toolbox. That's because when you take away the how of Big Data and look at the underlying goals and end results, deriving real-time insights from huge, high-velocity, high-variety streams of data is a fundamental, core use case. This explains the explosive popularity of systems such as Apache Kafka, Apache Spark, Apache Samza, Apache Storm, and Apache Apex—to name just a few!

Author: Baron Schwartz

Sponsor: VividCortex

> <https://geekguide.linuxjournal.com/content/sampling-stream-events-probabilistic-sketch>

Pancaking the Pyramid Economy

How Linux development models the networked world for the rest of us.



DOC SEARLS

Doc Searls is Senior Editor of *Linux Journal*. He is also a fellow with the Berkman Center for Internet and Society at Harvard University and the Center for Information Technology and Society at UC Santa Barbara.

PREVIOUS

◀ Feature: GCC Inline Assembly and Its Usage in the Linux Kernel

In 1937, Ronald Coase gave economics something new: a theory for why companies should exist (<http://www.economist.com/news/leaders/21584985-anyone-who-cares-about-capitalism-and-economics-should-mourn-death-ronald-coase-man>). Oddly, this hadn't come up before. His paper was called "The Nature of the Firm" (https://en.wikipedia.org/wiki/The_Nature_of_the_Firm). He wrote it at age 27, as a class assignment in grad school. He based it on a talk he gave at 22. It has since earned him a Nobel prize. Says *The Economist*:

Mr Coase argued that firms make economic sense because they can reduce or eliminate the "transaction

cost” of going to the market by doing things in-house. It is easier to co-ordinate decisions. At the time, when communications were poor and economies of scale could be vast, this justified keeping a lot of things inside a big firm, so car-makers often owned engine-makers and other suppliers.

In that same piece (a 2013 obituary for Coase), *The Economist* adds:

Mr Coase’s theory of the firm would suggest that firms ought to be in retreat at the moment, because technology is lowering transaction costs: why go to the bother of organising things under one roof when the internet lowers the cost of going to the market?

Could be that Coase has an answer for that one too. In a 2012 interview with Russ Roberts on the EconTalk podcast, when Coase was 102 years old, he said, “It’s not possible to study how things are dealt with without realizing the importance of the stupidity of human behavior” (http://www.econtalk.org/archives/2012/05/coase_on_extern.html).

Perhaps that’s why Hugh MacLeod (aka @gapingvoid, <http://gapingvoid.com>) in 2004 produced the cartoon shown in Figure 1 outlining his own model for the firm.

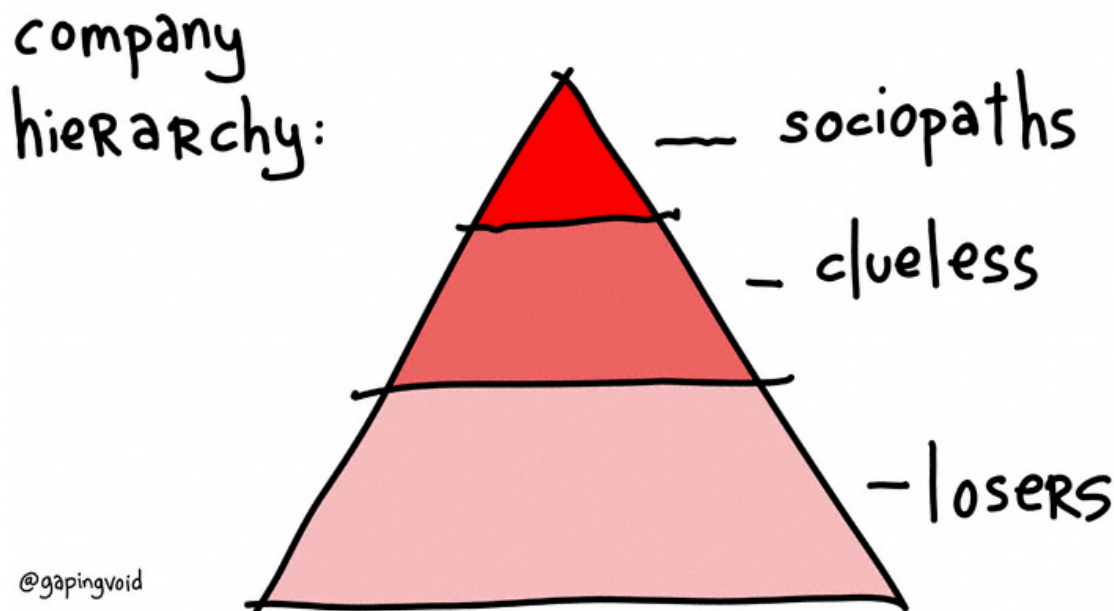


Figure 1. Cartoon by Hugh MacLeod (aka @gapingvoid) from 2004 that outlines his own model of the firm.



Figure 2. Hugh MacLeod's drawing of a company embodying "egology" from 2004.

That same year, Hugh and I brainstormed the future of business for a (now gone) open-source company we both consulted. As happens with Hugh, this generated lots of great illustrations. Figure 2 shows Hugh's drawing of a company like the one above, embodying what he called "egology".

Figure 3 shows what we both saw happening, inevitably.

That ecology was, and remains, the internet. The lines and dots in Figure 3 are employees and customers, all more native to the networked world than to any employer or "brand".

"The Internet is nothing less than an extinction-level event for the traditional firm", writes Esko Kilpi in his new book, *Perspectives on new work* (<https://twitter.com/eskokilpi>). That's because, as Vint Cerf puts it in the forward, Esko sees "a new kind of workforce" that will be "more independent and inter-dependent than workers of the past", thanks to the net. Before the net, full independence and inter-dependence for workers was hard or impossible. Now it is becoming necessary.



Figure 3. Hugh MacLeod’s Drawing of the Internet Ecology

Work also happens on the demand side of the marketplace, where the customers are. That’s us. We far outnumber the companies we sustain with our money and our sentiments. When we become fully independent and inter-dependent, we will cease being grass roots and instead become forests of trees. And when that happens, companies on the supply side must adapt or get composted.

ADVERTISER INDEX

Thank you as always for supporting our advertisers by buying their products!

ADVERTISER	URL	PAGE #
Drupalize.me	http://drupalize.me	35
O'Reilly Software Architecture Conference	http://conferences.oreilly.com/software-architecture/engineering-business-ca	43
Peer 1 Hosting	http://go.peer1.com/linux	13
SeaGL	http://SeaGL.org	49
SUSECON	http://susecon.com	7

ATTENTION ADVERTISERS

The *Linux Journal* brand's following has grown to a monthly readership nearly one million strong. Encompassing the magazine, Web site, newsletters and much more, *Linux Journal* offers the ideal content environment to help you reach your marketing objectives. For more information, please visit <http://www.linuxjournal.com/advertising>

Adaptation will require making the most of everybody who is both independent and inter-dependent: workers on the inside and customers on the outside. No more losers working for sycophants and sociopaths. And no more consumers content with being captive in silos and walled gardens. If your company continues to look toward customers as just “consumers” to “target”, “capture”, “acquire”, “manage”, “control” and “lock in”, you’re manure.

The model for how people work in the new economy was, and remains, open-source programming. That’s what Hugh and I saw in 2004, and the observation hasn’t changed.

Prime example: Linux kernel hackers. Their Linux work is for the kernel—meaning the whole world that depends on that kernel—and not just for their employers. If anyone wants to dispute that, I suggest reading what I wrote about it for *Linux Journal* in 2005 (<http://www.linuxjournal.com/article/8664>) and 2008 (<http://www.linuxjournal.com/content/linux-now-slave-corporate-masters>). The case (made by Linus Torvalds and Andrew Morton) hasn’t changed.

In a fully networked workplace and marketplace, everybody who works for a company needs to be just as independent and inter-dependent as a Linux kernel hacker. And everybody in the marketplace should be the same.

Once the results of that independence and interdependence start to pay off, the market will flatten, because the connections will disintermediate systems built to capture and control, rather than to liberate and enable. Egology will drown in ecology.

But what will make this happen? In a word, cost. In the networked world, holding employees and customers captive is more costly than liberating both and letting the whole market do its work.

For years I’ve been saying that free customers will prove more valuable than captive ones. A corollary is that free markets will prove more valuable than captive ones. Same causes, same effects.

Right now we are part-way there, staging the flattened future with new “disruptive” intermediaries that appear to create a whole new model, when in fact they’re the old model done better. But hey, it’s a start. Case in point: Uber. As I write this, Uber is worth \$70+ billion, while also losing \$1.2 billion in the first half of this

Slick and cool as both of those are, they're just hacks on dispatch, which is a centralized intermediating system.

year (<http://www.bloomberg.com/news/articles/2016-08-25/uber-loses-at-least-1-2-billion-in-first-half-of-2016>)—nice work if you can get it.

It's easy to think of Uber as a new kind of company, but it's not. You can tell it's not when you ask yourself this question: *Why not hire a ride from anybody who has one, rather than just through Uber or Lyft?* Slick and cool as both of those are, they're just hacks on dispatch, which is a centralized intermediating system. The better hack will be one that connects demand with supply directly, on an individual basis. When you call for a ride, it should ring up everybody in a position to provide it, including Uber, Lyft, your neighborhood taxi and car services, and everyone else in the market for selling an open passenger seat.

How do we get there? Start with customers. Equip demand to drive supply directly. Equip everybody to participate. Not in a value chain, but rather in a value constellation, throughout which costs are reduced and benefits increased for everybody.

Look at it this way: the marketplace is a kernel space. Any of us can make it work better for all of us. ■

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

[RETURN TO CONTENTS](#)