LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

WATCH: ISSUE OVERVIEW

OCTOBER 2016 | ISSUE 270
http://www.linuxjournal.com

Fixing the Network Time Protocol

+

Simple Steps for Hardening Your Server

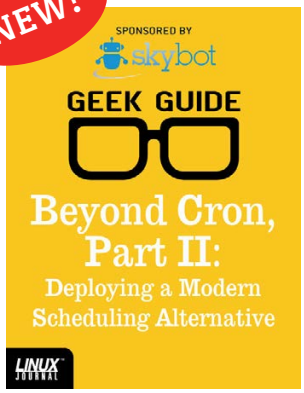
The Importance of Machine Learning

Shell Scripting a Mars Lander

EOF: a New Networking Model

Flat File Encryption with **OpenSSL** and **GPG**

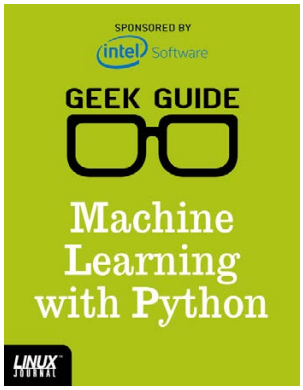
NEW!



Beyond Cron, Part II: Deploying a Modern Scheduling Alternative

Author: Mike Diehl

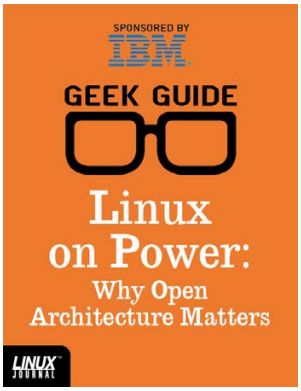
Sponsor: Skybot



Machine Learning with Python

Author: Reuven M. Lerner

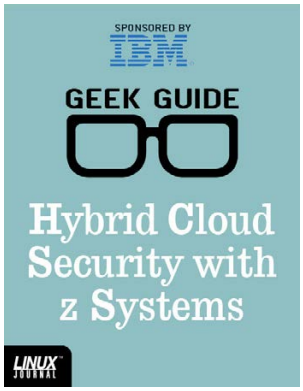
Sponsor: Intel



Linux on Power: Why Open Architecture Matters

Author: Ted Schmidt

Sponsor: IBM



Hybrid Cloud Security with z Systems

Author: Petros Koutoupis

Sponsor: IBM

LinuxONE: the Ubuntu Monster

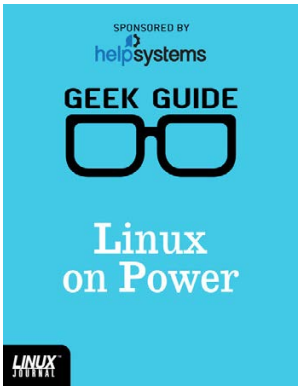
Author: John S. Tonello

Sponsor: IBM

Ceph: Open-Source SDS

Author: Ted Schmidt

Sponsor: SUSE



Linux on Power

Author: Ted Schmidt

Sponsor: HelpSystems

SSH: a Modern Lock for Your Server?

Author: Federico Kereki

Sponsor: Fox Technologies

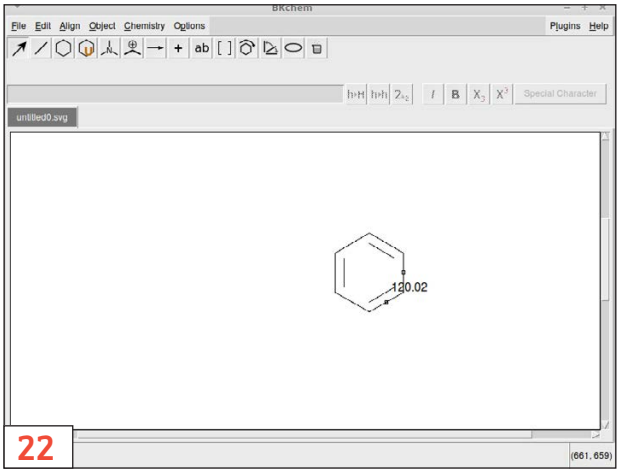
CONTENTS

OCTOBER 2016
ISSUE 270

FEATURES

COLUMNS

IN EVERY ISSUE



22


52

ON THE COVER

- **Fixing the Network Time Protocol, p. 68**
- **Simple Steps for Hardening Your Server, p. 46**
- **The Importance of Machine Learning, p. 34**
- **Shell Scripting a Mars Lander, p. 40**
- **EOF: a New Networking Model, p. 106**
- **Flat File Encryption with OpenSSL and GPG, p. 80**

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Belltown Media, Inc., PO Box 980985, Houston, TX 77098 USA.
Subscription rate is $29.50/year. Subscriptions start with the next issue.

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

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

Out with the New, and in with the Newer!

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.

There was a show a few years back called, "Extreme Makeover: Home Edition". The premise of the show was to find families who needed their houses overhauled, but couldn't afford to do it on their own. Generally, those chosen had sacrificed for others rather than spend time and money on themselves. Then the show would completely redo their houses, making it so nice the happy families no longer could afford the taxes, and they'd soon be homeless. I might have missed the point of the show, but the idea of improving on outdated infrastructure certainly rings true for IT folks. This month, we look at improving our lives by improving on the tech we depend on every day.

Reuven M. Lerner starts out by teaching how to create Skynet and turn civilization over to robotic overlords. More specifically, he discusses machine learning. Thankfully, it's not all laser guns and extermination, but rather a great way to get computers to make smart decisions instead

VIDEO: Shawn Powers runs through the latest issue.

of just crunching numbers. The notion of machine learning doesn't have to be science-fiction dystopia, and Reuven shows how it can be a huge benefit. Dave Taylor follows with part two of his Mars landing simulation. I assume it's a coincidence that Dave is teaching how to colonize another planet the same month Reuven is teaching how to make thinking machines. Either way, both columns are very educational!

Kyle Rankin explores how to defend against attack by hardening your servers. The idea of server hardening has been around for so long, much of the information on the internet is outdated. Kyle walks through some simple, practical procedures for making sure your servers are as secure as possible. If you have any servers exposed to the internet, or even an untrusted intranet, you owe it to yourself and your company to read Kyle's column this month. As for me on the other hand, I couldn't decide what to write about, so I just wrote about all the various topics that I couldn't decide between. My day-to-day life is pretty nerdy, so hopefully some of my stream-of-consciousness mashup will be of use. I tend to get excited about the things I love, and I sure do love technology!

NTP is a service that has been around for a long time, and most of us just install it without thinking twice. Eric S. Raymond covers NTPsec this month, which is a huge overhaul to the NTP system we know so well. If your experience with NTP ends with `sudo apt-get install ntp`, you should really read his article and consider NTPsec. Charles Fisher follows Eric with a look at encryption—specifically, OpenSSL and GPG encryption with flat files. The concept of encrypting files isn't new, but Charles will force you to look at the idea in a different light. Plus, he includes lots of code examples, which always helps me understand things.

Whether you want to improve your old technology with new or just improve your existing tech, this issue should be fun. We have all the features you've come to expect, including product reviews, announcements, tech tips and so on. The best part about improving your infrastructure with Linux is that unlike "Extreme Makeover", it will save you problems in the future instead of causing more! So without further ado, "Driver, Move That Bus!" ■

PREVIOUS
Current_Issue.tar.gz

NEXT
UpFront

Self-Sovereign Identity

I always read Doc Searls' column, not to learn anything new but to try to figure out what his point is. In the July 2016 issue, he introduces the concept of "self-sovereign identity" with several obscure sentences, and then, to take fuzzy thinking to a higher level, he quotes Devon Loffreto with a paragraph of absolute gibberish. Here's one excerpt:

> A self-Sovereign identity produces an administrative trail of data relations that begin and resolve to individual humans.

And another:

> A self-Sovereign identity is the root of all participation as a valued social being within human societies of any type.

Now that's gibberish.

It's ironic that this column appears in a magazine that is so informative otherwise.

—Phil Miller

Doc Searls replies: *Phil, I try to bring up subjects, and make points, that nobody else does. If some of that effort comes off as gibberish, at least it beats silence.*

Lots of original thinkers and authorities on topics don't make full sense.

But that doesn't mean what they say isn't worth listening to, or de-bugging. That's why I followed the Devon Loffreto quote by compressing his point down to "only the individual has root for his or her own source identity".

In a world where surveillance is the norm, I believe that insight can help guide some necessary work. That's why I wrote this piece.

Tiny Internet—Test for CPU Extensions for Virtualization

I'm a bit behind in my reading, so I'm not sure if anyone else has commented on this. In the May 2016 issue of *LJ* in John S. Tonello's "The Tiny Internet Project, Part I", he provides instructions on seeing if the computer can support virtualization, but he tests only for Intel's VT. Some individuals new to this may have AMD-based systems (especially if they are using older hardware, as AMD is not nearly as popular as it once was). Anyway, it's always better not to assume what the CPU is and do: `egrep '(vmx|svm)' /proc/cpuinfo` and instruct users to make sure they have *either* vmx or svm in the output. Also, don't forget that, again when using older PCs, some may not have 64-bit capability so checking for "lm" in the output also is important, and if it isn't present, to use 32-bit distros. And finally, some virtualization platforms require Execute Disable to be enabled, so users should check for *either* XD or NX in the output of /etc/cpuinfo.

—Mark Dean

John S. Tonello replies: *You're absolutely right about AMD. Many of my early machines were*

AMD-powered, albeit well before 64-bit was widely available. Your tips are great for anyone looking to test their hardware before proceeding with building a "Tiny Internet". Thank you for sharing!

Photo of the Month

This drawing of Tux was a birthday gift to me from my five-year-old daughter.
—big.foot

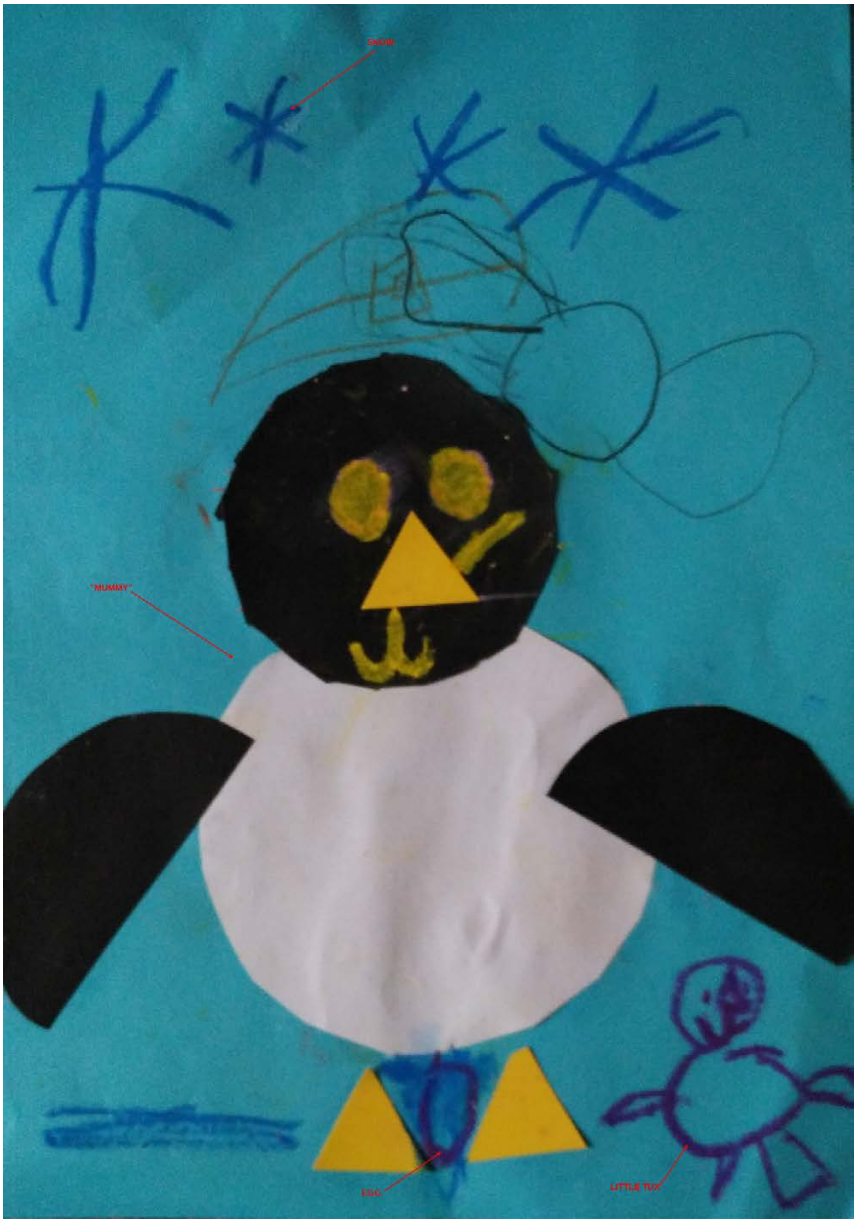


PHOTO OF THE MONTH

Remember, send your Linux-related photos to ljeditor@linuxjournal.com!

WRITE LJ A LETTER

We love hearing from our readers. Please send us your comments and feedback via http://www.linuxjournal.com/contact.

RETURN TO CONTENTS

O'REILLY®

O'Reilly Live Training

Online or in person, but always in real time.

Get intensive, hands-on training on current critical technology topics, led by instructors from O'Reilly's unparalleled network of tech innovators and expert practitioners.

See the list of courses

PREVIOUS
Letters

NEXT
Editors' Choice

diff -u

What's New in Kernel Development

Kan Liang recently tried to get some networking configuration code into the kernel that would interpret a simple set of user policies in order to tweak the networking code in subtle ways to speed up data transfers as much as possible.

Unfortunately, although his ideas seemed sound to folks like **Stephen Hemminger** and **Alexei Starovoitov**, they both objected to including code in the kernel that wasn't strictly necessary. They felt the same features could be achieved entirely in user space.

Kan's argument that a user-space implementation would be more complex and difficult fell on deaf ears. The same argument has been used many times before, but the kernel folks have to take a hard line on the issue or risk the kernel being overrun with bloat. Some even would argue that this already has happened.

Because of this, unless Kan finds a better argument, it doesn't seem likely that his code will get into the kernel, although it could very well become a tightly integrated user-space tool.

William C. Roberts recently posted some code to randomize the

locations of new memory allocations. This is a standard security technique, but William wanted to apply it by default to all mmapped memory regions.

Various folks objected that **Android** had experienced particular problems with this sort of thing in the past, as it caused extreme memory fragmentation that inevitably would lead to the failure of all attempted memory requests.

The solution, as suggested by **Dave Hansen**, turned out to be simply to disable William's patch on 32-bit systems. Once this idea was presented, everyone immediately agreed that it would solve the problem. Even William liked it.

Presto—an extremely rare case of a security issue having a simple, clean solution that everyone agrees on.

Luis R. Rodriguez and others have been simplifying the very complex Linux boot procedure, in part by removing support for early access to device firmware. Their hope was that kernel devices could access firmware at a slightly later time in the boot process, after the firmware could be made available on a mounted filesystem.

As it turned out, however, there were more pieces of code and more use cases, such as embedded systems, relying on early access to firmware than Luis had realized. After some discussion, it became clear that support for a few remaining users of early firmware access would have to remain in the kernel, at least for now, and that a more gradual approach to rooting out the remaining users would have to be taken.

Rafael J. Wysocki recently proposed a new kind of runtime driver dependency, in which a given driver could be loaded only if the drivers it depends on are also loaded, and may be unloaded only if no other drivers depend upon it.

It turns out there are some nuances to get right before something like this really could be accomplished. For one thing, the code to implement dependencies might look awfully similar to the existing code to probe for resources before loading a given driver. It would be important to avoid too much code duplication, which might

require refactoring that entire area of the kernel source.

There's also the question of when a dependency might be identified. For some drivers, certain dependencies would be clearly known and could be registered in a configuration file. For others, a dependency would be based on which other drivers already had been loaded and could provide certain resources, so there would have to be at least two phases of dependency identification.

Some dependencies also might be "weak"—useful if present, but not absolutely needed for normal operations.

After some discussion, Rafael posted some code implementing the beginnings of his idea. There seems to be general approval of the overall concept. The only issues are exactly how to support the various features and how to avoid too much complexity in the implementation.—Zack Brown

THEY SAID IT

One must desire something to be alive.
—Margaret Deland

Confidence is 10% hard work and 90% delusion.
—Tina Fey

I must create a system or be enslaved by another man's.
—William Blake

To live a creative life, we must lose our fear of being wrong.
—Joseph Chilton Pearce

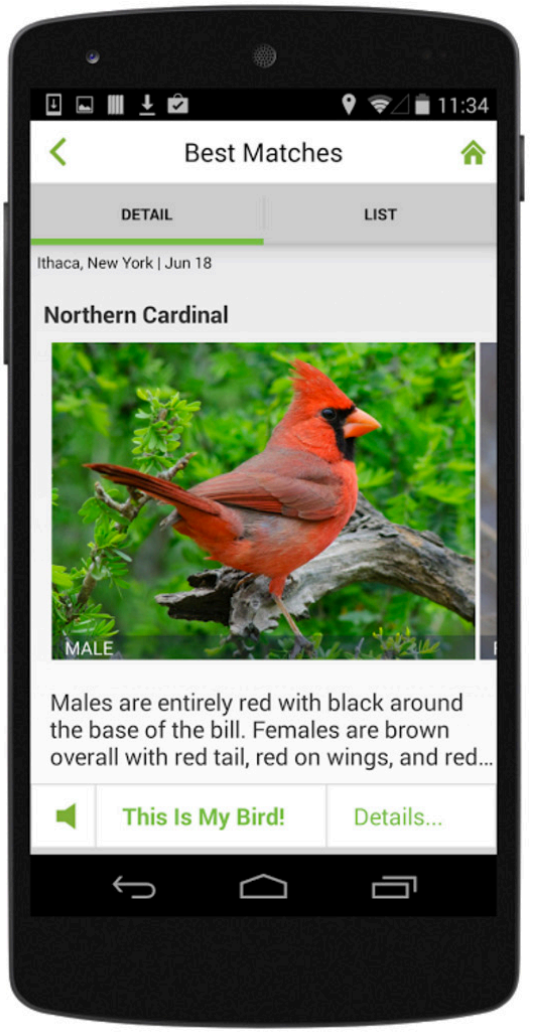
There's a place in the brain for knowing what cannot be remembered.
—John Green

Android Candy: That App Is for the Birds!

Usually bird-related apps involve pigs and anger, but if you're a bird watcher like myself, there's another bird app you must download. Cornell Labs has released a free app called Merlin Bird ID that helps identify birds you see in the wild.

Hundreds of books are available to help you figure out what sort of bird you're looking at, but Merlin uses several metrics to narrow down the feathery mystery. It uses:

- Size.
- Color(s).
- Geographic location (via GPS).
- Time of year.
- Environment (on tree, in water, on ground and so on).



(Image from Google Play Store)

Once it narrows down the options to a handful of birds, it provides photos of the male, female and juvenile varieties. It even includes a button that lets you listen to their particular birdsong.

If you're a bird-lover, or just like to sound smart in front of your friends, the Merlin Bird ID app is a must-have. It's completely free and remarkably accurate. Find it on the Google Play store today!—Shawn Powers

JOIN 2,000+ OPEN SOURCE TECHNOLOGISTS AND DECISION MAKERS FROM ALL OVER THE WORLD

ALL THINGS OPEN®

2016

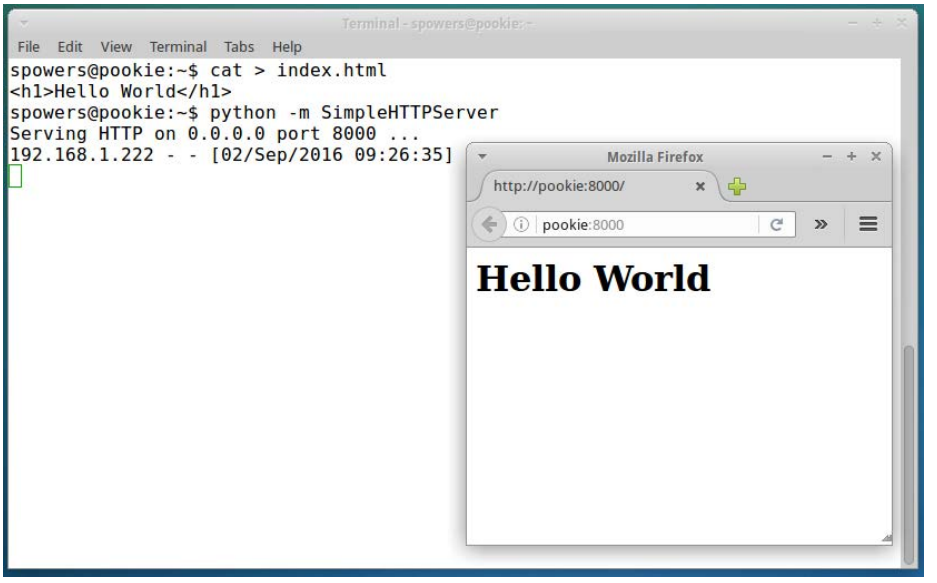
OCTOBER 26 & 27 | DOWNTOWN RALEIGH

THE 2016 EVENT WILL FEATURE:

- Nearly every major technology company in the U.S.
- More than 150 speakers and 180 sessions
- Some of the most well known speakers in the world
- 10 news-making keynotes
- 37 tracks over both days on nearly every “open” topic

ALLTHINGSOPEN.ORG

On-the-Fly Web Server



Most of you have a web server installed on your network somewhere. In fact, most of you probably have several. In a pinch, however, getting to the web directory can be difficult. Thankfully, there's a super-simple, incredibly awesome one-liner you can type to get a functional web server running and serving out your current directory:

```
python -m SimpleHTTPServer
```

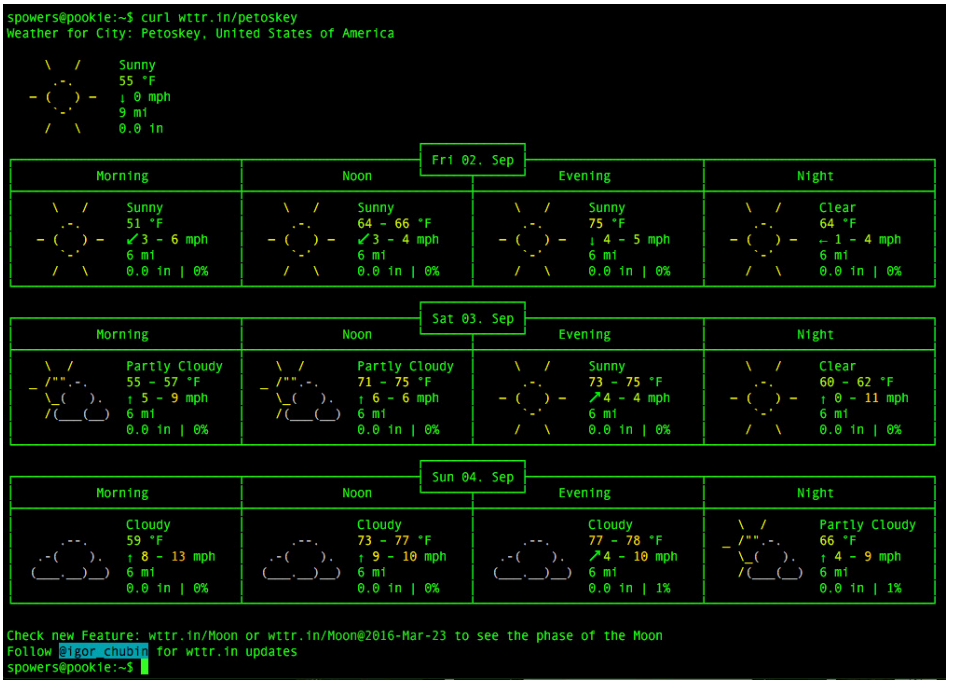
That one-liner (or the Python 3 alternative, `python -m http.server`) will start a web server on port 8000, serving files from your current directory. Usually pressing ^C will stop the server, but if not, some more command-line fu will stop the process as well:

```
kill `ps | grep SimpleHTTP | grep -v grep | awk '{print $1}'`
```

It's possible to change the port by adding it after the `SimpleHTTPServer`, but since you're running as a user, you won't be able to run on a privileged port.

(Thanks to jafraldo on http://www.commandlinefu.com for the kill script.)—Shawn Powers

Sunshine in a Room with No Windows



I'm a bit of a weather nut. It might be because I'm getting older, but for some reason, the weather fascinates me. I'm not quite to the point that I watch The Weather Channel on a regular basis, but I do check the forecast often.

I also spend the vast majority of my day in a terminal window. Until recently, if I wanted to check the weather, I had to open a browser and click a link in order to get the forecast. Thanks to Igor Chubin (@igor_chubin on Twitter), I now can get the forecast from the comfort of my terminal window. All you need to do is type:

```
curl wttr.in/your_town
```

and you'll get a nice text-based graphical forecast. You also can view the page in a web browser if you prefer (it looks cool there too), but being able to whip up a forecast on the command line is just awesome. I've tried using city names and zip codes, and both seem to work well. If you want to know what the weather is like, but don't want to open a window, give it a try!—Shawn Powers

Chemistry on the Desktop

For this article, I thought I'd introduce another chemistry application—specifically, BKChem, a free chemical drawing program. As opposed to many other chemistry applications, BKChem provides both a nice GUI for constructing molecules and a set of chemical analysis tools to look at the properties of the newly constructed molecule.

Most distributions should have a package available to make installation easier—for example, Debian-based distributions can install BKChem with

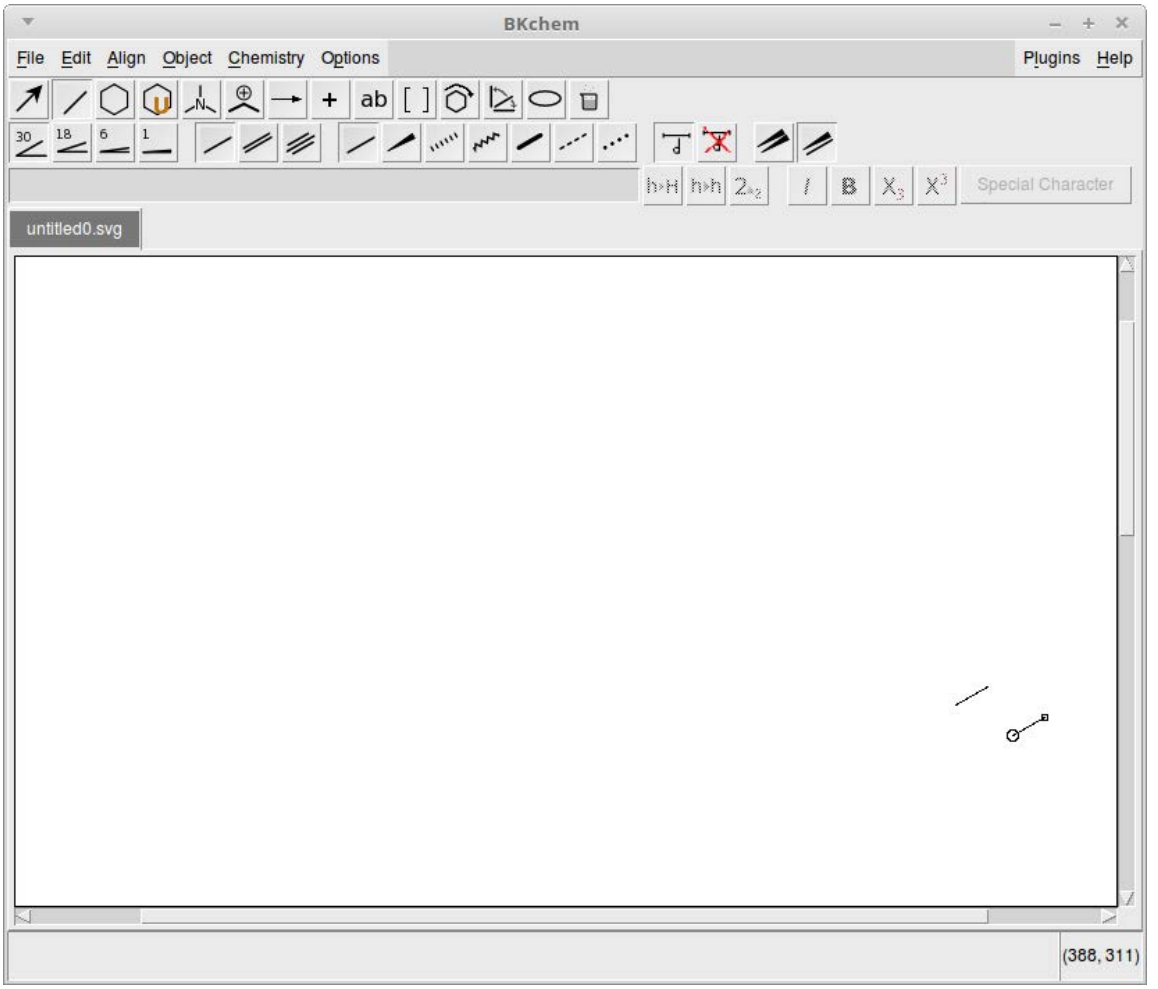


Figure 1. When you first start BKChem, you get a blank canvas to start building your molecule.

the following command:

```
sudo apt-get install bkchem
```

Once BKChem is installed, you can start it either from the menu entry or by executing the command `bkchem` from a terminal window.

When it first opens, you'll see a blank screen where you can start your chemical construction.

If you have a previously created molecule, you can load it by clicking the File→Load menu item, which will load the data into a new tab, or you can click the File→Load to the same tab menu option to load it into the currently active tab.

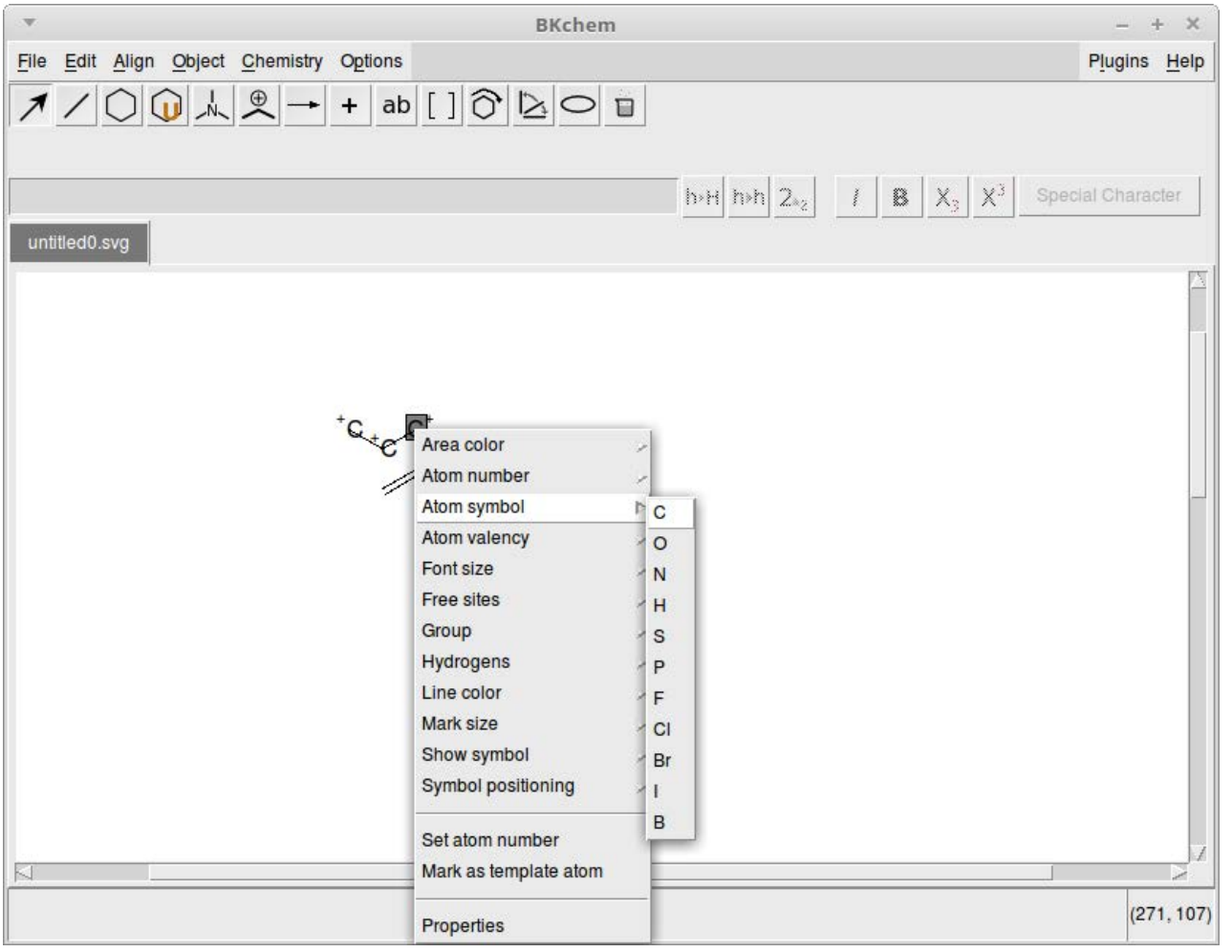


Figure 2. You can edit an element by clicking the middle mouse button. The menu you get depends on the type of element you are editing.

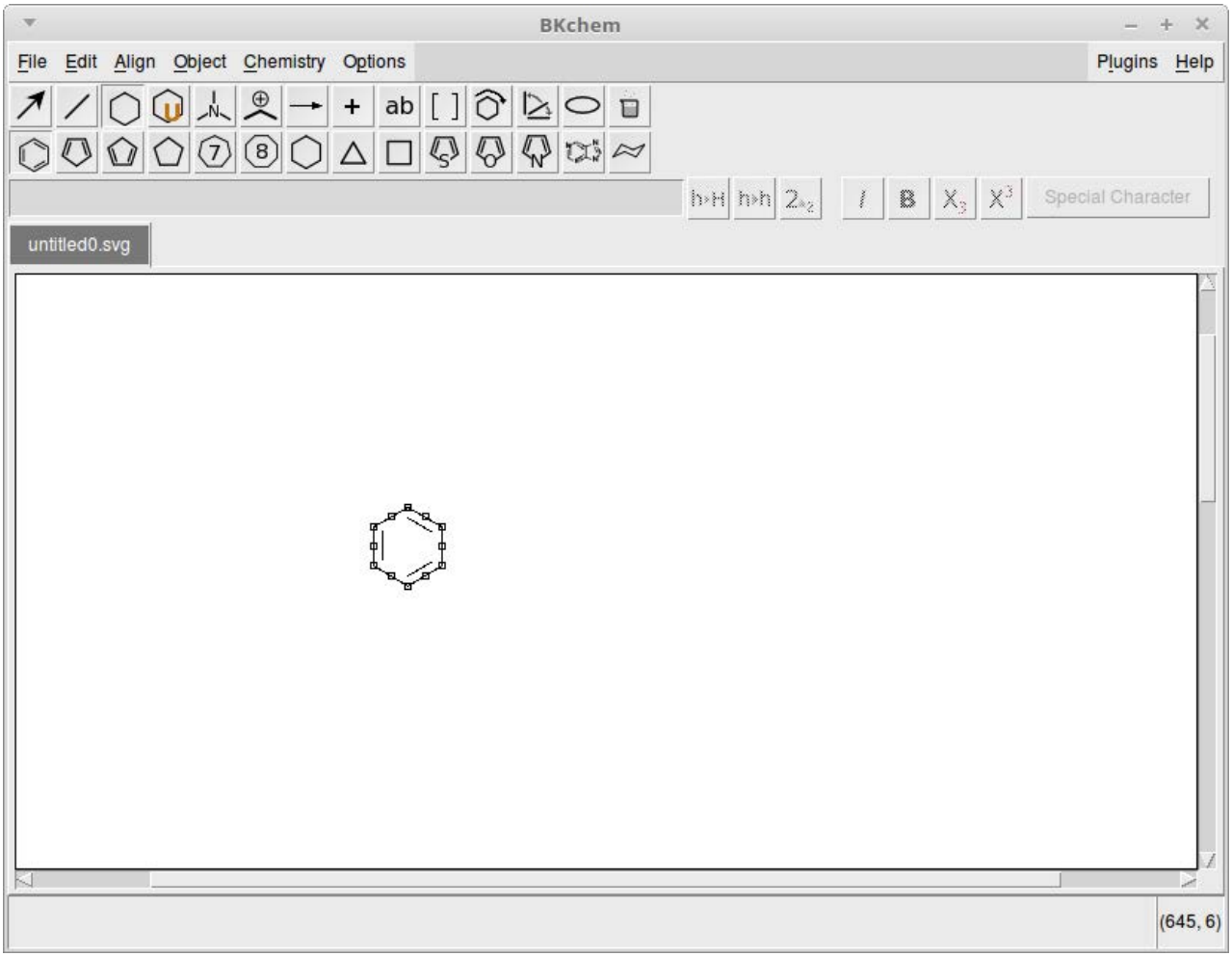


Figure 3. You can add larger commonly used structures with templates.

BKChem also can import data from other file formats. If you click File→Import, you'll see that you can import files with CML, CML2 or Molfile formats.

If you want to start by building your own molecule, several menus of building blocks are available. They are laid out as a pair of rows, just below the menu listings at the top of the window. The top row of icons selects which list of icons will be available in the second row. The first icon in the first row is simply an arrow, allowing you to select objects within your molecule so you can edit their properties. The next icon pulls up the row of drawing elements where you can start to draw your new molecule.

There are several choices in terms of line thicknesses, styles and bond angles, and you can create a chain of elements simply by clicking on the end of an existing line segment.

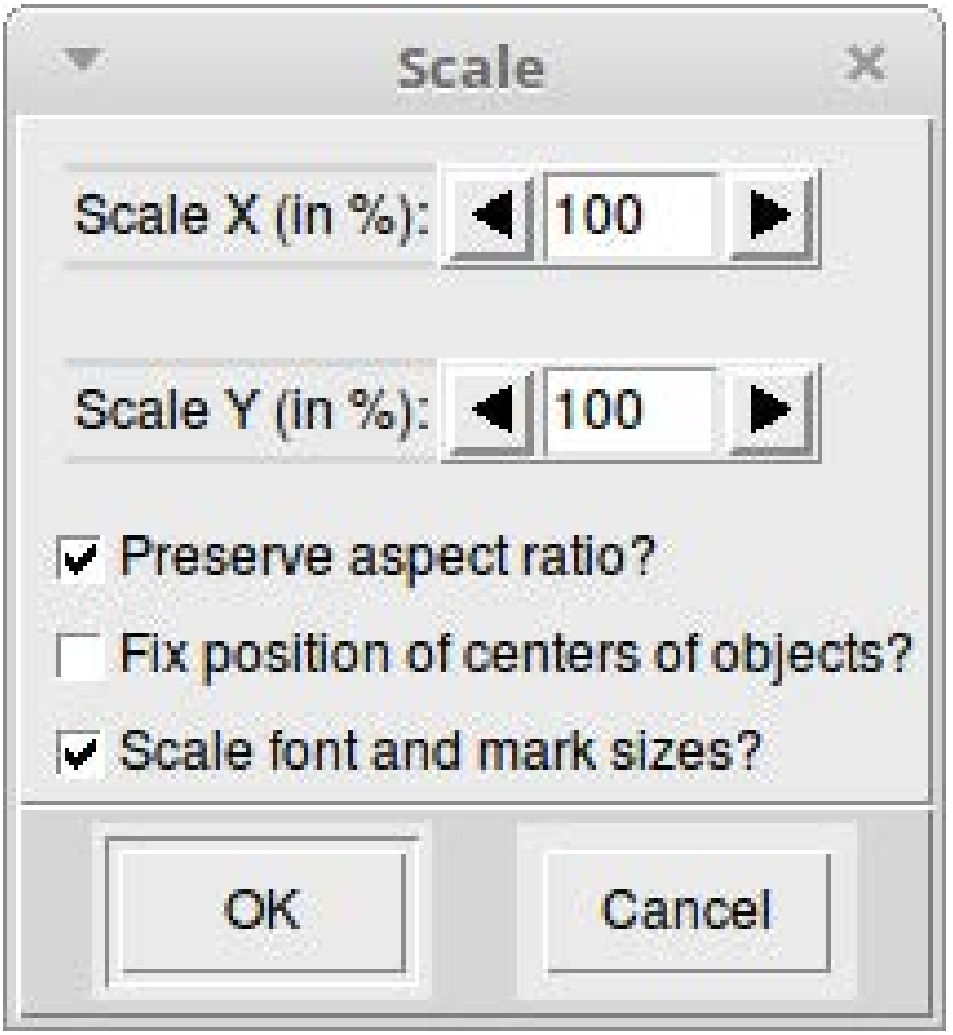


Figure 4. You can scale parts of your molecular structure to make it easier to work on.

Once you have the basics of your structure laid out, you'll want to edit the details next. To do this, click on the first icon again (captioned with "edit"), and then click on the structure element you want to edit. This is where having a proper mouse is a must, as you need to click with the middle button on your mouse to pull up the edit panel.

If you are using a laptop touch panel, you need to click the left and right buttons together and then scroll with the touch pad itself. As an example, if you click on an atom, you can change the atom in this location or even replace it with some type of atomic group, such as an alcohol group.

You also can edit all kinds of display options, such as colors used, fonts for text, text placement and line widths.

The third icon on the top icon row pulls up a list of available templates for larger commonly used atomic structures, such as benzene rings.

The fourth icon on the top row lets you pull up a list of templates that

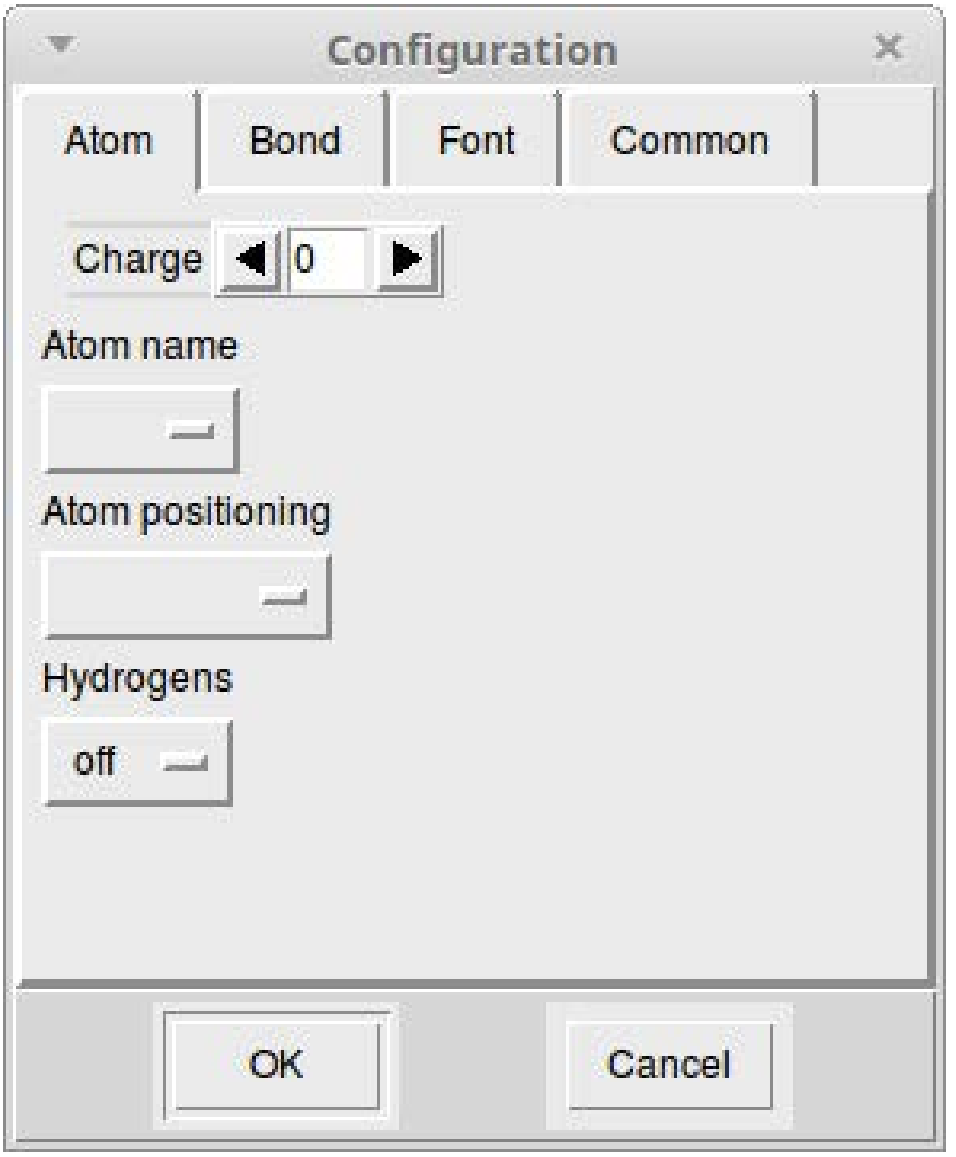


Figure 5. A configuration window allows you to edit properties of your structure's elements easily.

you have created previously, allowing you to add templates for those substructures that you use most often.

You may need to play with the display in order to be able to see everything clearly while you are working. The appropriate instructions are under the Object menu item. The first option on that menu is Scale. Selecting either a portion of your structure or the whole thing, you essentially can zoom in or out to see the structure better as you work on it.

You also can make changes to the display, such as altering the stack layers or mirroring horizontally or vertically.

The last option available is the Object→Configure menu item. This will

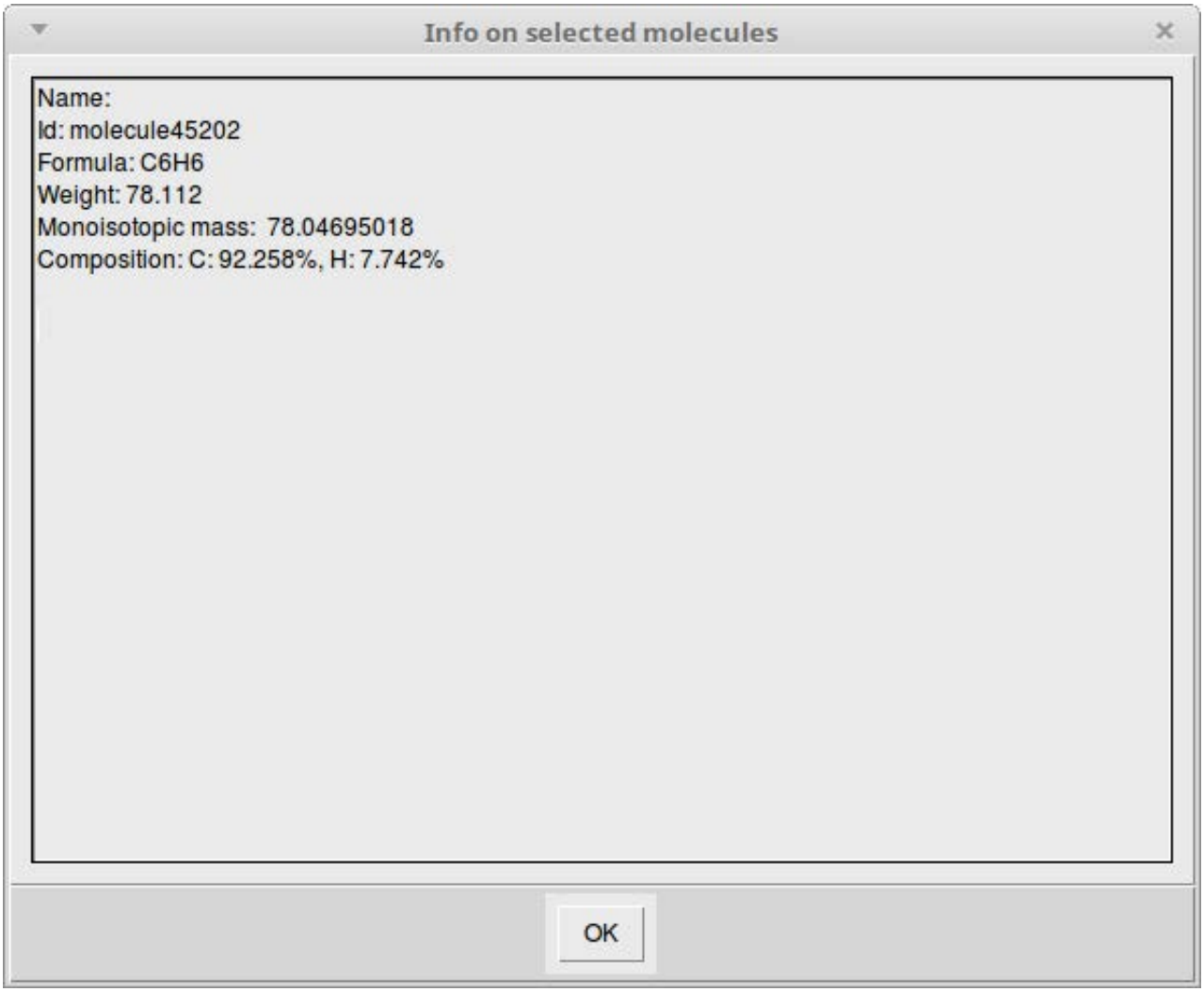


Figure 6. You can get a list of basic chemical information about your structure.

pop up another window where you easily can change atom, bond, font or common details for the display. This is also the same window that appears when you click on the Properties item on the Edit menu (which you get when clicking the middle mouse button).

So far, I've just been describing building up a molecule. In the rest of the screenshots here I'm using a simple benzene ring to discuss the chemistry you can do with BKChem.

You can get basic information about your structure by clicking on Chemistry→Info, which will pop up a new window with items like the chemical formula, molecular weight and composition.

If you want to check to see whether the structure you've built makes

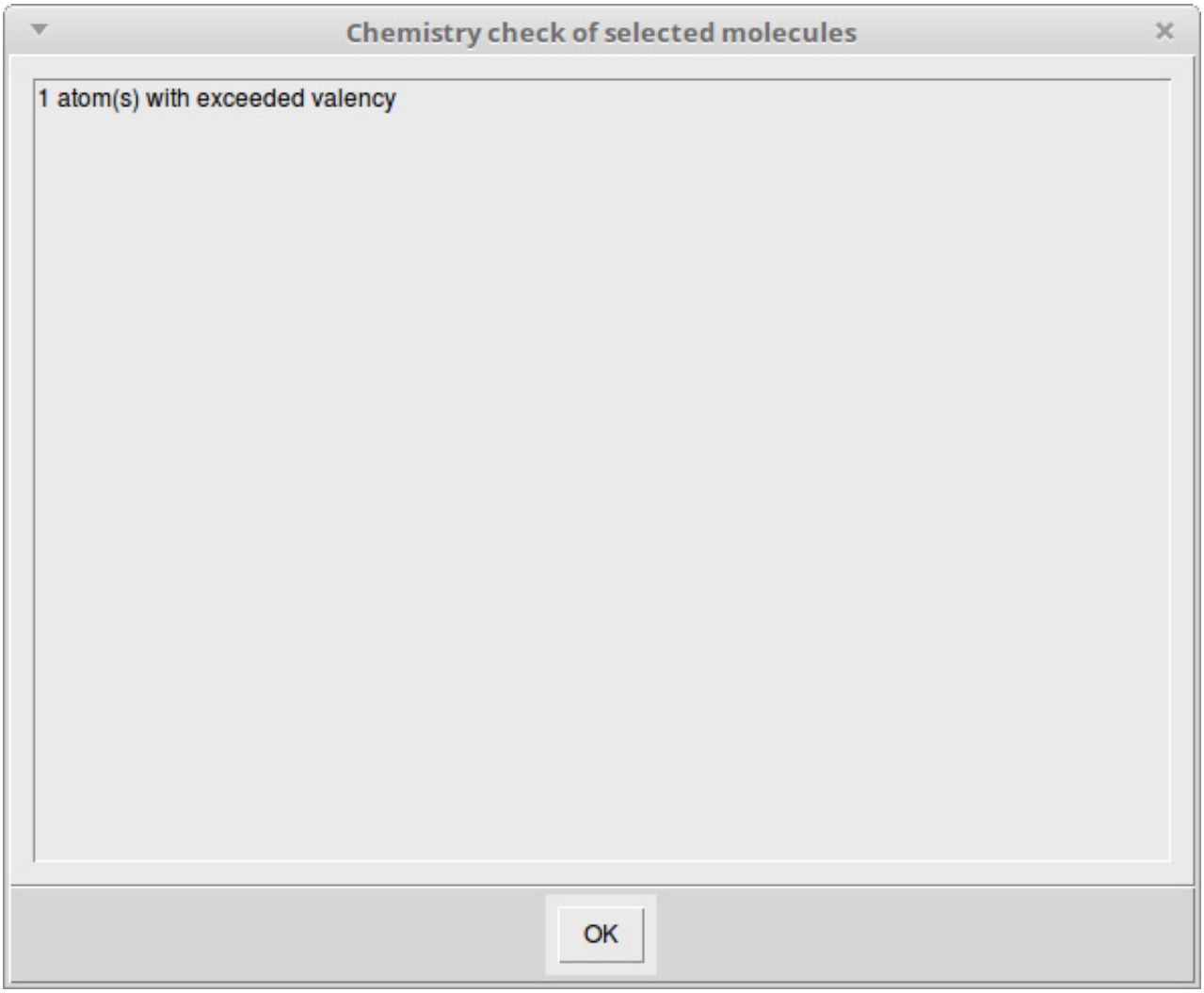


Figure 7. Sometimes, the chemistry doesn't work out.

sense, you can select it and click on the Chemistry→Check chemistry menu item. If everything makes sense, you'll see a new window telling you that everything is okay. Otherwise, you'll get an error message highlighting what doesn't quite work from a chemical point of view. This is where you may need to dig into the grittier items available from the Edit menu when you click on an element with the middle mouse button. You may need to change the type of atom or change its valency. As an example of the type of error you might encounter, see what happened when I changed one of the carbon atoms to an oxygen atom in the benzene ring (Figure 7).

At the far right side of the menu bar, there is an entry for available

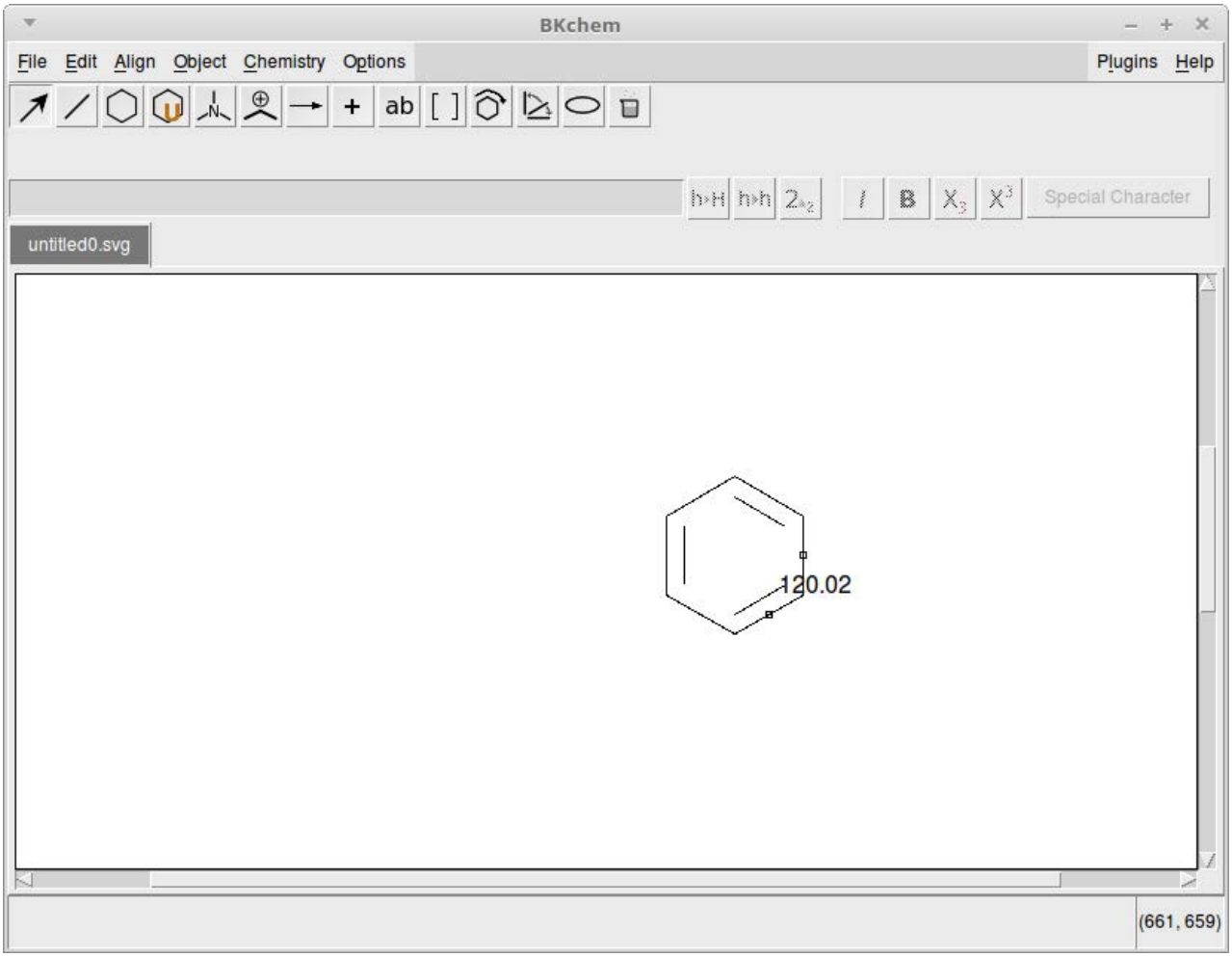


Figure 8. You can calculate bond angles for your molecule.

plugins. One of these plugins allows you to calculate bond angles. You can select two connected bonds by holding down the Shift key and clicking them one after the other. Then you can click on the Plugins→Angle between bonds menu item to calculate the angle.

When you've finished all of the work of creating your new molecular structure, you'll want to save it for further analysis or to share with other researchers. The default file format that structures are saved in is an SVG (Scalable Vector Graphics) file. If you want to save your work using a different file format, click on File→Export to get a list of all the supported file formats. Most of them are simply other graphics file formats, but a few are specifically used for storing chemical information.

You can select CML or CML2 (Chemical Markup Language) to save

more of the chemical information for your structure. You also can save your structure in the molfile file format, which was created by MDL Information Systems to store more detailed information about a chemical structure. If you just want an easily shared image of your molecular structure, you can export it into either a PDF or PNG file.

As you have seen here, you can use BKChem to build molecular structures in a way similar to many other chemistry applications. What is slightly different with BKChem is that you can do some basic chemistry with your newly created structure directly from BKChem. This added functionality might be enough to make BKChem a tool you'll want to add to your arsenal.—Joey Bernard

PREVIOUS
UpFront

NEXT
Reuven M. Lerner's
At the Forge

Non-Linux FOSS: Don't Drink the Apple Kool-Aid; Brew Your Own!

Some tools that I use on the command line are so basic and so ingrained in my day-to-day actions that it's weird when they're not available. This often happens to me on OS X. I love that OS X has UNIX underpinnings. I love that the terminal window is a *real* terminal window and works like a terminal window should work.

But, I don't like the lack of basic tools that are available. Even more, I dislike that I can't simply `apt-get` (or `yum`) the missing applications into my system. Thankfully, I'm not alone. If you've ever opened a terminal window on OS X and tried to use `wget`, you know the feeling too. Enter: Homebrew.

The concept of Homebrew has been around for quite a while. Fink, MacPorts and probably a few others give you the options to install software that isn't part of OS X itself. Those other options make me uncomfortable, however, because they're more integrated into the

(Image from http://brew.sh)

UNIX ports system. It might seem odd that better integration makes me uncomfortable, but it does. I'd rather have something I can erase easily without affecting the rest of the system. I'd rather have the non-native applications live separately from the native apps. Part of that is personal preference, but part of it is based on years of experience with troubleshooting problems. If there's a problem with Homebrew, it's much easier to troubleshoot.

Anyway, if you'd like to have a package manager for some of the "missing" programs in OS X, but you'd also like to be able to delete the entire system fairly simply, Homebrew is perfect for you. Simply head over to http://brew.sh (cool URL, right?) and paste the installation script into a terminal window. There's an un-install script as well, or you simply can delete the bits manually. Even if you're just curious, it's easy to install Homebrew and give it a go. In fact, thanks to its simple install/un-install and the incredible convenience of having common tools available on OS X, Homebrew gets this month's Editors' Choice award. If you have a Mac, check it out.—Shawn Powers

RETURN TO CONTENTS

Machine Learning Everywhere

You've probably already heard of machine learning, but read on for some examples of why you should learn it and how it can help you.

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

THE FIELD OF STATISTICS TYPICALLY HAS HAD A BAD REPUTATION. It's seen as difficult, boring and even a bit useless. Many of my friends had to take statistics courses in graduate school, so that they could analyze and report on their research. To many of them, the classes were a form of nerdy, boring torture.

Maybe it's just me, but after I took those courses, I felt like I was seeing the world through new eyes. Suddenly, I could better understand the world around me. Newspaper articles about the government and scientific and corporate reports made more sense. I also could identify the flaws in such reports more easily and criticize them from a position of understanding.

Much of the power of statistics lies in the creation of a "model", or a mathematical description of reality. A model is a caricature of sorts, in that it doesn't represent all of reality, but rather just those factors that you think will affect the thing you're trying to understand. A model lets you say that given inputs A, B, C and D, you can know, more or less, what the output will be.

Sometimes, the goal of a statistical model is to predict a value—for example, given a certain size and neighborhood, you can predict the price of a house. Or, given someone's age, weight and where they live, you can predict his or her likelihood of getting a certain disease.

Often, the goal is to predict a category—for example, in an upcoming election, for whom are people likely to vote? Taking into account where they live, what level of education they've received, their ethnic background and a few other factors, you can often predict for whom people will vote before they know it themselves.

During the past few years, there has been a huge amount of buzz around the terms "big data", "data science" and "machine learning". As these buzzwords continue to gain acceptance, many statisticians are wondering what the big deal is. And to be honest, their complaint makes some sense, given that "machine learning" is, more or less, a computerized version of the predictive models that statisticians have been creating for decades.

Now, why am I telling you this? Because I actually do believe that machine learning is a game-changer for huge parts of our lives. Just as my perspective was changed when I learned statistics, giving me tools to understand the world better, many businesses are having their perspectives changed, as they use machine learning to understand themselves better. Everything from online shopping, to the items you see in your social-network feeds, to the voice-recognition algorithms in your phone, to the fraud detection used by your credit-card company is being affected, boosted and (hopefully) improved via machine learning.

This means that no matter what sort of software development you do, you would be wise to gain as much experience as you can with machine learning. Its benefits might not be obvious to you at once or even be applicable to your work right away. But machine learning is becoming ubiquitous, and there is no shortage of ways in which to use it.

So with this article, I'm starting a series on machine learning and some of the ways your organization can take advantage of it. I'll look at a number of problems, many of which are common on web applications, that can benefit from using machine learning. Along the way, I hope you'll get lots of ideas for the sorts of analysis and uses that machine learning can bring to your applications.

If you're completely new to the world of machine learning, I encourage you to read the free Geek Guide I wrote on the subject, published by *Linux Journal* and available at http://geekguide.linuxjournal.com/content/machine-learning-python-0.

Uses for Machine Learning

If you have ever invested money, you'll undoubtedly remember that the fund in which you invested, or the broker with whom you worked, warned you that "past performance is no guarantee of future results" or words to that effect. That's because we, as living beings, are conditioned to assume that if the world worked a certain way in the past, then it'll likely work a certain way in the future. For most of us, most of the time, this is a good way to live our lives.

Machine learning works on this principle, that the past is a good indicator of the future. We create a machine-learning model, telling the model that given a set of inputs, we got a particular output. One such piece of information is unlikely to give us anything useful. But, several hundred samples later, the model can start to make some predictions. Several thousand, or even million, samples later, and your predictions can potentially be quite accurate.

Customer Patterns

If you run an online store, machine learning can help you to understand your customers better. For example, if you know that customers who bought products X and Y also bought product Z, you can send email promotions to such people who haven't yet bought Z, knowing that a proportion of them will respond positively.

More nefariously, you also could raise the price on product Z when those people visit your site, knowing that a fair number of such people are likely to buy it anyway.

(And by how much should you raise the price? Assuming enough traffic, you can try different numbers on different people, until you figure out the optimal setting.)

If you have additional information about your customers, such as their age, gender or where they live, a machine-learning algorithm can help you determine even more about them—from what they're likely to buy, to how often they'll visit your store. You also can keep track of things they thought about buying, but later removed from their shopping carts.

Take the famous story of Target, which sent a "so you're expecting" promotion to a teenage girl in the United States. It turns out that Target's machine-learning systems had correctly identified that based on her purchasing habits, she was likely to be pregnant. The only problem was that this teenager in question, who was indeed pregnant, hadn't told her parents.

I recently spoke with the CTO of a new online marketplace for a specific type of consulting, in which customers and consultants would communicate, with a goal toward solving the problem. After each session, each of the participants would then indicate how satisfied they were. The CTO wanted to know where machine learning could help; I told him that over time, they could accumulate a huge amount of data regarding which types of customers got along best with which sorts of consultants—allowing them to make increasingly good recommendations and be better matchmakers.

And speaking of "matchmakers", every modern online dating site uses machine learning. They know a lot about their users, and they use that data to try to predict which people in their database are likely to be the greatest success. You could say that these dating sites, thanks to their machine-learning systems, know more about people's dating preferences than people could explicitly say about themselves.

Recommendation Engines

One classic example of machine learning is a recommendation engine. I have been shopping on Amazon since it first opened, so I've provided that company with a great deal of data about myself and the things I like to purchase. When it suggests that I might be interested in a product, the odds are good that I either have it already, or that I considered it or that I would indeed be interested in it.

If you run an e-commerce site, you can use machine learning to similar

ends. You can create a model that identifies which products are similar to which other products. Then, you can go through someone's purchase history, finding unpurchased products that are similar to the ones that they already have bought.

Nowadays, sites like Amazon often have access not only to your purchase history, but also to the ratings you gave to various products. In this way, sites can suggest not only the products that you're likely to buy, but also those that you're most likely to enjoy as well.

Another way to handle recommendations is to look at people, rather than products. Instead of telling me what I'm likely to buy based on my past purchases, a site could tell me what I'm likely to buy, based on my friends' purchasing habits. If you have access to friends' recommendations, the combination of a friend's purchase and a high rating from that friend might make a product especially attractive—and, thus, the object of a special promotion.

Finances

Of course, many of the heaviest and largest users of machine learning are financial firms. You can be sure that credit-card companies and other payment companies, such as PayPal, spend a great deal of time and effort on machine-learning algorithms that identify when someone might be committing fraud. If I use my credit card to buy something unusual or in a country I haven't visited before, my credit-card company sometimes will contact me to make sure the transaction is legitimate.

As you can imagine, such machine-learning models take a wide variety of inputs to try to determine whether the transaction is legitimate or fraudulent. These checks often have to take place in real time, which makes them particularly impressive in my book, given the amount of data that they have to deal with in a given moment. But, there inevitably will be some false positives and false negatives in such a system. If you're like me, the false negatives continue to haunt you for a long time afterward. After all, although I understand how these companies are just trying to do their jobs, it's maddening to be stuck at the supermarket checkout, trying to convince your credit-card company that you're simply trying to buy milk and bread in Stockholm.

"FinTech" companies are popping up all over, and much of what they

do is an application of machine learning to finance (hence the name, of course). Whether it's identifying fraud or looking for investment opportunities, such models can crunch more data more quickly than people—and can draw upon millions of previous examples, rather than the dozens that humans can keep in their heads.

Summary

As you can see, machine learning offers an incredible variety of solutions, providing opportunities for new types of analysis. It's definitely worth sitting down to learn something about machine learning. In upcoming articles, I plan to walk you through how to solve some of the problems described in this column using open-source languages and tools. I hope you'll soon see that in data science and machine learning, the coding is the easy part. The hard part is thinking about how to build your model, what to include and which algorithm would be the most appropriate.■

RESOURCES

Machine learning is a huge field, and part of the problem is the plethora of sources of information.

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 I 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.

Finally, 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://machinelearningmastery.com), where he sells a number of short, dense, but high-quality ebooks on these subjects.

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

RETURN TO CONTENTS

Mars Lander, Take II: Crashing onto the Surface

Dave succeeds at crashing the lander on the Martian surface—and says it's progress!

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 ARTICLE, I SPENT ALMOST THE ENTIRE PIECE EXPLORING GRAVITATIONAL PHYSICS, OF ALL UNLIKELY TOPICS. The focus was on writing a version of the classic arcade game *Lunar Lander*, but this time, it would be landing a craft on the red planet Mars rather than that pockmarked lump of rock orbiting the Earth.

Being a shell script, however, it was all about the physics, not about the UI, because vector graphics are a bit tricky to accomplish within Bourne Shell—to say the least!

Landing on the Earth's surface has lots more complexity with atmospheric drag and weather effects, but looking at Mars, and not during its glory days as Barsoom, it's atmosphere-free.

To make the solution a few dozen lines instead of a few thousand, I simplify the problem to two dimensions and assume safe, flat landing spaces. Then it's a question of forward velocity, which is easy to calculate, and downward velocity, which is tricky because it has the constant pull of gravity as you fire your retro rockets to compensate and thereby avoid crashing onto the planet's surface.

If one were working with Space X or NASA, there would be lots of factors to take into account with a real Martian lander, notably the mass of the spacecraft: as it burns fuel, the mass decreases, a nuance that the gravitational calculations can't ignore.

That's beyond the scope of this project, however, so I'm going to use some highly simplified mathematics instead, starting with the one-dimensional problem of descent:

```
speed = speed + gravity

altitude = altitude - speed
```

Surprisingly, this works pretty well, particularly when there's negligible atmosphere. Landing on the Earth's surface has lots more complexity with atmospheric drag and weather effects, but looking at Mars, and not during its glory days as Barsoom, it's atmosphere-free.

In my last article, I presented figures using feet as a unit of measure, but it's time to switch to metric, so for the simulation game, I'm using Martian gravity = 3.722 meters/sec/sec. The spaceship will enter the atmosphere at an altitude of 500 meters (about 1/3 mile), and players have just more than 15 seconds to avoid crashing onto the Martian surface, with a terminal velocity of 59m/s.

Since I'm making game out of it, the calculations are performed in

one-second increments, meaning that you actually can use the retro rockets at any point to compensate for the tug of gravity and hopefully land, rather than crash into Mars!

The equation changes only a tiny bit:

```
speed = speed + gravity + thrust
```

Again, there are complex astro-mechanical formulas to figure out force produced in a retro rocket burn versus fuel expended, but to simplify, I'm assuming that fuel is measured in output force: meters of counter thrust per second.

That is, if you are descending at 25 meters/second, application of 25 units of thrust will fully compensate and get you to zero descent, essentially hovering above the surface—until the inexorable pull of gravity begins to drag you back to the planet's surface, at least!

Gravity diminishes over distance, so too much thrust could break you completely free of the planet's gravitational pull. *No bueno*. To include that possibility, I'm going to set a ceiling altitude. Fly above that height, and you've broken free and are doomed to float off into space.

Floating-Point Math

Shell scripts make working with integer math quite easy, but any real calculations need to be done with floating-point numbers, which can be tricky in the shell. Therefore, Instead of using the `$(( ))` notation or `expr`, I'm going to tap the power of `bc`, the binary calculator program.

Being in a shell script, it's a bit awkward, so I'm going to use a rather funky notational convenience to constrain each calculation to a single line:

```
speed=$( $bc <<< "scale=3; $speed + $gravity + $thrust" )
```

By default, for reasons I don't understand, `bc` also wants to work with just integer values, so ask it to solve the equation 1/4, and it'll return 0. Indicate how many digits after the decimal place to track with `scale`, however, and it works a lot better. That's what I'm doing above with `scale=3`. That gives three digits of precision after the decimal point, enough for the game to function fine.

Thrust is the force being exerted by the rocket when it's firing, so that'll have to be something the user can enter after each second (the "game" part of the game).

Martian Lander Core Code

With that notation in mind, I can finally code the basics of the Martian lander:

```
while [ $altitude -gt 0 ]
do
  speed=$( $bc <<< "scale=3; $speed + $gravity + $thrust" )
  altitude=$( $bc <<< "scale=3; $altitude + $speed" )
  time=$(( $time + 1 ))
done
```

Obviously, there are a lot of variables to instantiate with the correct values, including gravity (–3.722), altitude (500 meters), thrust (retro rockets start powered down, so the initial value is 0), and speed and time also should both be set to 0.

Even with this tiny snippet, however, there are some problems. For example, the conditional that controls the `while` loop tests whether altitude is greater than zero. But altitude is a floating-point number, so the test fails. The easy solution is a second variable that's just the integer portion of altitude:

```
alt=$( echo $altitude | cut -d\. -f1 )
```

One problem solved.

Thrust is the force being exerted by the rocket when it's firing, so that'll have to be something the user can enter after each second (the "game" part of the game). But once it's fired, it should shut off again, so thrust needs to be set back to zero after each calculation is complete.

There's also a tricky challenge with positive and negative values here.

Gravity should be a negative value, as it's pulling the craft inexorably closer to the center of the planet. Therefore, thrust should be positive, since it's fighting gravity. That means speed will be negative when dropping toward the surface, and positive when shooting upward, potentially escaping the planet's gravity field entirely.

Here's a refinement on the core program loop:

```
while [ $alt -gt 0 ]
do
  speed=$( $bc <<< "scale=3; $speed + $gravity + $thrust" )
  thrust=0     # rocket fires on a per-second basis
  altitude=$( $bc <<< "scale=3; $altitude + $speed" )
  alt=$( echo "$altitude" | cut -d\. -f1 )
  time=$(( $time + 1 ))
  echo "$time seconds: speed: $speed m/s
      altitude: $altitude meters."
done
```

That works if you just want to plummet to the planet without any rocket firing. It'd look like this:

```
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.
6 seconds: speed: -22.332 m/s altitude: 421.838 meters.
7 seconds: speed: -26.054 m/s altitude: 395.784 meters.
8 seconds: speed: -29.776 m/s altitude: 366.008 meters.
9 seconds: speed: -33.498 m/s altitude: 332.510 meters.
10 seconds: speed: -37.220 m/s altitude: 295.290 meters.
11 seconds: speed: -40.942 m/s altitude: 254.348 meters.
12 seconds: speed: -44.664 m/s altitude: 209.684 meters.
13 seconds: speed: -48.386 m/s altitude: 161.298 meters.
14 seconds: speed: -52.108 m/s altitude: 109.190 meters.
15 seconds: speed: -55.830 m/s altitude: 53.360 meters.
```

At this point, the craft is dropping at 55m/s and is only 53 meters above the surface of the planet, so you can count on a big, ugly crash. BOOM!

At second 15, you could apply 55 units of thrust to jerk the craft back to zero speed, but what if you didn't have 55 units of fuel or if the max thrust you could exert at any given unit time was 25 due to rocket design (and passenger survival) constraints?

That's where this gets interesting.

In my next article, I'll dig into those constraints and finally add some interactivity to the program. For now, be careful out there flying this particular space craft. It's your budget that the replacement parts are coming out of, after all!

Props to Joel Garcia and Chris York for their ongoing assistance with all the gravitational formulas. Any errors and glitches are all due to my own rusty physics.■

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

RETURN TO CONTENTS

Simple Server Hardening

Server hardening doesn't have to be a series of arcane complex commands.

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

THESE DAYS, IT'S MORE IMPORTANT THAN EVER TO TIGHTEN UP THE SECURITY ON YOUR SERVERS, yet if you were to look at several official hardening guides, they read as though they were written for Red Hat from 2005. That's because they *were* written for Red Hat in 2005 and updated here and there through the years. I came across one of these guides when I was referring to some official hardening benchmarks for a PCI audit and realized if others new to Linux server administration were to run across the same guide, they likely would be overwhelmed with all of the obscure steps. Worse though, they likely would spend hours performing obscure sysctl tweaks and end up with a computer that was no more protected against a modern attack. Instead, they could have spent a few minutes performing a few simple hardening steps and ended

up with a more secure computer at the end. So in this article, I describe a few hardening steps that provide the most bang for the buck. These tips should take only a few minutes, yet for that effort, you should get a much more secure system at the end.

Classic Hardening

Before I talk about some hardening recommendations, I figured I'd start by highlighting some of those classic security steps you are likely to see in those older hardening guides. Now this isn't to say that all of these steps are necessarily bad advice, it's just that in many cases the advice refers to deprecated systems or describes steps that modern Linux server distributions have taken by default for years.

For instance, many hardening guides spend a lot of time focusing on tcpwrappers, a classic Linux service that lets you restrict which IPs can access particular services. These days, most administrators use iptables firewall rules to restrict access to ports instead. You also will be advised to enable the use of shadow passwords and to disable shells on common role accounts (like the mail, bind, www and mysql users). Although that isn't bad advice, the fact is that all Linux distributions already do this for you out of the box.

Another tip you usually will see in a hardening guide is to disable all unnecessary services, and in particular, the guides will tell you to disable telnet, daytime, chargen and a number of other obscure inetd services that not only haven't been turned on by default in a long time, but in many cases they also aren't even installed by default anymore. The fact is that most server distributions ship with all network services apart from SSH turned off. Speaking of SSH, now that I've talked a bit about some classic hardening tips, let me discuss a few modern hardening tips starting with SSH.

SSH

As I mentioned, just about every server you will encounter turns on SSH by default, and there is an assumption that you will use it for remote administration. Here are a few simple changes you can make to your /etc/ssh/sshd_config file that take only a second but make it more secure.

First, disable root logins and make sure that you use only SSH protocol 2

(previous protocols have known vulnerabilities). In many distributions (in particular many cloud images), these steps already may be done for you:

```
PermitRootLogin no
Protocol 2
```

A lot of people focus way too much, in my opinion, on SSH brute-force attacks when they talk about server hardening. It's true that if you put a Linux server on the internet, one of the first things you will see in your logs is a steady stream of SSH brute-force attempts. Many sysadmins go to lengths that I think fall somewhere between ineffective, absurd and overkill, including moving SSH to some random port (security by obscurity) or using a system like fail2ban. With fail2ban, your system reads failed login attempts and creates firewall rules to block attackers after a few failed attempts. This seems sensible on the surface, but it has a few problems:

1. This stops only attackers who have one machine—most have botnets and spread brute-force attacks across many IPs.

2. If you have a weak, easily guessable password and a common user name, they might guess the password before fail2ban kicks in.

3. It's risky to let attackers perform an action that automatically updates your system's firewall rules.

4. Usually internal networks are whitelisted—attackers still can brute-force attack you from a different compromised machine on your network.

Instead of going through all of those steps to mitigate SSH brute-force attacks, I recommend that you eliminate the attack entirely: disable password authentication and rely on SSH keys only. Before you enable this option, be sure that everyone who logs in to this machine (or at least the administrators) have generated and tested logging in using SSH keys—you wouldn't want to get locked out. When you are ready, change

the `PasswordAuthentication` parameter in your sshd_config to:

```
PasswordAuthentication no
```

The final quick SSH hardening step is to restrict which cryptography cipher suites and algorithms to use, so that you use only the ones that are considered to be safe by today's standards. I'm no cryptographer, but I don't have to be one to look at the recommendations from cryptographers and copy and paste them into my SSH config:

```
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,
  aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr

KexAlgorithms curve25519-sha256@libssh.org,
➥diffie-hellman-group-exchange-sha256

MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,
  hmac-ripemd160-etm@openssh.com,umac-128-etm@openssh.com,
➥hmac-sha2-512,hmac-sha2-256,hmac-ripemd160,umac-128@openssh.com
```

Once all of these settings are in place, restart the SSH service to use them.

Account Hardening

For general hardening of the system accounts, the best recommendation I can make is to disable the root account altogether and use only sudo. You also should avoid direct login to any shared accounts, whether it's the root account or some role account like a user that manages your application or web server. By requiring users to log in as themselves and then sudo up to root or role accounts, you provide a nice audit trail for who did what, and you make revoking access simpler when users no longer need an account—since the shared accounts won't have a password, you don't have to change them every time a member of the team leaves; instead, you can just remove that user's account.

Most distributions currently include sudo, and some also either disable

the root account by default or allow you to disable it during installation. Otherwise, you simply can edit your /etc/shadow file and replace whatever password you have in place for the root user with a * symbol. Just make sure you do have sudo working first with at least one account so you don't lock yourself out.

When using sudo there are a few practices you should follow to help keep it secure. First, while the use of `NOPASSWD` sudo rules (rules that don't require you to enter a password) are somewhat unavoidable for dæmons that may run cron jobs like backup jobs, you should restrict any `NOPASSWD` sudo rules to just those dæmon role accounts and require all real users to type in a password. As much as possible, you also should follow the principle of least privilege and grant users sudo access only to the specific commands they need instead of granting them access to run all commands as a particular user (especially the root user). Finally, if you find yourself granting users access to a general-purpose command to do something specific (like granting them access to service or systemctl so they can restart just one service), consider creating a simple shell script that runs the command with only the specific parameters you want and granting them sudo access to that script instead.

Although these hardening steps aren't the only things you should do to lock down your server, they are a good start and should take only a few minutes. In my next article, I'll add another round of simple hardening tips, including SSH client hardening and cloud hardening steps, and I'll finish up with some general-purpose recommendations.■

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

RETURN TO CONTENTS

2016

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.

Hodge Podge

Changing topics six times in one column? That's what it's like to talk with Shawn Powers!

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
New Products

FOR EVERY COLUMN, I TRY TO WRITE SOMETHING THAT IS INTERESTING, ENTERTAINING, EDUCATIONAL AND FUN. Sometimes I even succeed. Many other times I have some things I'd like to talk about, but there's not enough of it to fill a column. Sometimes I turn those ideas into UpFront pieces, and sometimes I just forget about them. This column, I decided a disjointed hodge podge would be the theme. So let's just have a virtual nerdy talk about stuff, shall we?

My Little Cloud Puff

It's really nice to have a server on the internet that is online even when you're not. Even with my two business-class internet connections (one cable modem and one DSL), the reliability of my connection is shoddy at best. The thing is, I don't really *need* very much in a server. I mainly use them for hosting files I want to access when out and about, or to test services, and so on. It used to be

prohibitively expensive to purchase a dedicated server. Thanks to VPS options though, it's actually really cheap to get full-blown internet servers that have a dedicated public IP address. In order to get the best deal, however, it's important to think about what you need and then shop around.

For example, I like to have a server I can use as a VPN. There are affordable services, like Private Internet Access (https://www.privateinternetaccess.com), which I do use for $40/year, but if you want your own VPN to protect your data, a VPS is perfect. The thing is, you need a specific type of VPS. It doesn't take much CPU, RAM or storage to run a VPN, but limited bandwidth on low-end servers really can be a showstopper for something like a VPN service. The key is to find a very low-end machine that has unmetered data. It usually doesn't have to be any faster than 100mbps either, because that sort of speed is still more than enough for browsing the web. (That said, it's hard to find a VPS with less than 1gbps speed.)

I'm not endorsing VPSCheap.net as a vendor, since I don't actually use it, but a quick search found this plan from https://vpscheap.net (Figure 1). For $20/year, you get a small server with unlimited bandwidth. For a VPN, it's perfect! Plus, with unmetered bandwidth, you can do things like run a BitTorrentSync server and not worry about getting your VPS shut off. The only downside with unmetered bandwidth is that it usually comes as a trade-off with memory, CPU and storage space/speed.

If you need storage, like for an offsite backup, other VPS plans exist that favor large storage while trading off CPU/RAM. For instance, Bit Accel (http://www.bitaccel.com) has 100GB of storage with 256MB of RAM

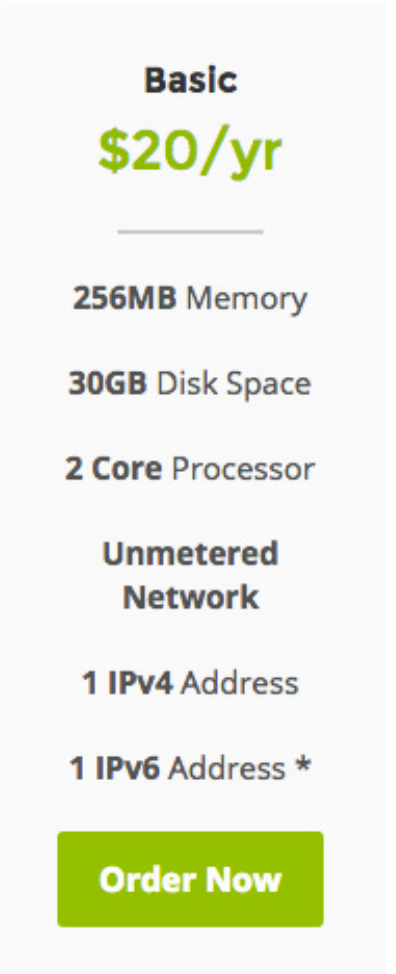


Figure 1. Although $20 is a significant amount of money, for a year, it's pretty reasonable!

and a shared CPU. It even offers unlimited transfer. It's only $1.99/month (Figure 2) with other plans scaling up and down from there. I don't use this either, so I can't vouch for its reliability.

Really, if you're looking for a small server of your own, the best place to look for a deal is LowEndBox (https://lowendbox.com). It's a very active community sharing links to deals from vendors all over the world. There's not really a database of current deals or the ability to sort based on criteria, but scrolling through a few pages is almost always worth the effort. Plus, the community leaves comments after each posted deal, so you get a feel for what sort of quality the vendors provide. (Most vendors also support Bitcoin payment, so if you truly want to stay anonymous, it's easier than going with places that require credit cards.)

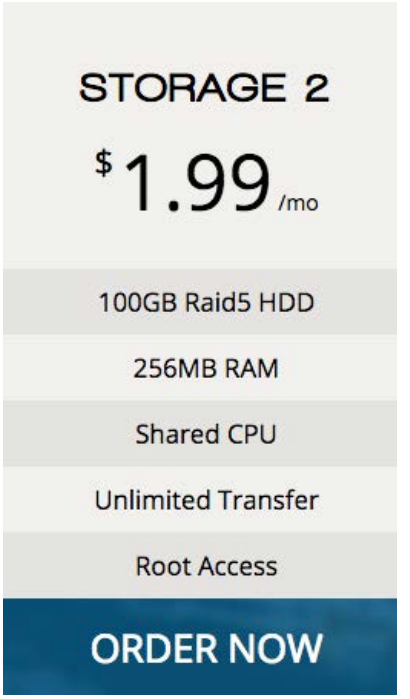


Figure 2. This example is only slightly more expensive than the other VPS, but it has 100GB of storage!

NAS

Through the the years, I've mentioned the various NAS devices I use for my home network. Kyle Rankin recently talked about his foray into ARM-based servers and his NAS system. I've gone between full-blown servers with off-the-shelf distributions installed, to embedded systems with proprietary partitioning (I'm looking at you, Drobo). There are advantages and disadvantages to the various methods and brands, but I would be lying if I said I didn't have a favorite: Synology (https://www.synology.com).

My current Synology NAS is the Synology 1815+ (Figure 3), which probably already is outdated. It has eight 3.5" SATA slots, and I have it populated with eight 6TB Western Digital Red NAS drives. It's configured to use RAID6, so I have 36TB of usable space. Plus, it has a handful of actually useful tools built in. (I use it as a reverse proxy, Transmission BitTorrent client and SickRage for keeping track of my television shows.) The best part about it, however, is that it manages to maintain its

Figure 3. Seriously, this is the nicest NAS I've ever used—and I've used many.

integrity automatically with regular system checks, self-installing updates and data scrubbing to keep the RAID system clean. It does those things without needing me to interact, and somehow it manages not to lock up my other servers connected via NFS and Samba. If there was ever a set-it-and-forget-it NAS device, it's the Synology in my basement. Compared to the Qnap, Netgear, Drobo and countless other NAS devices I've used, there's simply no comparison. I can't recommend Synology enough.

If you're set on using something like FreeNAS or a standard Linux distribution and hosting your own files, I should mention that I've always had very, very good luck with software-based RAID on Linux. The best part about software RAID is that it's not tied to a specific piece of RAID hardware. If your computer fails, you can take all the drives out of the broken computer, put them in a new system, and the RAID partition is simple to rebuild. In most cases, it will detect the software RAID partitions and rebuild itself automatically! Seriously, I'd chose Linux software RAID over a hardware-based RAID card any day, especially in a home or small office environment.

Bitcoin

Bitcoin isn't nearly the media sensation it once was, but for some of us, it's still an incredible idea that is just starting to gain traction. I still

use Bitcoin often, and I love how fast transactions take place. There have been a few recent advances in how you can use Bitcoin that are worth looking into.

Coinbase (http://www.coinbase.com) is still a very popular online wallet system. It supports buying and selling Bitcoin, and it recently has added support for Ethereum (an alternative cryptocurrency). One of my favorite new features, however, is the integration with PayPal. It's possible to buy and sell Bitcoin instantly using a connected PayPal account. That means rather than waiting for transfers to go back and forth to bank accounts (which takes days, even in our modern world), it's possible to send money via PayPal in a flash. It seems like a silly thing to get excited about, but far more people are comfortable with PayPal than are comfortable with Bitcoin, so having the ability to transfer back and forth is very nice.

Also, I've mentioned it before, but the too-good-to-be-true service offered by Purse (https://purse.io) is still amazingly reliable. You literally can get 15–20% or more off purchases from Amazon by using Bitcoin instead of paying directly. The system works by having people who can purchase from Amazon, but can't buy Bitcoin (maybe they have Amazon gift cards, or maybe they're from a country where buying Bitcoin is hard), buy items from your wish list in return for Bitcoin. Purse acts as an escrow service, and the entire process is simple. Plus, it works. The shipping often takes a couple extra days as you're waiting for someone to "buy" your offer, but if the item is something you can afford to wait on (such as Christmas gifts), it's a wonderful way to save significant money. And, most of us in the US easily can buy Bitcoin from Coinbase. In fact, Coinbase allows you to store your money as US dollars so the volatility of Bitcoin doesn't burn you while your money is stored there.

Being Big Brother, or Father

I love automation, the IoT, smart houses and quite frankly, data in general. My family has a mix of phones (various iPhones and Androids), and now that my eldest is in college, we're separated by geography as well. Thankfully, we can all keep track of each other with Life 360. If you're creeped out by the idea of your family,

and potentially the Life 360 company knowing your every move, I highly recommend you *do not* install the program. It's a real-time GPS tracker that shares your location with others in your circle. It doesn't have to be a family circle, but that certainly makes the most sense in my case. It also allows two free geo-fenced areas that allow you to get notifications when circle members come and go. For example, we have "Home" and "School" as our two free locations, so whenever the kids and my wife go to school (she's a teacher), I get notifications. When they leave to come home, I get notifications, and I can start dinner. In addition, we like that anyone in the family can look and see where the others are at a glance. Again, we're not concerned about privacy, but if you are, Life 360 might not be for you. Check it out at https://www.life360.com.

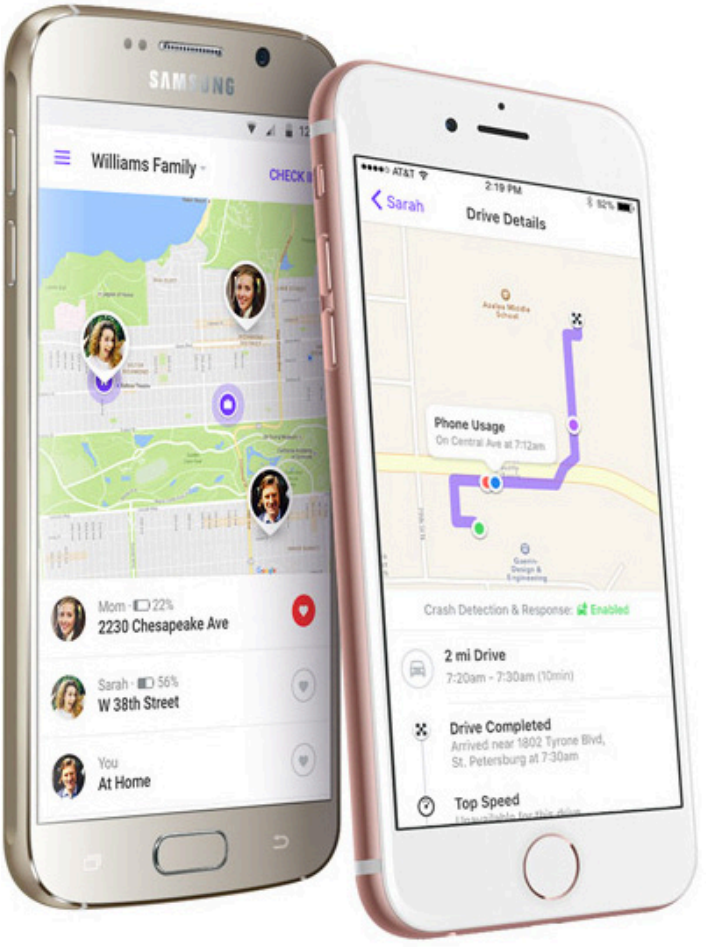


Figure 4. You might find it creepy, but my family loves it (photo from https://www.life360.com).

The other issue we're facing with a daughter away at college is how to monitor her car. A couple years ago we purchased "Automatic" devices that plug in to the OBD2 port under the dash of modern vehicles. It syncs with the driver's phone and explains check-engine lights, along with monitoring driving speeds, and makes suggestions for saving fuel and so on. The problem is that with our daughter leaving, it would sync only with her phone, so I wouldn't be able to help her remotely. Thankfully, Automatic has come out with a new product called Automatic Pro (Figure 5). It's fairly pricey at $129,

Figure 5. It plugs in under the dash, so thieves don't even know they're low-jacked!

but it has built-in 3G data, built-in GPS, and it works with If This Then That for triggering notifications. Plus, there's no monthly fee. What it means is that when my daughter is driving around campus and inevitably gets a check-engine notification on her 16-year-old Volkswagen Beetle, and she calls me in a panic, I can help her figure out how serious the problem is. Plus, it tracks her car, so if it's stolen, we can help the police find it. And the icing on the cake is that if she's in an accident, the Automatic Pro will call me and the police automatically, even if she's unconscious and even if she doesn't have her phone (it has 3G, remember?). I bought one for each of our

vehicles, and I sleep a lot better at night.

Send Me More!

This type of column reminds me of the "Lightning Talks" that were popular at conventions a few years ago. Sometimes it's nice to cover a bunch of interesting things that deserve mention but aren't really complex enough to warrant their own article. If you have cool tech info, or interesting ways Linux is integrated into your life, drop me an email at ljeditor@linuxjournal.com and let me know. If I share with everyone, I'll be sure to give you credit. Besides, sharing awesome ideas is really what the Open Source community is all about. Expanding that beyond code is awesome!■

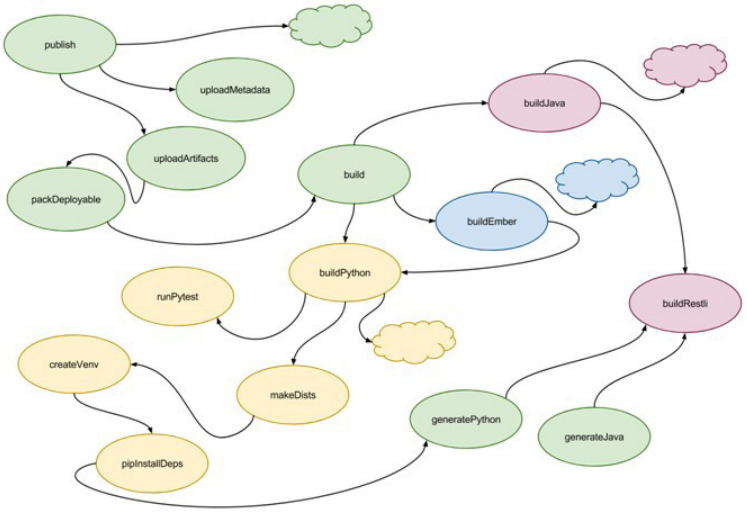
Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

RETURN TO CONTENTS

PREVIOUS
Shawn Powers' The Open-Source Classroom

NEXT
Feature: NTPsec: a Secure, Hardened NTP Implementation

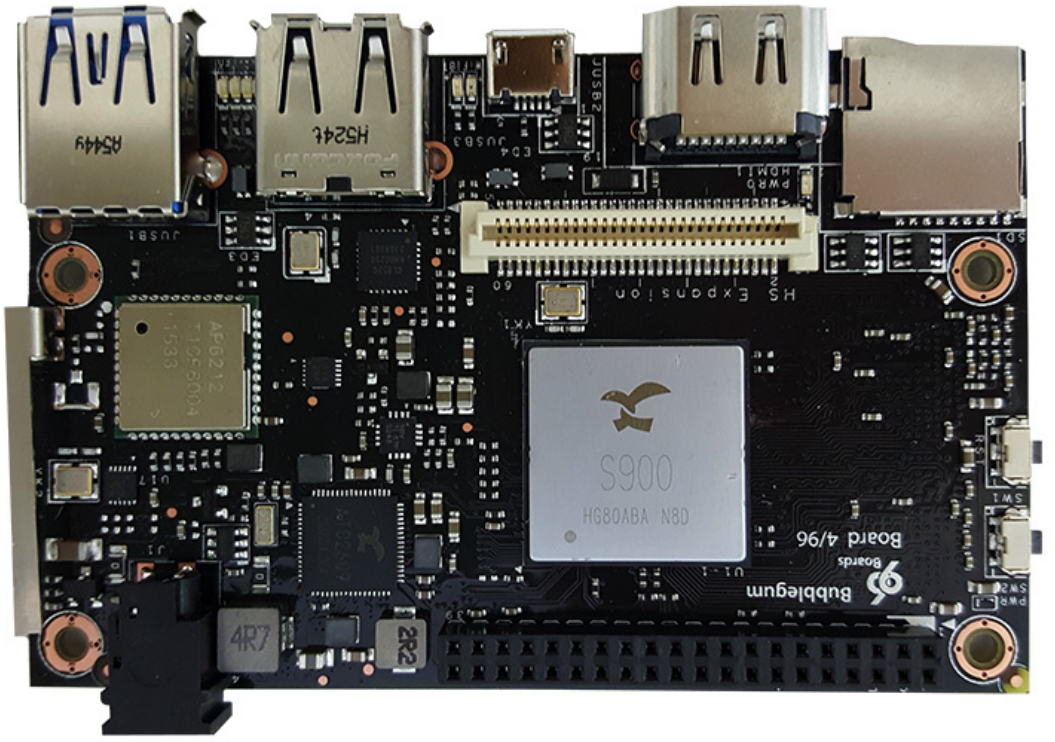
LinkedIn's {py}gradle



To facilitate better building of Android apps, the technical team at LinkedIn has developed {py}gradle, a new powerful, flexible and reusable Python packaging system. Now available to the Open Source community, {py}gradle wraps Python code into the Gradle build automation tool so that developers can build Android apps more easily. The tool currently is used for all Android projects at LinkedIn, and the company expects it to be widely used in the Open Source community as well. With {py}gradle, LinkedIn has bridged a gap between two similar but different technologies: Setuptools and Gradle. LinkedIn says that Python's Setuptools works well for self-contained Python applications with a small set of external dependencies. However, Setuptools can become problematic in certain situations as an organization's Python footprint grows, which led LinkedIn to integrate Gradle and a plugin architecture. For each language or technology stack, one simply needs to apply the build plugin for the underlying language or technology stack. With Gradle, LinkedIn was careful to enhance rather than replace the existing and idiomatic Python package management ecosystem.
http://linkedin.com/in/sholsapp

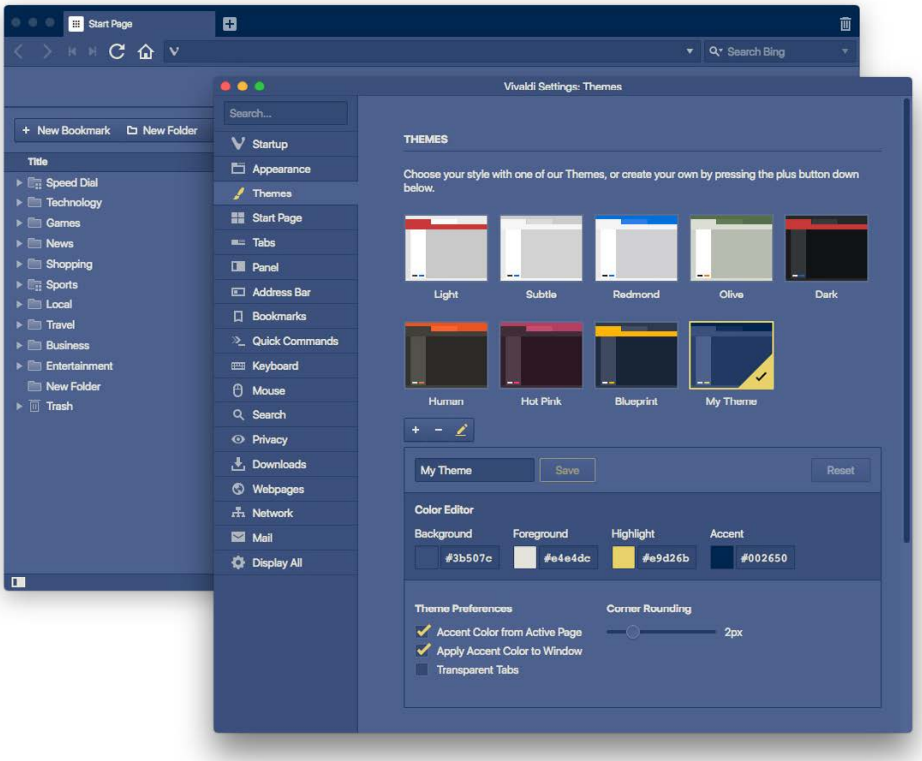
Netlist, Inc.'s HybriDIMM Storage Class Memory

The metachallenge in today's data-saturated world is turning Big Data into actionable insight. A straight line to faster insights can be found in Netlist, Inc.'s new HybriDIMM Storage Class Memory (SCM), which the company describes as the industry's first standards-based, plug-and-play SCM solution. Based on an industry-standard DDR4 LRDIMM interface, Netlist calls HybriDIMM the first SCM product to operate in current Intel x86 servers without BIOS and hardware changes, as well as the first unified DRAM-NAND solution that scales memory to terabyte storage capacities and accelerates storage to nanosecond memory speeds. Netlist adds that HybriDIMM's breakthrough architecture combines an on-DIMM co-processor with Netlist's PreSight technology—predictive software-defined data management—to unify memory and storage at near-DRAM speeds. The result is a dramatic improvement in application performance by reducing data access latency by up to 1,000 times vs. the fastest existing storage solution and up to 80% cost reduction compared to the highest existing memory density for in-memory applications.
http://netlist.com



Canonical Ltd.'s Ubuntu Core

Canonical Ltd.'s "Snappy" Ubuntu Core, a stripped-down version of Ubuntu designed for autonomous machines, devices and other internet-connected digital things, has gained significant traction in the chipset/semiconductor market recently. Following on partnerships with Samsung ARTIK, Qualcomm and MediaTek, Ubuntu core now adds UcRobotics' Bubblegum-96 board to the list of supported devices. Canonical describes Ubuntu Core as the ideal platform for developers in the semiconductor arena for deploying large numbers of IoT devices due to its secure and open-source design, transactional updates that are fast and reliable and its small footprint. The Bubblegum-96 board, a great open platform for applications and IoT devices, represents one of the most powerful commercial-edition Linaro 96Boards and the third Linaro board enabled on Ubuntu Core.
http://insights.ubuntu.com



Vivaldi Technologies Vivaldi Web Browser

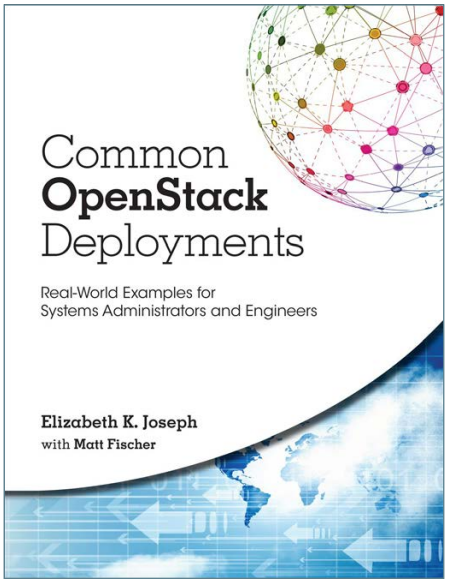
Wired magazine likes the Vivaldi web browser, calling it a tool for power users just like "500-pound squats are to power lifters". Led by a founder of the Opera browser, Vivaldi Technologies' browser eschews the pared-down base browser plus extensions model for one in which personalization rules. "You can truly make Vivaldi yours" is the company's mantra. The new Vivaldi 1.3 adds new options for personalization, most notably custom themes to allow customization of every UI element, protection for WebRTC IP leakage to improve privacy and additional mouse gestures, bringing to 90 the number of customizable browser actions. Platform-specific improvements for Linux users in this release include improved right-click tab hibernation for conserving system resources and better support for proprietary media in HTML5.
http://vivaldi.com

Penclic B3 Mouse

"Does the world need a new computer mouse?" asks Penclic. "Yes it does!" says the Swedish peripherals developer. Most devices in our lives have undergone extensive changes through the years, notes Penclic, save the unlucky, unglamorous computer mouse. The poor little guy is due not just for a facelift but a total makeover in the form of the new Penclic B3 Mouse, a faster, better and sleeker product. With this device, Penclic applies its formula of uniting cutting-edge technology with innovative, ergonomic Swedish design, resulting in a "new and amazing user experience". The Penclic Mouse looks, feels and moves like a pen, says the company, and is so responsive and intuitive that it nearly feels like the mouse predicts where the user intends to move the cursor. The device enables the extension of bodily movements and harnesses the natural power and dexterity in our fingers and hands, which promotes a healthy and natural working position. Endless clicking and unnecessary cursor movements are things of the past. Penclic's innovative pen grip counteracts health problems like repetitive strain injury that often arise from using a traditional computer mouse. The Penclic B3 mouse is compatible with Linux, Mac OS and Windows systems that support HID 1.1.
http://penclic.se

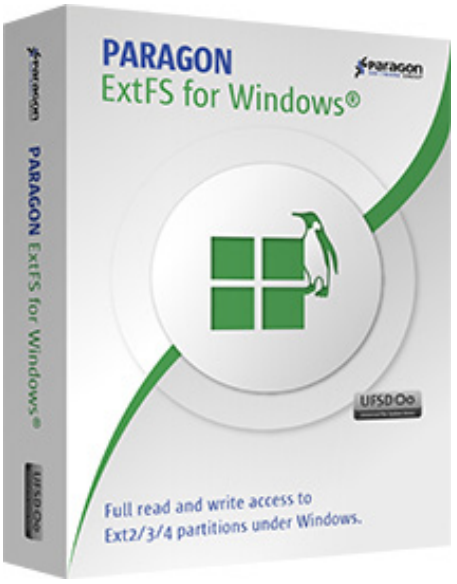
Linaro Ltd.'s OpenDataPlane

The OpenDataPlane (ODP) project is a founding initiative by the Linaro Networking Group to produce an open-source, cross-platform application programming interface (API) for the networking Software Defined Data Plane. Linaro Ltd. recently announced the availability of the first Long Term Support (LTS) Monarch release of OpenDataPlane, which will enable other projects to leverage the acceleration provided by the ODP APIs now that the code base will be fully supported for the foreseeable future. Linaro adds that work already has begun on network protocol stacks, such as OpenFastPath (OFP), products like the nginx web server accelerated with ODP and OFP and libraries like OpenSSL that provide crypto acceleration via ODP. In addition, ODP and ODP-based products, such as OFP, nginx and OpenSSL, now can be made available as packages in popular Linux distributions like Debian, CentOS and OpenEmbedded. To accompany the release, Linaro launched a validation test suite that permits users and vendors to verify API compatibility between different ODP implementations.
http://linaro.org and http://opendataplane.org



Elizabeth K. Joseph and Matt Fischer's *Common OpenStack Deployments* (Prentice Hall)

Public and private clouds typically are built and integrated using OpenStack technology. Professionals seeking guidance on this important topic should investigate Elizabeth K. Joseph and Matthew Fischer's new book *Common OpenStack Deployments*, which its publisher, Prentice Hall, describes as "a complete, practical guide to deploying OpenStack and understanding its internals". The authors share up-to-date, detailed strategies for deploying OpenStack on both virtual and physical servers, as well for using OpenStack to address any real-world challenge. Joseph and Fischer begin the book by covering OpenStack concepts and components by guiding the reader through small-scale, virtualized deployments. Later, readers learn how to build large, horizontally scalable infrastructures that integrate multiple components in a feature-rich cloud environment. Sprinkled throughout the book is current coverage of enhancements that make the OpenStack platform more mature and production-ready, plus expert tips on debugging and growth. Finally, the authors explain the broader OpenStack ecosystem, illustrating how to drive value through hybrid clouds blending local and hosted solutions.
http://informit.com



Paragon Software Group's ExtFS for Windows

Fellow Linux/Windows dual-booters out there are familiar with this problem: you can access Windows files from your Linux session, but not the other way around. Saving humanity by chipping away at Windows' illogic is ExtFS for Windows, a handy utility from the Paragon Software Group that gives dual-boot users full read-write access to Linux partitions from their Windows session. ExtFS' drivers are based on Paragon's proprietary cross-platform Universal File System Driver (UFSD) technology, which provides a higher data transfer rate than native filesystem performance. Paragon recently unveiled a new feature-rich version 4 of ExtFS for Windows, which comes with extended support for Ext4 file formats enabling the highest transfer speeds and mounts Linux volumes, including those more than 2TB in size, at up to twice the rate of the previous release. Thanks to the development of ExtFS, Paragon calls itself the first software developer to implement a full set of drivers with complete read and write access to partitions on all popular filesystems. ExtFS for Mac is also available from the company.

http://paragon-software.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

NTPsec
a Secure, Hardened NTP Implementation

Network time service has been in trouble. Now it's getting a makeover.

ERIC S. RAYMOND

PREVIOUS
New Products

NEXT
Feature: Flat File Encryption with OpenSSL and GPG

Network time synchronization—aligning your computer's clock to the same Universal Coordinated Time (UTC) that everyone else is using—is both necessary and a hard problem. Many internet protocols rely on being able to exchange UTC timestamps accurate to small tolerances, but the clock crystal in your computer drifts (its frequency varies by temperature), so it needs occasional adjustments.

That's where life gets complicated. Sure, you can get another computer to tell you what time it thinks it is, but if you don't know how long that packet took to get to you, the report isn't very useful. On top of that, its clock might be broken—or lying.

To get anywhere, you need to exchange packets with several computers that allow you to compare your notion of UTC with theirs, estimate network delays, apply statistical cluster analysis to the resulting inputs to get a plausible approximation of real UTC, and then adjust your local clock to it. Generally speaking, you can get sustained accuracy to on the close order of 10 milliseconds this way, although asymmetrical routing delays can make it much worse if you're in a bad neighborhood of the internet.

The protocol for doing this is called NTP (Network Time Protocol), and the original implementation was written near the dawn of internet time by an eccentric genius named Dave Mills. Legend has it that Dr Mills was the person who got a kid named Vint Cerf interested in this ARPANET thing. Whether that's true or not, for decades Mills was *the* go-to guy for computers and high-precision time measurement.

Eventually though, Dave Mills semi-retired, then retired completely. His implementation (which we now call NTP Classic) was left in the hands of the Network Time Foundation and Harlan Stenn, the man *Information Week* feted as "Father Time" in 2015 (http://www.informationweek.com/it-life/ntps-fate-hinges-on-father-time/d/d-id/1319432). Unfortunately, on NTF's watch, some serious problems accumulated. By that year, the codebase already was more than a quarter-century old, and techniques that had been state of the art when it was first built were showing their age. The code had become rigid and difficult to modify, a problem exacerbated by the fact that very few people actually understood the Byzantine time-synchronization algorithms at its core.

Among the real-world symptoms of these problems were serious security issues. That same year of 2015, InfoSec researchers began to realize that NTP Classic installations were being routinely used as DDoS amplifiers—ways for crackers to packet-lash target sites by remote control. NTF, which had complained for years of being under-budgeted and understaffed, seemed unable to fix these bugs.

This is intended to be a technical article, so I'm going to pass lightly over the political and fundraising complications that ensued. There was, alas, a certain amount of drama. When the dust finally settled, a very reluctant fork of the Mills implementation had been performed in early June 2015 and named NTPsec (https://www.ntpsec.org). I had been funded on an effectively full-time basis by the Linux Foundation to be the NTPsec's architect/tech-lead, and we had both the nucleus of a capable development team and some serious challenges.

This much about the drama I will say because it is technically relevant: one of NTF's major problems was that although NTP Classic was nominally under an open-source license, NTF retained pre-open-source habits of mind. Development was closed and secretive, technically and socially isolated by NTF's determination to keep using the BitKeeper version-control system. One of our mandates from the Linux Foundation was to fix this, and one of our first serious challenges was simply moving the code history to git.

This is never trivial for a codebase as large and old as NTP Classic, and it's especially problematic when the old version-control system is proprietary with code you can't touch. I ended up having to revise Andrew Tridgell's SourcePuller utility heavily—yes, the same code that triggered Linus Torvalds' famous public break with BitKeeper back in 2005—to do part of the work. The rest was tedious and difficult hand-patching with reposurgeon (http://www.catb.org/esr/reposurgeon). A year later in May 2016—far too late to be helpful—BitKeeper went open source.

Strategy and Challenges

Getting a clean history conversion to git took ten weeks, and grueling as that was, it was only the beginning. I had a problem: I was expected to harden and secure the NTP code, but I came in knowing very little about time service and even less about security engineering. I'd picked up a few clues about the

If I could refactor, cut and simplify the NTP Classic codebase enough, maybe all those domain-specific problems would come out in the wash.

former from my work leading GPSD (http://catb.org/gpsd), which is widely used for time service. Regarding the latter, I had some basics about how to harden code—because when you get right down to it, *that* kind of security engineering is a special case of reliability engineering, which I *do* understand. But I had no experience at "adversarial mindset", the kind of active defense that good InfoSec people do, nor any instinct for it.

A way forward came to me when I remembered a famous quote by C. A. R. Hoare: "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies." A slightly different angle on this was the perhaps better-known aphorism by Saint-Exupéry that I was to adopt as NTPsec's motto: "Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."

In the language of modern InfoSec, Hoare was talking about reducing attack surface, global complexity and the scope for unintended interactions leading to exploitable holes. This was bracing, because it suggested that maybe I didn't actually need to learn to think like an InfoSec specialist or a time service expert. If I could refactor, cut and simplify the NTP Classic codebase enough, maybe all those domain-specific problems would come out in the wash. And if not, then at least taking the pure software-engineering approach I was comfortable with might buy me enough time to learn the domain-specific things I needed to know.

I went all-in on this strategy. It drove my argument for one of the very first decisions we made, which was to code to a fully modern API—pure POSIX and C99. This was only partly a move for ensuring portability; mainly I wanted a principled reason (one we could give potential users and allies) for ditching all the cruft in the codebase from the big-iron UNIX era.

And there was a *lot* of that. The code was snarled with portability #ifdefs and shims for a dozen ancient UNIX systems: SunOS, AT&T System V, HP-UX, UNICOS, DEC OSF/1, Dynix, AIX and others more obscure—all relics from the days before API standardization really took hold. The NTP Classic people were too terrified of offending their legacy customers to remove any of this stuff, but I knew something they apparently didn't. Back around 2006, I had done a cruft-removal pass over GPSD, pulling it up to pretty strict POSIX conformance—and nobody from GPSD's highly varied userbase ever said boo about it or told me they missed the ancient portability shims at all. Thus, what I had in my pocket was nine years of subsequent GPSD field experience telling me that the standards people had won their game without most UNIX systems programmers actually capturing all the implications of that victory.

So I decrufted the NTP code *ruthlessly*. Sometimes I had to fight my own reflexes in order to do it. I too have long been part of the culture that says "Oh, leave in that old portability shim, you never know, there just *might* still be a VAX running ISC/5 out there, and it's not doing any harm."

But when your principal concern is reducing complexity and attack surface, that thinking is wrong. No individual piece of obsolete code costs very much, but in a codebase as aged as NTP Classic, the cumulative burden on readability and maintainability becomes massive and paralyzing. You have to be hard about this; it all has to go, or exceptions will pile up on you, and you'll never achieve the mission objective.

I'm emphasizing this point, because I think much of what landed NTP Classic in trouble was not want of skill but a continuing failure of what one might call surgical courage—the kind of confidence and determination it takes to make that first incision, knowing that you're likely to have to make a bloody mess on the way to fixing what's actually wrong. Software systems architects working on legacy infrastructure code need this quality almost as much as surgeons do.

The same applies to superannuated features. The NTP Classic codebase was full of dead ends, false starts, failed experiments, drivers for obsolete clock hardware, and other code that might have been a good idea once but had long outlived the assumptions behind it—Mode 7 control messages, Interleave mode, Autokey, an SNMP dæmon that was never conformant to the published standard and never finished, and a

half-dozen other smaller warts. Some of these (Mode 7 handling and Autokey especially) were major attractors for security defects.

As with the port shims, these lingered in the NTP Classic codebase not because they couldn't have been removed, but because NTF cherished compatibility back to the year zero and had an allergic reaction to the thought of removing any features at all.

Then there were the incidental problems, the largest of which was Classic's build system. It was a huge, crumbling, buggy, poorly documented pile of autoconf macrology. One of the things that jumped out at me when I studied NTF's part of the code history was that in recent years they seemed to spend as much or more effort fighting defects in their build system as they did modifying code.

But there was one amazingly good thing about the NTP Classic code: that despite all these problems, it *still worked*. It wheezed and clanked and was rife with incidental security holes, but it did the job it was supposed to do. When all was said and done, and all the problems admitted, Dave Mills had been a brilliant systems architect, and even groaning under the weight of decades of unfortunate accretions, NTP Classic still functioned.

Thus, the big bet on Hoare's advice at the heart of our technical strategy unpacked to two assumptions: 1) that beneath the cruft and barnacles the NTP Classic codebase was fundamentally sound, and 2) that it would be practically possible to clean it up without breaking that soundness.

Neither assumption was trivial. This could have been the a priori *right* bet on the odds and still failed because the Dread God Finagle and his mad prophet Murphy micturated in our soup. Or, the code left after we scraped off the barnacles could actually turn out to be unsound, fundamentally flawed.

Nevertheless, the success of the team and the project at its declared objectives was riding on these premises. Through 2015 and early 2016 that was a constant worry in the back of my mind. *What if I was wrong?* What if I was like the drunk in that old joke, looking for his keys under the streetlamp when he's dropped them two darkened streets over because "Offisher, this is where I can see."

The final verdict is not quite in on that question; as I write, NTPsec is still in beta. But, as we shall see, there are now (in August 2016) solid indications that the project is on the right track.

Thus, it came to be that the windowsill above my home-office desk is now home to six headless Raspberry Pis, all equipped with on-board GPSes, all running stability and correctness tests on NTPsec 24/7—just as good as a conventional rack full of servers, but far less bulky and expensive!

Stripping Down, Cleaning Up

One of our team's earliest victories after getting the code history moved to git was throwing out the autoconf build recipe and replacing it with one written in a new-school build engine called waf (also used by Samba and RTEMS). Builds became *much* faster and more reliable. Just as important, this made the build recipe an order of magnitude smaller so it could be comprehended as a whole and maintained.

Another early focus was cleaning up and updating the NTP documentation. We did this before most of the code modifications because the research required to get it done was an excellent way to build knowledge about what was actually going on in the codebase.

These moves began a virtuous cycle. With the build recipe no longer a buggy and opaque mess, the code could be modified more rapidly and with more confidence. Each bit of cruft removal lowered the total complexity of the codebase, making the next one slightly easier.

Testing was pretty ad hoc at first. Around May 2016, for reasons not originally related to NTPsec, I became interested in Raspberry Pis. Then it occurred to me that they would make an excellent way to run long-term stability tests on NTPsec builds. Thus, it came to be that the windowsill above my home-office desk is now home to six headless Raspberry Pis, all equipped with on-board GPSes, all running stability and correctness tests on NTPsec 24/7—just as good as a conventional rack full of servers, but far less bulky and expensive!

We got a lot done during our first 18 months. The headline number shows just how much was the change in the codebase's total size. We went from 227KLOC to 75KLOCK, cutting the total line count by a full factor of three.

Dramatic as that sounds, it actually understates the attack-surface reduction we achieved, because complexity was not evenly distributed in the codebase. The worst technical debt, and the security holes, tended to lurk in the obsolete and semi-obsolete code that hadn't gotten any developer attention in a long time. NTP Classic was not exceptional in this; I've seen the same pattern in other large, old codebases I've worked on.

Another important measure was systematically hunting down and replacing all unsafe C function calls with equivalents that can provably not cause buffer overruns. I'll quote from NTPsec's hacking guide:

- strcpy, strncpy, strcat: use strlcpy and strlcat instead.
- sprintf, vsprintf: use snprintf and vsnprintf instead.
- In scanf and friends, the %s format without length limit is banned.
- strtok: use strtok_r() or unroll this into the obvious loop.
- gets: use fgets instead.
- gmtime(), localtime(), asctime(), ctime(): use the reentrant *_r variants.
- tmpnam(): use mkstemp() or tmpfile() instead.
- dirname(): the Linux version is re-entrant but this property is not portable.

This formalized an approach I'd used successfully on GPSD—instead of fixing defects and security holes after the fact, constrain your code so that it *cannot have* entire classes of defects.

The experienced C programmers out there are thinking "What about

wild-pointer and wild-index problems?" And it's true that the achtung verboten above will not prevent those kinds of overruns. That's why another prong of the strategy was systematic use of static code analyzers like Coverity, which actually is pretty good at picking up the defects that cause that sort of thing. It's not 100% perfect, C will always allow you to shoot yourself in the foot, but I knew from prior success with GPSD that the combination of careful coding with automatic defect scanning can reduce your bug load a very great deal.

To help defect scanners do a better job, we enriched the type information in the code. The largest single change of this kind was changing int variables to C99 bools everywhere they were being used as booleans.

Little things also mattered, like fixing all compiler warnings. I thought it was shockingly sloppy that the NTP Classic maintainers hadn't done this. The pattern detectors behind those warnings are there because they often point at real defects. Also, voluminous warnings make it too easy to miss actual errors that break your build. And you never want to break your build, because later on, that will make bisection testing more difficult.

An early sign that this systematic defect-prevention approach was working was the extremely low rate of bugs we detected by testing as having been introduced during our cleanup. In the first 14 months, we averaged less than one iatrogenic C bug every 90 days.

I would have had a lot of trouble believing that if GPSD hadn't posted a defect frequency nearly as low during the previous five years. A major lesson from both projects is that applying best practices in coding and testing really works. I pushed this point back in 2012 in my essay on GPSD for *The Architecture of Open Source, Volume 2* (http://www.aosabook.org/en/gpsd.html); what NTPsec shows is that GPSD is not a fluke.

I think this is one of the most important takeaways from both projects. We really don't have to settle for what have historically been considered "normal" defect rates in C code. Modern tools and practices can go a very long way toward driving those defect rates toward zero. It's no longer even very difficult to do the right thing; what's too often missing is a grasp of the possibility and the determination to pursue it.

And here's the real payoff. Early in 2016, CVEs (security alerts) started issuing against NTP Classic that NTPsec dodged because we had already

cut out their attack surface before we knew there was a bug! This actually became a regular thing, with the percentage of dodged bullets increasing over time. Somewhere, Hoare and Saint-Exupéry might be smiling.

The cleanup isn't done yet. We're testing a major refactoring and simplification of the central protocol machine for processing NTP packets. We believe this already has revealed a significant number of potential security defects nobody ever had a clue about before. Every one of these will be another dodged bullet attributable to getting our practice and strategic direction right.

Features? What Features?

I have yet to mention new features, because NTPsec doesn't have many; that's not where our energy has been going. But, here's one that came directly out of the cleanup work.

When NTP originally was written, computer clocks delivered only microsecond precision. Now they deliver nanosecond precision (though not all of that precision is accurate). By changing some internal representations, we have made NTPsec able to use the full precision of modern clocks when stepping them, which can result in a factor 10 or more of accuracy improvement with real hardware, such as GPSDOs and dedicated time radios.

Fixing this was about a four-line patch. It might have been noticed sooner if the code hadn't been using an uneasy mixture of microsecond and nanosecond precision for historical reasons. As it is, anything short of the kind of systematic API-usage update we were doing would have been quite unlikely to spot the problem.

A longstanding pain point we've begun to address is the nigh-impenetrable syntax of the ntp.conf file. We've already implemented a new syntax for declaring reference clocks that is far easier to understand than the old. We have more work planned toward making composing NTP configurations less of a black art.

The diagnostic tools shipped with NTP Classic were messy, undocumented and archaic. We have a new tool, ntpviz, which gives time-server operators a graphical and much more informative view of what's been going on in the server logfiles. This will assist in understanding and mitigating various sources of inaccuracy.

Where We Go from Here

We don't think our 1.0 release is far in the future—in fact, given normal publication delays, it might well have shipped by the time you read this. Our early-adopter contingent includes a high-frequency-trading company for which accurate time is business-critical. The company hasn't actually put NTPsec in production yet, though its techie in charge of time actively contributes to our project and expects to adopt it for production in the not-distant future.

There remains much work to be done after 1.0. We're cooperating closely with IETF to develop a replacement for Autokey public-key authentication that actually works. We want to move as much of the C code as possible outside ntpd itself to Python in order to reduce long-term maintenance load. There's a possibility that the core dæmon itself might be split in two to separate the TCP/IP parts from the handling of local reference clocks, drastically reducing global complexity.

Beyond that, we're gaining insight into the core time-synchronization algorithms and suspect there are real possibilities for improvement in those. Better statistical filtering that's sensitive to measurements of network weather and topology looks possible.

It's an adventure, and we welcome anyone who'd like to join in. NTP is vital infrastructure, and keeping it healthy over a time frame of decades will need a large, flourishing community. You can learn more about how to take part at our project website: https://www.ntpsec.org.■

Eric S. Raymond is a wandering anthropologist and trouble-making philosopher. He's been known to write a few lines of code too. Actually, if the tag "ESR" means nothing to you, what are you doing reading this magazine?

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

RETURN TO CONTENTS

Flat File Encryption with OpenSSL and GPG

PGP is the more well known utility for handling flat files, but OpenSSL offers a modular set of tools that allows easy "mix and match" of asymmetric key exchange, digest verification and symmetric ciphers. Comparing approaches with each yields new insight into the features and limitations of the whole collection of components.

CHARLES FISHER

PREVIOUS
Feature: NTPsec: a Secure, Hardened NTP Implementation

NEXT
Doc Searls' EOF

The Pretty Good Privacy (PGP) application, which has long been known as a primary tool for file encryption, commonly focused on email. It has management tools for exchanging credentials with peers and creating secure communication channels over untrusted networks. GNU Privacy Guard (GPG) has carried on this legacy with a free and open implementation included in most major Linux distributions. PGP/GPG has proven highly resistant to cryptographic attack and is a preeminent tool for secure communications.

OpenSSL is more known for network security, but it also has tools useful for most aspects of encrypting flat files. Although using OpenSSL requires more knowledge of specific algorithms and methods, it can be more flexible in a number of scenarios than other approaches. OpenSSH keys can be used transparently for flat file encryption with OpenSSL, allowing user and/or host SSH keys to pervade any number of unrelated services.

OpenSSL is also useful for illustrating the sequence of encryption techniques that create secure channels. This knowledge is applicable in many other situations, so the material is worth study even if there is no immediate need for the tools.

OpenSSL Flat File Processing

Many common programs in UNIX have implementations within the OpenSSL command-line utility. These include digest/checksum tools (md5sum, sha1sum, sha256sum), "ASCII-Armor" tools (base64/uuencode/uudecode), "safe" random number generation and MIME functions in addition to a suite of cipher and key management utilities. Because OpenSSL often is found on non-UNIX platforms, those utilities can provide a familiar interface on unfamiliar systems for UNIX administrators.

Let's begin with a complete script for flat file encryption with OpenSSL, using asymmetric exchange of a session key, SHA-256 digest checksums and the use of a symmetric cipher. This entire exchange, both to encode and decode, is presented in the following text for the Korn shell (GNU Bash also may be used with no required changes):

```
#!/bin/ksh

set -euo pipefail
```

```
IFS=$'\n\t' #http://redsymbol.net/articles/unofficial-bash-strict-mode/

# openssl genrsa -aes256 -out ~/.prv.key 2868        # Make private key
# openssl rsa -in ~/.prv.key -pubout -out ~/.pub.key # Make public key

PVK=~/.prv.key; PBK=~/.pub.key
SKEY=$(mktemp -t crypter-session_key-XXXXXX)          # Symmetric key

case $(basename "${0}") in

encrypter) ####sender needs only public key - not .pas or .prv.key#####
 openssl rand -base64 48 -out "${SKEY}"              # Generate sesskey
 openssl rsautl -encrypt -pubin -inkey "${PBK}" -in "${SKEY}" |
  openssl base64;
 echo __:
 for F                                               # Generate digest
 do echo $(openssl dgst -sha256 "${F}" | sed 's/^[^ ]*[ ]//') "${F}"
 done | openssl enc -aes-128-cbc -salt -a -e -pass "file:${SKEY}"
 echo __:
 for F                                               # Encrypt files
 do openssl enc -aes-128-cbc -salt -a -e -pass "file:${SKEY}" -in "${F}"
    echo __:
 done ;;

decrypter) #####receiver#################################################
 TMP=$(mktemp -t crypter-tmp-XXXXXX); PW=${HOME}/.pas; unset IFS
 DGST=$(mktemp -t crypter-dgst-XXXXXX); #cd ${HOME}/dest #unpack dest
 while read Z
 do if [[ ${Z%%:*} == '__' ]]
    then if [[ -s "${SKEY}" ]]
         then if [[ -s "${DGST}" ]]
              then openssl enc -aes-128-cbc -d -salt -a -in "${TMP}" \
                    -pass "file:${SKEY}" -out "${NAME[I]}"
                   ((I+=1))                          # Decrypt files
              else openssl enc -aes-128-cbc -d -salt -a -in "${TMP}" \
                    -pass "file:${SKEY}" -out "${DGST}"
```

```
                    date +%Y/%m/%d:%H:%M:%S
                    I=0
                    while read hash file
                    do echo "${hash} ${file}"
                       HASH[I]=${hash}
                       NAME[I]=$(basename "${file}")  # Unpack only @dest
                       ((I+=1))
                    done < "${DGST}"
                    I=0
                fi
            else openssl base64 -d -in "${TMP}" |       # Extract sesskey
                 openssl rsautl -decrypt -inkey "${PVK}" \
                  -passin "file:${PW}" -out "${SKEY}"

                #Older OpenSSL: decrypt PVK; c/sha256/sha1/; no strict
                #openssl rsa -in "${PVK}" -passin "file:${PW}" -out "$DGST"
                #openssl base64 -d -in "${TMP}" |          # Extract sesskey
              # openssl rsautl -decrypt -inkey "${DGST}" -out "${SKEY}"
                #> "${DGST}"
            fi
            > "${TMP}"                                    # Erase tempfile
       else echo "${Z}" >> ${TMP}
       fi
    done
    I=0
    while [[ ${I} -lt ${#NAME[*]} ]]                      # Verify digest
    do F=$(openssl dgst -sha256 "${NAME[I]}" | sed 's/^[^ ]*[ ]//')
       if [[ "${F}" = "${HASH[I]}" ]]
       then echo "${NAME[I]}: ok"; else echo "${NAME[I]}: **SHA CORRUPT**"
       fi
       ((I+=1))
    done
    rm "${TMP}" "${DGST}" ;;

esac
rm "${SKEY}"
```

I will specifically cover everything above to the end of the `encrypter` case block, as this succinctly addresses the major cryptographic components of most encryption tools—that is, SSH, TLS, PGP and so on.

First, I include a well known strict mode for Korn/Bash published by Aaron Maxwell that can prevent coding errors, as documented at the URL near the top of the script.

Next, I generate an RSA private key. RSA, as an "asymmetric cipher", uses pairs of keys for communication and was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977. Other asymmetric ciphers in common use are Diffie-Hellman key exchange and Elliptic Curve, but OpenSSL's support for RSA is more thorough, complete and widespread (a bug listed in OpenSSL's dhparam manual page indicates "There should be a way to generate and manipulate DH keys."). With an asymmetric cipher, content encrypted by one key can only be read in clear text by the other. You can use such keypairs not only to communicate securely, but also to prove authenticity. Below is an example of the generation of an RSA private key of a non-standard size of 2868 bits:

```
$ openssl genrsa -aes256 -out ~/.prv.key 2868
Generating RSA private key, 2868 bit long modulus
.............++
.....................................................................++
e is 65537 (0x10001)
Enter pass phrase for /home/ol7_user/.prv.key:
Verifying - Enter pass phrase for /home/ol7_user/.prv.key:

$ chmod 400 .prv.key

$ cat .prv.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,16846D1D37C82C834E65B518C456DE2F

WXF7aX6M0KiQTFxSApsbj5Tsg/duW61CgkDJxjxmc16BOZ7oAzUS05gqYy5FtTbK
tNTnRXj8EvZ2qkNXDpPIOzc9frG5YFN/XNctnNKpdQTgLXRdkGjR+dVanPo2ZY5s
DxzZMPKkpXs6J8ZV2jPhQ+5Xj/ZjcdyKbogIqH4JDGE4+RnzT9yGr5rJ4oIgfa62
```

```
Ty30CVkgBzHv8CPA9KZzvjtoco4Sm6YQRArFajCYjSbYc3gJfOxqTpOhDvOlSSau
nJ8fgwq/DIMoS1ZwNPrCDuTZ6r3rCwlalPLRZC9zhs0tdGzP/9PmTH9Il1W6m36p
5C4656/MVjVgtG4K10Fl+cCrjuPgJgEeb/CuYRkoWRJb0FIYqDND2pWuavfZtAXW
VQPQPWKl9//BSPwDK6A+jubZQoidXwaPUPKMNW25uTrrw9Fuiw11X7LyJT3wNWwC
0KsiXqKpO+jX7GGN5SBlZ1oJO/bNE6LhmPikEm+ZbLxDKPWU0HBY+uc9BcnG5ZKW
4npk/PcXQUxv1jozzKXQape0nPQMHbMrOcAao8feHTiUYcLM+/x+dc2Xlm5xr8jU
/yh9E2yDjkXI/MObuRaCzOTVRLyom8IFwVY99XaeaMGQUXe/C/E0Dg5NYpIo7GW6
7ptV22/pw8C9PHu5/ZJFFn0u3BSYzQqMGwyXojria/1xgGjtGBHsjLPH9oLresM1
IOfC0HD2223ug1vWo/Bf9OvuYkpKbmDXunLy14mosgmGvGltChkuec7rsHUjeC4a
RhGQU+mcqI/U4ffuyvSiEd3tpXKiwLtKkIEji4csMyTA1zCEZgoLo3qCm3nzlX3G
fI7IFzUXHstg0YrQ50Sp5A2Ip1Oeo2812wFOqDAdw04wLP0n/mr3jEGNJ11f5Xen
9hkWGVkMfvI2A2DdCbdRwPhXN3Z1RSKywgYJjf0kf1urMsSh8TfuOPI2fuu232y9
zkauiaaSAGGC9NAGv2a6UsnY/YUPujlGoIHgXPpc4thimPIZwaqUg1UhDX6bYFCN
OtBg6iIUB4TpYNAtNtpvxOvHZ8x4qwkIvTgQL4R4mBbVxMclPe+slEs7UbWrgYod
ERWB4WwGor+3XvzenXbgiX91936AFIGrBhmPxPOSPQT/ofBecgGTuwUPUH2wNWVc
q2HAT62hHhz+4of13MVEUnpGBc59NwRovrmNrtiI8gLv/Dnp98oVQLmJnTwRl849
+eiEExcVyl18pw33j3ntvjiKZuaITrCrQdGhMSN9jTy8ciKg4rOSzeKszFNjCnFD
mVNcDwMDFGVA9cgDSq9Stt5okO+PSaq5yVM6mCnqJaHeS2zbD24Egy+64r6lSCXI
JF0n9u7Z8VLKeQ/9CKp0noRKrABCzxaN0OBK5Ma84RjvoaKGyuSU8HNn15qqOrHd
dkhVLkNIT15PRRUbxbvlfPtqL+eMIihWLyEWKmp+AYOLQUqSfWY2TgG+zfib7OBb
etxJC5O0XgT3IFhZKYRaJKQa36J7Ag4qe5aJB2+UT556uyaOBrm7CtcdD5TlDHwO
H9eVd0mGMpkz+VQhoUoj5Hp4gPW24jUrAh/Owb7VHjI+f9BhLW39JVauxijB0zQn
zYkksXEk8tUZao7Cfcvaj9kDYn3qrKK3t+n4KrjgxxqLU2YdwW6IWVgZXfAvzEah
MvQFdn+k9b+ITNYlUl2jg1wEIYQ2Wp6TcCEqD4OGEsHLMU8IQLfWq0EK2mOlDoPM
682im648nyHOqtn0LduuppgvyzOTKSWV5qln2+dmSeOJzloxSmhxL912csnWPhL8
IHWFeAd+fw+nqn0UvIBMceG+YF37uD93TdqHQv0hNY8pmcjUl40EGfyBMjN/7sCu
rPGqqdpIgEnJ4j1WgJeV39zl6x61Jyg8JYKrQqbE16XaVvlpsn+LmeILDxva0Isj
wJxPKz8WYEcXvdWgZvD8b7XoK8Nqkw+cKO5WKjdjXhkAGazxIoaOK/Egc0XzsG6S
hkJWDdsIpP6AmfXmnGfJcylRzZckFzrGK3dnQGyB8CW5+tiSQg6HSXJLWKkrvT2x
e6UscsBBZWfmkc8D7r6HzBX+N5F5bhJBs2N6vmhvW5SjbZoBNMBBtnsT5DrpkD2A
Samf79BQaXY98mpQt9q3poGYfFwmgu2xngMzITZ4YL31rg81oV7k1/+2IS5Jk3t9
DjNZX34GHhksrmUT8yEu2CtcR7oavsjOm37oE+UQ0Ng=
-----END RSA PRIVATE KEY-----
```

I have chosen this non-standard key size based upon recommendations from the US National Institute of Standards (NIST). NIST uses a "general

number field sieve" on page 92 of its implementation guidance document to determine minimum RSA key size (http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf). You can implement this formula with the GNU bc utility (part of GNU Coreutils):

```
$ cat keysize-NIST.bc
#!/usr/bin/bc -l
l = read()
scale = 14; a = 1/3; b = 2/3; t = l * l(2); m = l(t) # a^b == e(l(a) * b)
n = e( l(m) * b ); o = e( l(t) * a ); p = (1.923 * o * n - 4.69) / l(2)
print "Strength: ", p, "\n"

$ echo 2868 | ./keysize-NIST.bc
Strength: 128.01675571278223
$ echo 7295 | ./keysize-NIST.bc
Strength: 192.00346260354399
$ echo 14446 | ./keysize-NIST.bc
Strength: 256.00032964845911

$ echo 2048 | ./keysize-NIST.bc
Strength: 110.11760837749330
$ echo 2127 | ./keysize-NIST.bc
Strength: 112.01273358822347
```

In general, asymmetric ciphers are slower and weaker than "symmetric ciphers" (which are defined as using only one key to both encrypt and decrypt). Later I will be using a 128-bit symmetric cipher to communicate the bulk of my data, so I will use an RSA key of (strictly) comparable strength of 2868 bits. Note that many people feel that RSA key sizes over 2048 bits are a waste (https://gnupg.org/faq/gnupg-faq.html#no_default_of_rsa4096). Still, the most forward thinkers in cryptography conjecture that there may be "…some mathematical breakthrough that affects one or more public-key algorithms. There are a lot of mathematical tricks involved in public-key cryptanalysis, and absolutely no theory that provides any limits on how powerful those tricks can be.…The fix is easy: increase the key lengths" (https://www.schneier.com/blog/archives/2013/09/the_nsas_crypto_1.html).

For highly sensitive information that must be kept secret, consider an RSA key size of 7295 or 14446 bits as (strictly) recommended by NIST's formula.

In any case, I am strictly following NIST's recommended guidelines as I generate the key.

I have listed 192- and 256-bit equivalences because this symmetric cipher is not approved for "top secret" use at 128 bits. For highly sensitive information that must be kept secret, consider an RSA key size of 7295 or 14446 bits as (strictly) recommended by NIST's formula. Note that an RSA key size of 2048 bits computes to 110 bits of equivalent strength. This is below the requirement of RFC-7525 (https://www.rfc-editor.org/rfc/rfc7525.txt) of a minimum of 112 bits of security (128 recommended)—2127-bit RSA keys satisfy this mandate.

A corresponding public key can be generated for use with the script:

```
$ openssl rsa -in ~/.prv.key -pubout -out ~/.pub.key
Enter pass phrase for /home/ol7_user/.prv.key:
writing RSA key

$ cat .pub.key
-----BEGIN PUBLIC KEY-----
MIIBiDANBgkqhkiG9w0BAQEFAAOCAXUAMIIBcAKCAWcKpAcsnLXxoH4+ed2Bof2I
upOEwTYdz+N5R++7D/0Eo1LJKrq7CUq6D7jEjeBc/7Wr8mvvBVDgxi4eoYVpbaQa
NgTn1OSa7V7HH0DPVVjXfpIfF6qgk5R98L1Tyqz2agR3GF6F6QL+cxAscl0uFU2g
b/m66VHvxPVwi9ood20aPzBO6e01C6/l6l1tUMaS7PllQdFIXQe0i8ooAtEpvK5D
uBMebUjK0NjPsYxLSQJvJkNW1Sx2KBbIRKFEWPBZ0tFZ8PNokjez2LEV+CaX3ccc
tmeMvdg+w4PwuKmnWxCq0inFlDBE67aTMuYD8Wq7ATxtkkuc2aYL52jfD5YfTCkY
N41aH2w9ICTsuoVNfMUBJRtbhA0w7uoxkWnV2/a6N7VLCbeJncDaNABiOsn80MzY
bfJVrTHVqS0wPt3LY2Pt6/ZjQUejQwhKCjzgqx5DvzgGuTck3J0akhUvTe79OoCC
ZSeanYhX5QIDAQAB
-----END PUBLIC KEY-----
```

The private key is compatible with the OpenSSH Protocol 2 RSA format, and you can generate what normally would be stored as the id_rsa.pub file with a simple keygen command:

```
$ ssh-keygen -y -f ~/.prv.key
Enter passphrase:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABZwqkByyctfGgfj553YGh/Yi6k4TBNh3P43lH7
7sP/QSjUskqursJSroPuMSN4Fz/tavya+8FUODGLh6hhWltpBo2BOfU5JrtXscfQM9VWNd+kh
8XqqCTlH3wvVPKrPZqBHcYXoXpAv5zECxyXS4VTaBv+brpUe/E9XCL2ih3bRo/ME7p7TULr+X
qXW1QxpLs+WVB0UhdB7SLyigC0Sm8rkO4Ex5tSMrQ2M+xjEtJAm8mQ1bVLHYoFshEoURY8FnS
0Vnw82iSN7PYsRX4Jpfdxxy2Z4y92D7Dg/C4qadbEKrSKcWUMETrtpMy5gPxarsBPG2SS5zZp
gvnaN8Plh9MKRg3jVofbD0gJOy6hU18xQElG1uEDTDu6jGRadXb9ro3tUsJt4mdwNo0AGI6yf
zQzNht8lWtMdWpLTA+3ctjY+3r9mNBR6NDCEoKPOCrHkO/OAa5NyTcnRqSFS9N7v06gIJlJ5q
diFfl
```

An SSH server also runs with several types of host keys (which do not normally use a password), usually of 2048 bits in size. A host's private RSA key can be used with my crypter script by generating a compatible public key with this command:

```
# openssl rsa -in /etc/ssh/ssh_host_rsa_key -pubout
writing RSA key
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuxg2zI4ANHRCp+roqjnb
z/h6dc/ijs8uEwXnXE9mID02QvzusciQeeBUcRPXU5ncdPMzNuhUeiNK9y02vs9G
MzkV8vxciBGe6ovFERIDuE1QQPR3V1wZwsVjnG+65bxmGp5/OZpgE4WzMaMm3gla
iDnhfMUllUVzErNoMnR5yCQaoIW9j/AUiBtAymQ07YJcuVrxXBjzGWc/7ryHU1KH
IxKUJfwOhdgf81l0YNpoPdyImCV8PQdBIi8kTnuUl2hIPV2mOP3KWtINfOd94OLM
qfXd5F9LKkKW4XH55wfmJBsO6DTwhzGI9YOayGVJhdraOk7R84ZC/K4rt5ondgpO
3QIDAQAB
-----END PUBLIC KEY-----
```

Since I will be using this RSA keypair for batch transfers, I will be recording the clear-text password for this key in the ~/.pas file. Because of this, the RSA key likely should not be used for SSH. OpenSSL is able to read passwords from a variety of other sources, so if you remove the

~/.pas file and supply the password from a more secure source, the use of a single RSA key for both SSH network sessions and OpenSSL flat file encryption becomes more of an option. Alternately, use a key without a password, and dispense with the `${PW}` clauses above.

You cannot use the RSA keypair for the bulk of the encryption of a large amount of data, because RSA can encode only small amounts of information, as is detailed in the manual page:

```
$ man rsautl | col -b | awk '/NOTES/,/^$/'
NOTES
  rsautl, because it uses the RSA algorithm directly, can only be used
  to sign or verify small pieces of data.
```

Echoing a string of zeros to a text file, the maximum size of the clear-text input for RSA encryption with my 2868-bit RSA key is 348 bytes:

```
$ for((x=0;x<348;x+=1));do echo -n 0 >> bar;done

$ ll bar
-rw-rw-r--. 1 ol7_user ol7_user 348 Jul  7 17:49 bar

$ openssl rsautl -encrypt -pubin -inkey ~/.pub.key -in bar |
     openssl base64
BCfCA77mmbaLCsMQVFCw/uMYWI0+4FaK6meFuTL2OXP6neGa0elrszbAePeoCA/x
dMykxgYBFa/uM2nJl9vagKOlU+DAlRojWGAjrCqfF9XNhdnOjsNINsgNTTzKlVxh
aLfEMYB+vyIwWdaKTrpTz/v7wB20wL9l7eewLZh9yNy4tzyE83Tt5zsgWCvxIdLN
cqkZw7aHvXuXMzdNZn0PoQV/VKLvlmJU5IpDxUCcfPnvZd//f5Akb0tKO44x9hpz
jp/DhRqOYEaB67k5U8GZWYPZoy0XCfLAtSaLMnAkw6swqikVm1IDmLzsRsURgyGX
Qafbh4F33ivn7jaRNbSKbFmSMYc1ShACJuTgTQ2N519gc84Sd1TvSyL7v+m5WqXF
fuPJiIrpi6DkYZDOuNQP0cjEMVHLVuwjFh98uW7IyJY5sGVP+/cVlmVg9SUDhpNt
t6naZ/CwkyHal6PaFa4AhlDGNJ/RVNc=

$ echo -n 0 >> bar

$ ll bar
-rw-rw-r--. 1 ol7_user ol7_user 349 Jul  7 17:49 bar
```

```
$ openssl rsautl -encrypt -pubin -inkey ~/.pub.key -in bar |
     openssl base64
RSA operation error
139936549824416:error:0406D06E:rsa routines:
RSA_padding_add_PKCS1_type_2:data too large for key size:
rsa_pk1.c:151:
```

Similar testing with a 2048-bit RSA key yields a maximum of 245 bytes (slightly smaller than the size of the key).

Because of this limitation, I will generate a random 64-character password with OpenSSL's random number generator, encrypt a copy of it in the output with the public key, then use it for symmetric encryption for the bulk of the data as the "session key". For this example, I will use the following password obtained from OpenSSL in this manner:

```
$ openssl rand -base64 48
d25/H928tZ1BaXzJ+jRg/3CmLYxaM5kCPkOvkIxKAoIE8ajiwu+0zWz0SpDXJ5J7
```

If I store this file as /tmp/skey, I can see the encryption take place:

```
$ openssl rsautl -encrypt -pubin -inkey ~/.pub.key -in /tmp/skey |
     openssl base64
Ac5XfYjJUpJGRiCNVSPcRi7SBrEVBtQhVHgqYWgQH6eFrDuQLX4s/S50qKt1ObjT
17aV8pDMGqiHXOsbfD/P/GBpiymgQUJoa4VS40J+d5u9X20NmxmtNAvvlklmCC9q
lzJcX6QXg4QEDTOHD+jU0B3K5QOB3von0IIVgauKGfDvgkOJiqjK9bUhhSgdnNe3
yyivWXb8Xl+zDCSqtqtv0Xkzri2jmTXniu7HztGTnyOcpZ4PLFMT9ZC0Biu40xK9
ubuMPcfpVKVKRuR0iAu1kkstQY2k6xieZiIDIMtg4vHJIdb793aC8Spuhjca1puS
QaQTfkQIrN46oJ6IoGqmTMGem6IGiUAldan24nTl7C+Z7aF1nieXb55gDwfQcO55
Uk/1tbgQR6MMzXG6BglmjD6oa/urKjI2taJT02c+IT6w6nXpGWrGBMY5S7G8u++Y
tml7ILPwiA4lKhvukgbPZw/vFgNAGxo=
```

Note above the call to the base64 function—the encrypted output is a binary file, and it cannot be displayed directly with the standard ASCII seven-bit character set. The `base64` encoder changes binary data into

a stream of printable characters suitable for transmission over seven-bit channels—email, for example. This performs the same function as uuencode, using different ASCII symbols.

If I record the output of the encryption in the /tmp/ekey file, I can decrypt it with the private key:

```
$ openssl base64 -d < /tmp/ekey |
     openssl rsautl -decrypt -inkey ~/.prv.key
Enter pass phrase for .prv.key:
d25/H928tZ1BaXzJ+jRg/3CmLYxaM5kCPkOvkIxKAoIE8ajiwu+0zWz0SpDXJ5J7
```

Note above in the decryption section that very old versions of the OpenSSL `rsautl` command did not allow passwords to be specified on the command line. Therefore, an unencrypted copy of the key must be created before RSA decryption of the session key can take place. That procedure is documented in the comments for legacy systems and versions.

With the session key in hand, I next compute SHA-256 digest checksums of all the input files and record the encrypted results in the output. OpenSSL's version of the `sha256sum` utility differs slightly in formatting from the conventional version. Also included are SHA-1, RIPEMD-160 and MD5 checksums below:

```
$ sha256sum /etc/resolv.conf
04655aaa80ee78632d616c1...4bd61c70b7550eacd5d10e8961a70  /etc/resolv.conf

$ openssl dgst -sha256 /etc/resolv.conf
SHA256(/etc/resolv.conf)= 04655aaa80ee78632d6...1c70b7550eacd5d10e8961a70


$ openssl dgst -sha1 /etc/resolv.conf
SHA1(/etc/resolv.conf)= adffc1b0f9620b6709e299299d2ea98414adca2c
$ openssl dgst -ripemd160 /etc/resolv.conf
RIPEMD160(/etc/resolv.conf)= 9929f6385e3260e52ba8ef58a0000ad1261f4f31
$ openssl dgst -md5 /etc/resolv.conf
MD5(/etc/resolv.conf)= 6ce7764fb66a70f6414e9f56a7e1d15b
```

The SHA-family of digests were all created by the NSA, to whom we owe a great debt for their publication. The RIPEMD-160 digest was developed by researchers in Belgium and is an open alternative to SHA with no known flaws, but it is slower than SHA-1 and was released afterwards, so it is not used as often. MD5 digests should not be used beyond basic media error detection as they are vulnerable to tampering (http://www.mathstat.dal.ca/~selinger/md5collision).

The script adjusts the format produced by OpenSSL to more closely mimic the standard utility, then uses the AES128-CBC symmetric cipher to code the digest for all the input files after printing a delimiter (`__:`). Very old versions of the OpenSSL utility might lack SHA-256—notes in the script detail downgrading to the weaker SHA-1 when using legacy systems (MD5 never should be used). The `man dgst` command will give full details on OpenSSL's digest options if the manual pages are available.

Finally, the script enters the main encryption loop where each file is processed with AES128-CBC, encoded with base64, separated by delimiters, then sent to STDOUT under the intention that the script be redirected/piped to a file or program for further processing. Information on OpenSSL's various symmetric ciphers can be found with the `man enc` command when the manual pages are accessibly installed. An informative and amusing cartoon has been published online covering AES' history and theory of operation, for those who have a deeper interest in our chosen symmetric cipher (http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html). The GPG website currently advocates Camellia and Twofish in addition to AES, and Camellia can be found in OpenSSL.

OpenSSL can be called to encrypt a file to the standard output with AES like so:

```
openssl enc -aes-128-cbc -salt -a -e -pass file:pw.txt
 ➥-in file.txt > file.aes
```

The encryption is undone like so:

```
openssl enc -aes-128-cbc -d -salt -a -pass file:pw.txt -in file.aes
```

Here is an example of a complete run of the script:

```
$ ln -s crypter.sh encrypter
$ ln -s crypter.sh decrypter
$ chmod 755 crypter.sh

$ ./encrypter /etc/resolv.conf /etc/hostname > foo

$ ./decrypter < foo
2016/07/05:21:24:38
04655aaa80ee78632d616c1...4bd61c70b7550eacd5d10e8961a70 /etc/resolv.conf
4796631793e89e4d6b5b203...37a4168b139ecdaee6a4a55b03468 /etc/hostname
resolv.conf: ok
hostname: ok
```

To use this script, or otherwise use the OpenSSL utility for secure communication, it is only necessary to send a public key to a distant party. Assuming that the integrity of the public key is verified between the sender and receiver (that is, via an SHA-256 sum over the phone or another trusted channel), the sender can create a session key, then use it to encode and send arbitrary amounts of data through any untrusted yet reliable transfer medium with reasonable confidence of secrecy.

Note that the decryption block uses shell arrays, which are limited to 1024 elements in some versions (ksh88, pdksh). That will be a hard file limit in those cases.

This entire script can be worked into an email system for automated transfers. To do this on Oracle Linux 7 with the default Postfix SMTP server, ensure that the following two lines are set in /etc/postfix/main.cf:

```
inet_interfaces = $myhostname, localhost
...
default_privs = nobody
```

Here I will place a copy of the SSH private RSA host key in the /etc/postfix directory, set the configuration and permissions, open

firewall port 25, then generate a public key as outlined below:

```
cd /etc/postfix
cp /etc/ssh/ssh_host_rsa_key .prv.key
chown nobody:nobody .prv.key
chmod 400 .prv.key
chcon system_u:object_r:postfix_etc_t:s0 .prv.key
iptables -I INPUT -p tcp --dport 25 --syn -j ACCEPT
openssl rsa -in .prv.key -pubout -out .pub.key
```

Notice that I'm using the nobody user with the system host key. If you are not comfortable with this security, note that the key file is in the ssh_keys group, and create a separate user for postfix to handle the keypair.

Next, place a copy of decrypter in /etc/postfix. The script must be modified to do the following: 1) skip the email header, 2) remove the password clause from the host key processing, 3) set /tmp as the unpack directory and 4) define new locations for the keypair. Below, `sed` is used with in-place editing to accomplish this:

```
sed -i.old '/^ while read Z/s:^:sed '"'"'1,/^$/d'"'"' |:
     s/^[ ]*-passin "[^"]*"//
     /^ DGST=/s:#.*$:cd /tmp:
     /^PVK=/c \
PVK=/etc/postfix/.prv.key; PBK=/etc/postfix/.pub.key' decrypter
```

With those changes in place, I create an email alias that will trigger the decrypter:

```
echo 'crypter: "| /etc/postfix/decrypter >> /tmp/cryp.log 2>&1"' \
     >> /etc/aliases
newaliases
chcon system_u:object_r:postfix_local_exec_t:s0 decrypter
postfix reload
systemctl restart postfix.service
```

Now, pipe the encrypter output to the mail client:

```
cd /etc
encrypter resolv.conf hostname | mail crypter@localhost
```

The files sent into the mail client should appear in /tmp. Move the public key to a remote server, and automatic encrypted file transfer over SMTP is established.

It is also possible to work RSA encryption in reverse, decrypting with the public key. This is useful in establishing authenticity of data—for example, to encrypt a small amount of clear text (bounded by RSA length limitations) with the private key:

```
echo 'I have control of the private key.' |
  openssl rsautl -sign -inkey ~/.prv.key -passin "file:$HOME/.pas" |
  openssl base64 > blob
```

The blob file then can be posted in a public medium (website, file server and so on), and holders of the public key can successfully decrypt the message like so:

```
openssl base64 -d < blob |
     openssl rsautl -inkey ~/.pub.key -pubin
```

In doing so, users verify that the private key was involved in the creation of the message, lending some authenticity to the data that has been transferred. The public key is not assumed to be secret, so this establishes data authenticity, not data privacy.

Rather than arbitrary text, you can pipe in the text from an SHA-256 signature program call, and thus "sign" a larger file in a way that proves authenticity:

```
openssl dgst -sha256 crypter.sh |
  openssl rsautl -sign -inkey ~/.prv.key -passin "file:$HOME/.pas" |
  openssl base64 > csign
```

You decrypt this text in exactly the same manner as you did before, producing an SHA-256 clear-text digest that you can verify independently. However, OpenSSL can summarize in one step the signed SHA-256 checksum (note that full x.509 keys also can be manipulated to sign a digest):

```
openssl dgst -sha256 -sign ~/.prv.key \
     -out crypter.sha256 crypter.sh
```

If the two files above are placed accessibly, holders of the public key can verify that the files have not been altered:

```
openssl dgst -sha256 -verify ~/.pub.key \
     -signature crypter.sha256 crypter.sh
```

OpenSSL should output "Verified OK" when the files are intact. The capability of using an encrypted SHA-256 digest to verify a file securely is far beyond the features of the standard sha256sum utility and demonstrates authenticity unambiguously.

Introduction to GPG

GNU Privacy Guard has much more comprehensive tools for the management of keypairs and peer identities. This includes databases for storing the various types of keys, tools for revocation of keys and mechanisms for establishing key reputation in a "web of trust".

Oracle Linux 7 bundles GPG 2.0.22, which uses the 128-bit CAST5 symmetric cipher by default (newer versions have switched to AES128). Here, I will conform to the previous NIST guidelines for a 2868-bit asymmetric keypair of equal strength (note that the GPG documentation does warn that "Moving past RSA-2048 means you lose the ability to migrate your certificate to a smartcard, or to effectively use it on some mobile devices, or to interoperate with other OpenPGP applications that don't handle large keys gracefully."):

```
$ gpg --gen-key

gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
```

```
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/home/ol7_user/.gnupg' created
gpg: new configuration file `/home/ol7_user/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/ol7_user/.gnupg/gpg.conf' are not yet
      active during this run
gpg: keyring `/home/ol7_user/.gnupg/secring.gpg' created
gpg: keyring `/home/ol7_user/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2868
Requested keysize is 2868 bits
rounded up to 2880 bits
Please specify how long the key should be valid.
         0 = key does not expire
        = key expires in n days
      w = key expires in n weeks
      m = key expires in n months
      y = key expires in n years
Key is valid for? (0) 5y
Key expires at Sat 10 Jul 2021 08:40:19 PM CDT
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Oracle Linux
Email address: ol7_user@localhost
Comment: Test Key
You selected this USER-ID:
    "Oracle Linux (Test Key) <ol7_user@localhost>
```

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/ol7_user/.gnupg/trustdb.gpg: trustdb created
gpg: key 6F862596 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2021-07-11
pub   2880R/6F862596 2016-07-12 [expires: 2021-07-11]
      Key fingerprint = F423 3B2C ACE1 AD0E 95C3  4769 679D 66ED 6F86 2596
uid                  Oracle Linux (Test Key)
sub   2880R/FF79FC31 2016-07-12 [expires: 2021-07-11]
```

Once the (rounded-up) 2880-bit private key has been created, a command is needed to generate a public key that can be shared with others:

```
$ gpg --export -a

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQF1BFeESqMBC0C7mB+Arj5aWfOF8Ald3TGBjBXUGZcZ5SObYSifDf+OwwBUGHEE
7eP5al3PySCUqFM/fsTEWFDg4AeuZYcTQ/4qzaYu05SLbDZeZSuTm9HM9SkpGu11
gTlYMYese9y5luxCHpnq0F1tj12+r66e7txIlQLr8j7A0o4zz/C6ki5unWGHNP/r
/xVspC3NpNcxvnU/XUPjVutkeb9lGte4rYkwKRUmrSG1yNfRdnTVeMQTae6QXeL/
NAYidjJW4ds2UU8lks15KkWXj87CljI2MxnCZmv915k0ibYX1f631kettACCoV8u
jmMtn+1ahJOxsduDe1NLI0bfGoeP3eiOHjD6W8iBhPtOFQEeb9TqJmA7xFjSIpVE
bDGq17ijEkczm+Bj15GZ44UCymJDQLBCUzoE5Al5s5BUAxr+Z/c8nW5ZPJpDUjDZ
```

```
1rkXr+Y6qE65tSplbGrlkq/vnqkKbpuB7aFA+uZiBeRfpTkAEQEAAbQsT3JhY2xl
IExpbnV4IChUZXN0IEtleSkgPG9sN191c2VyQGxvY2FsaG9zdD6JAacEEwECACkF
AleESqMCGwMFCQlmAYAHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIXgAAKCRBnnWbt
b4YllmimC0CEBI4F2VKV6NeyQ1WZYMp78jojkQwV8ERas/cPLjpdQM2lbaZ99yoA
5Ip7uvPT3y7CZfaWew1rV1eMvZdmgQ3H9rQC2sYKDh0Rvft1BJSkv4fJ9GcRREND
jahlMA7hP/bx5RI/LxvNKJUEOdZ2gVV1ux7glT0/lr9WMQxcDIjKoa5C5zTs9DmZ
76pZE9Pv3EHd0WxU6YKHQUf25Bd/Y7kpwVxkdJrm294R2HdXBs0BzHx061O8H01o
UzVdbQ8LDsPKv9je6wdmy3Olf7xRfUnG8FelLdeAyrttkQNJPIbVCeEKIsQoDamb
TnHKzSWCre/ii0lpwCCUJveYtUb746QkpRd2Y7PDCBi1mG1sPPayK64ee4B3m0NH
JXoc/ivFP55Xaqmvz41QM4DRyK+g2JBjYkj7X8Fo38QgKWmOrVw/YU/OLm8EWtrt
sHYaelJSkjtf0OZeGrlqHWECSWfVDy9jp2BoQTLUlsm5AXUEV4RKowELQLU+3B/T
tPEzVeigql/P/34Q80lgQpG2Nfo6VwxCajDEofSzJVEWnT6/CrWJ91NrLr7QNV62
AbxIIoZt06vZGN7pnxl4vIsgn4R5XswehXkh8HOwJ5eVtYEOozul7e0eegPhu8CP
wHlEc/2Uc1RIT1HxwWGs0Vlp0BxcRtubU15vaCOoM1Gd4zExzl7KSocLgEuNnl56
4t5JcCfOBbSi0TTR69xIuXhwCLIps0j6fnMh6Bh+Uev0cTwFlLNBe0X3TNE0V0be
Y3AmV8ZVnaQ3oZkm8XO4fopW+9/rs48qG1GF7NBKvsbQAJx0MzbOvXp0OELR/6sq
/2NxafX5L3fseXEnje5Ks2yam9oVX13dKT4hO97UZ7aL25z3LYJnhl52LX8gscv+
kIki/vxvQbDbJLdDFuljysf36FCucUHvNysdv8JpJ0cTJqx2d3JUNdvhS89NScSB
EDmsIXF2Ij7ptRalwibCUC2wwwARAQABiQGNBBgBAgAPBQJXhEqjAhsMBQkJZgGA
AAoJEGedZu1vhiWWeKwLQKz04zGJM1Sa20SJ9H39Hts+IL4NZYk1Kf5qRQ2RDjXX
dHOpfzOBZUan1CsBoghxZ+9BI6GWs9Mr76OwZwGU+810vRMqe6Z0/N1DaG4eX4UU
N0PVcMRf6y+cl7sxVrWq9YppZXHzt2PkwP1JTU0dIHnHcX64LgYpKObiM7FFJ2xf
HTTF3PzRH5hiKOqMJhaRlA4Gu93uv4I7KT1LxVtnmN2u55zmzl1VzD/l7RtEavmX
0K7UwBzlzqpVyHQF0TH41WDnJqv9CwVUoIQ0Z6JldCCkhNiCL12szYJ2CCbXQ7H0
hZKVQNAikOXlimtp2taAnyRNxdKrUaNYp5UmZ4lTHroTdKXqwRvv+Z7dHbGc3V7s
Cn4avsvpuhl5NDFQrLRwrKA4ycIhTE1OhhSlumLpiv1di2CcmOHzaNrIkWCyj0m0
4oJKTUrjHnYp+PMvOJU4tU9B2uXA1+M8m2lPygxwc3whqaP0nqYusg==
=gBp9
-----END PGP PUBLIC KEY BLOCK-----
```

This key typically would be included in a signature at the end of email messages, bundled with software or documentation that requires either privacy or authenticity, or pasted in a public forum (such as a website) where similar activities might take place.

Other GPG/PGP users that desired to communicate with the originator of this key might then import it, prior to engaging in

There are many email packages that will use various PGP components directly, enabling integrated cryptography.

these activities:

```
$ gpg --import /tmp/test.pub
gpg: key F3BD3FF5:
public key "Test User (Test Key) <testuser@localhost>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
$ gpg --import /tmp/ol7.pub
gpg: key 6F862596:
public key "Oracle Linux (Test Key) <ol7_user@localhost>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

There are many email packages that will use various PGP components directly, enabling integrated cryptography. My focus here is flat file encryption, so I will confine my GPG demonstration to this specific action and use it to encrypt the script from the last section, sending from ol7_user@localhost to testuser@localhost:

```
$ gpg -u ol7_user@localhost -r testuser@localhost --armor
 ➥--sign --encrypt crypter.sh

You need a passphrase to unlock the secret key for
user: "Oracle Linux (Test Key) "
2880-bit RSA key, ID 6F862596, created 2016-07-12

$ mv crypter.sh.asc /tmp
```

```
$ head -5 /tmp/crypter.sh.asc
-----BEGIN PGP MESSAGE-----
Version: GnuPG v2.0.22 (GNU/Linux)

hQF0A05zbjK/t9mRAQs/fog4FSkocxnJBKp1hb64yGf1xiecqLWwZBqct3kLiU5e
Ekmqdt06E+XU4N3bMtt808SwSXSLvKWT18Iy6WtGz4r+B3dYAlHo1vfeSt3L5dE0
```

The recipient (testuser) is then able to log in and decrypt (which will go to the standard output by default):

```
gpg -d /tmp/crypter.sh.asc
```

Any activity that causes GPG to request the password to a key will spawn an "agent" that will tie future GPG sessions and supply credentials so the key password need not be entered repeatedly:

```
testuser 4252 0:00 gpg-agent --daemon --use-standard-socket
```

The holder of a GPG private key also can sign fies digitally in a manner similar to OpenSSL (but somewhat more flexibly). There are three methods to add signatures: create a compressed binary file that contains a packed copy of the original message, add a clear-text "ASCII-armored" signature that allows the original content to be read, or write a binary signature to a separate file (requiring both a clean file and signature to validate). The first method writes a compressed binary to a new file with a .gpg extension:

```
gpg -s crypter.sh
(or)
gpg --sign crypter.sh
```

The second method will add a clear-text signature, allowing the original content to remain visible, into a new file with an .asc extension:

```
gpg --clearsign crypter.sh
```

The third will write a binary signature to a separate file with a .sig extension:

```
gpg -b crypter.sh
(or)
gpg --detach-sign crypter.sh
```

All of these methods can be verified by holders of the public key with the `gpg -v (file)` command, where `(file)` points at the output of GPG.

Although GPG has the ability to support many types of digests and ciphers, forcing specific algorithms can cause compatibility problems with users of various distributions and versions of PGP software. It is wise to adhere to the capabilities of general versions, rather than specify algorithms directly (this discussion can be found in the `man gpg` pages):

```
man gpg | col -b | awk '/^INTEROPERABILITY/,/reduce/'

INTEROPERABILITY
  GnuPG tries to be a very flexible implementation of the OpenPGP
  standard. In particular, GnuPG implements many of the optional
  parts of the standard, such as the SHA-512 hash, and the ZLIB and
  BZIP2 compression algorithms. It is important to be aware that
  not all OpenPGP programs implement these optional algorithms and
  that by forcing their use via the --cipher-algo, --digest-algo,
  --cert-digest-algo, or --compress-algo options in GnuPG, it is
  possible to create a perfectly valid OpenPGP message, but one
  that cannot be read by the intended recipient.

  There are dozens of variations of OpenPGP programs available, and
  each supports a slightly different subset of these optional
  algorithms. For example, until recently, no (unhacked) version
  of PGP supported the BLOWFISH cipher algorithm. A message using
  BLOWFISH simply could not be read by a PGP user. By default, GnuPG
  uses the standard OpenPGP preferences system that will always do the
  right thing and create messages that are usable by all recipients,
```

```
regardless of which OpenPGP program they use. Only override this safe
default if you really know what you are doing.

If you absolutely must override the safe default, or if the
preferences on a given key are invalid for some reason, you are far
better off using the --pgp6, --pgp7, or --pgp8 options. These
options are safe as they do not force any particular algorithms in
violation of OpenPGP, but rather reduce the available algorithms
to a "PGP-safe" list.
```

GPG also has the ability to be used non-interactively with the `--batch` and the various `--passphrase` options. It is likely unwise to use the same keys for both interactive and batch activity—use an email key for online communication and a batch key for automated activities. GPG offers several options for key revocation—be ready to use them for any key that is compromised, especially automated keys.

Conclusion

OpenSSL flat file use might be preferable to network services like TLS (or even SSH) for several reasons:

- Removing TLS vastly reduces the attack surface of a server.
- When an encryption process takes place offline and is not visible in action from the network, several classes of exploit are removed or greatly reduced in scope: timing attacks (such as Lucky Thirteen), other side-channel attacks (such as CRIME), and versioning attacks (such as DROWN).
- Cipher algorithm code within OpenSSL is used in OpenSSH, which attests to quality. OpenSSH reviews are extremely thorough, and the security record is quite good.
- One of OpenSSL's aes_core.c authors is Vincent Rijmen, who developed AES with fellow cryptographer Joan Daemen (although custom high-speed assembler code is substituted on architectures where it is available). Fragments of the aes_core.c code also are found in the

libtomcrypt library that is used directly in the dropbear SSH server, which I discussed in a previous article (see "Infinite BusyBox with systemd" in the March 2015 issue: http://www.linuxjournal.com/content/infinite-busybox-systemd).

- OpenSSL's support for exotic systems introduces more problem code for networking than for basic math.

- Far more time is spent in code reviews for OpenSSL's basic cipher algorithms than for the networking features. Merely the legal analysis of source code for the question of patent infringement can dwarf network security reviews (for example, Red Hat's recent decisions on Elliptic Curve within OpenSSL and Sun's careful coding of said routines to avoid existing patents). It was unlikely that the DTLS heartbeat TCP implementation received comparable analysis, and it became the greatest flaw ever found in OpenSSL (which never impacted flat file processing).

- A scripted solution allows easier interfacing to custom programs (new compression tools, alternate data sources, legacy systems and applications and so on).

There are a few drawbacks to using the crypter script as presented:

- The script places delimiters between the content of each file. The number of files sent, and their length, will be known by anyone observing the traffic. Use a ZIP utility to send only one file if this is troublesome—some ZIP utilities use AES directly, allowing an RSA exchange of a ZIP archive's password, then the transmission of the ZIP over unencrypted channels (this might allow the ZIP file directory to be read by observers, even if the file content remains opaque).

- The script will read each file twice—once for the digest and once for the symmetric algorithm. This will cost time, processing power and I/O (GPG does this all in one step).

GPG also has a few concerns:

- Some PGP implementations can have problems with larger RSA keys.
- Compatibility issues between PGP implementations greatly influence chosen digests and ciphers.
- GPG 2.0.22 (the older version found in Oracle Linux 7) uses the SHA-1 digest, which has been deprecated.

None of these tools are perfect, but they are the bedrock of secure communications. To ponder the scale of their influence upon commerce and trusted communication is almost beyond comprehension. These algorithms are as ubiquitous as they are generally unknown.

Hopefully, this tutorial has cast a bit more light upon them.■

Charles Fisher has an electrical engineering degree from the University of Iowa and works as a systems and database administrator for a Fortune 500 mining and manufacturing corporation. He has previously published both journal articles and technical manuals on Linux for *UnixWorld* and other McGraw-Hill publications.

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

RETURN TO CONTENTS

A New Mental Model for Computers and Networks

We've built and re-built centralized top-down systems for the duration. Time for something that's not.

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: Flat File Encryption with OpenSSL and GPG

One of the great works of geekdom is Neal Stephenson's *In the Beginning Was the Command Line*, an essay-length book that came out in 1999 (http://www.cryptonomicon.com/beginning.html). As with Linux, the code was open. Still is. Here's one copy of the book's full text: http://cristal.inria.fr/%7Eweis/info/commandline.html. Though many of Neal's references (for example, the Be operating system) are forgotten or stale, his case for Linux (and its UNIX relatives) is as fresh and right as ever. Here is the gist of it:

The file systems of Unix machines all have the same general structure. On your flimsy operating systems, you can create directories (folders) and give them names like Frodo or My Stuff and put them pretty much anywhere you like. But under Unix the highest level—the root—of the filesystem is always designated with the single character "/" and it always contains the same set of top-level directories:

- /usr
- /etc
- /var
- /bin
- /proc
- /boot
- /home
- /root
- /sbin
- /dev
- /lib
- /tmp

and each of these directories typically has its own distinct structure of subdirectories. Note the obsessive use of abbreviations and avoidance of capital letters; this is a system invented by people to whom repetitive stress disorder is what black lung is to miners. Long names get worn down to three-letter nubbins, like stones smoothed by a river.

> This is not the place to try to explain why each of the above directories exists, and what is contained in it. At first it all seems obscure; worse, it seems deliberately obscure. When I started using Linux I was accustomed to being able to create directories wherever I wanted and to give them whatever names struck my fancy. Under Unix you are free to do that, of course (you are free to do anything), but as you gain experience with the system you come to understand that the directories listed above were created for the best of reasons and that your life will be much easier if you follow along (within /home, by the way, you have pretty much unlimited freedom).
>
> After this kind of thing has happened several hundred or thousand times, the hacker understands why Unix is the way it is, and agrees that it wouldn't be the same any other way. It is this sort of acculturation that gives Unix hackers their confidence in the system, and the attitude of calm, unshakable, annoying superiority captured in the Dilbert cartoon. Windows 95 and MacOS are products, contrived by engineers in the service of specific companies. Unix, by contrast, is not so much a product as it is a painstakingly compiled oral history of the hacker subculture. It is our Gilgamesh epic.
>
> What made old epics like Gilgamesh so powerful and so long-lived was that they were living bodies of narrative that many people knew by heart, and told over and over again—making their own personal embellishments whenever it struck their fancy. The bad embellishments were shouted down, the good ones picked up by others, polished, improved, and, over time, incorporated into the story. Likewise, Unix is known, loved, and understood by so many hackers that it can be re-created from scratch whenever someone needs it. This is very difficult to understand for people who are accustomed to thinking of OSes as things that absolutely have to be bought.

When Tim Berners-Lee invented the World Wide Web, he did it on a NeXT machine, which also had UNIX ancestors. But never mind that. What matters is that Tim arranged the web's directory on the UNIX model: files were nested in directories divided hierarchically by a series

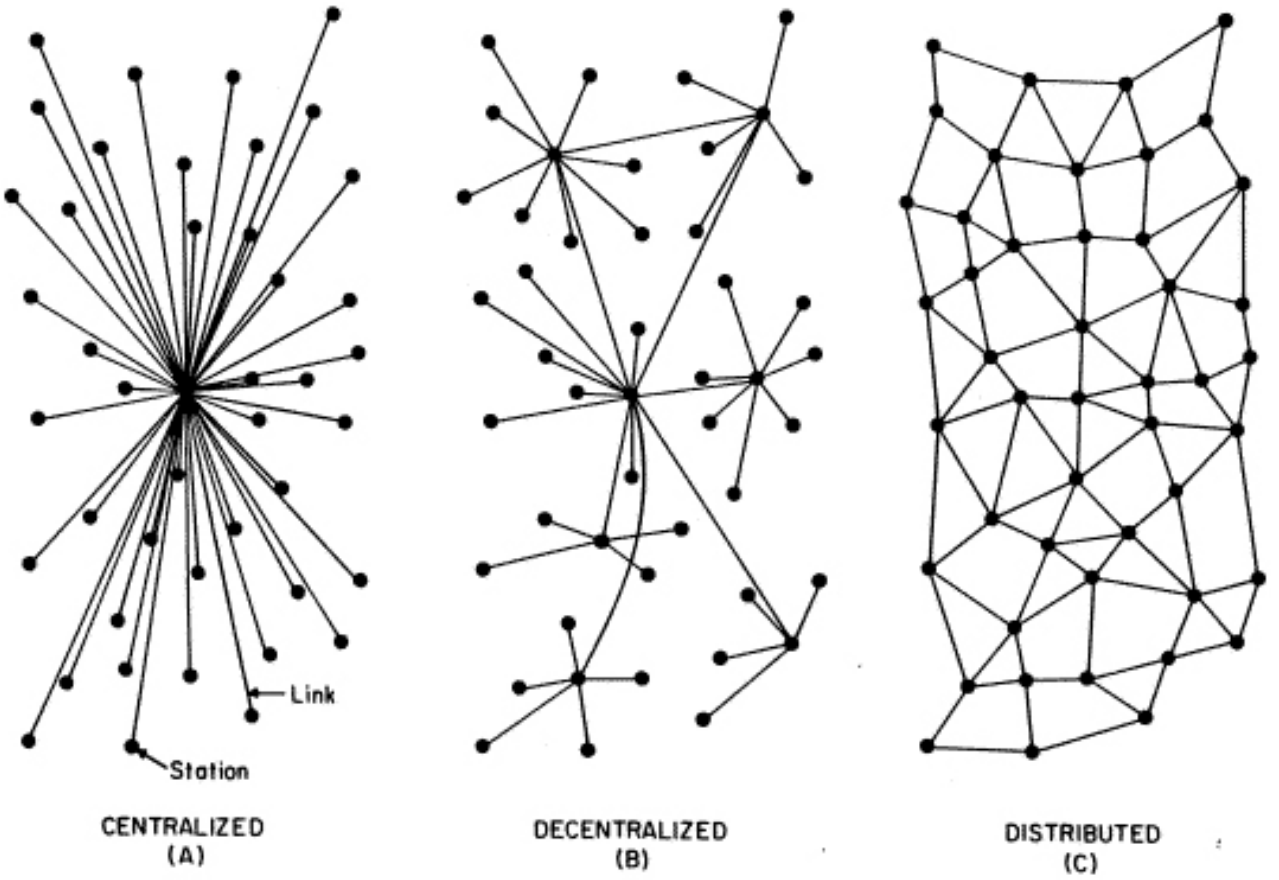


Figure 1. Centralized, Decentralized and Distributed Networks

of slashes: http://example.com/this/that/etc. The same hierarchy is also implicit in client-server, which should have been called server-client, to reflect the arrangement of status: server over client.

Between the directory path and client-server, what we've got is a mental model of computing and networking that anchors all of computing's civilization.

But it also gives us problems when it comes to conceiving and designing *distributed* approaches, such as peer-to-peer. It gives us hierarchy after hierarchy, and hierarchies within hierarchies, rather than the *heterarchy* that Paul Baran imagined the future internet to embody when he drew his diagram in 1964 (Figure 1).

While we're at it, let's also revisit "End-To-End Arguments in System Design" (http://www.ece.ucdavis.edu/%7Echuah/classes/eec273/eec273-w12/refs/SRC84-e2e.pdf) by David P. Reed, Jerome H. Saltzer

and David D. Clark: a design guide (http://web.mit.edu/Saltzer/www/publications/endtoend/ANe2ecomment.html) that helped manifest and rationalize the internet protocol (https://en.wikipedia.org/wiki/Internet_Protocol), which in an ideal world would give us a fully distributed network (C, in Figure 1). Alas, the current internet's implementation is closer to decentralized (B, in Figure 1). As distributed networks go, it's good, but not good enough, because it still subordinates client nodes to server ones, so servers get scale, while clients get little more scale than the servers (and the operators of sphincters in the network itself) allow. We also have a networked world where governments can favor or ban traffic they don't like. Even society itself has to some degree been re-organized and re-centralized by giant private "social networks", such as Facebook and LinkedIn.

Back in April 2016, I suggested that our next fight is for freedom from exactly the kind of centralized systems I just described (http://www.linuxjournal.com/content/whats-our-next-fight). Fighting for freedom would also get us closer to each of these ideals:

- General-purpose computing and networking.
- Decentralization and distributed everything.
- Privacy.
- The true Internet of Things.

Now I suggest that we also need to free ourselves from the very mental models that we used to build giant centralized traps from which we need to escape.

There are positive signs. The blockchain, for all its faults, is distributed by design. To come up with blockchain and Bitcoin (which uses a blockchain), Satoshi Nakamoto (or whoever that really is) had to think outside of fiat currency, banks, centralized trust systems and the other familiar boxes that control transactions in the world's economies, nearly all of which are centralized by design. He had to think of ways that a fully distributed peer-to-peer approach to all those things would open possibilities and

We are embodied animals, and we can't get away from that fact. But we are also inherently distributed, and different. At a base level, we are *heterozygous*.

outperform currency, payments and record-keeping done the old ways. One can criticize Bitcoin and blockchain on many grounds, but what matters is that a distributed system got imagined and implemented causing many other new re-thinks and re-builds in the world.

The problem with mental models is that they work—also that we can't escape the deepest ones that are anchored in our own experience as physical bodies, operating in the physical world.

Ever wonder why good is up (or high) and bad is down (or low)? Why it is easier to conceive of heaven in the sky and hell below the ground, than vice versa? Or why light is good and dark is bad? Or why people say "enlightened" rather than "endarkened?" Or why we "catch" or "grasp" ideas? The answer is, *because we are upright-walking diurnal animals with hands and opposable thumbs*. If owls or moles were equipped by nature with the means to have moral systems and speak about them, their metaphors would be radically different. Dark might be good, and light might be bad.

We are embodied animals, and we can't get away from that fact. But we are also inherently distributed, and different. At a base level, we are *heterozygous*. No two of us are the same, unless we are identical twins; and even then we are separate and distinct individuals. (An interesting fact: so are apples. Writes Michael Pollan in *The Botany of Desire*, every seed in every apple "contains the genetic instructions for a completely new and different apple tree, one that, if planted, will bear only the most glancing resemblance to its parents". All the varieties of Apple we know—Granny Smith, Delicious, Macintosh—grow on trees that start as grafts off a single ancestral plant.)

The designs we need are ones that appreciate our heterozygous inheritances, and the fact that we are designed to learn throughout

our healthy lives. "Encompass worlds, but never try to encompass me", Walt Whitman advises. He adds:

> Urge and urge and urge,
> Always the procreant urge of the world.
> Out of the dimness opposite equals advance.
> Always substance and increase, always sex,
> Always a knit of identity, always distinction,
> Always a breed of life.

I love how Whitman puts those together, because none of them fits in a system, other than one even he fails to comprehend, even as he embraces its mystery. I also love "knit of identity", because each life is a thread distinct in its substance and capacity for increase, yet part of a whole that changes as well. Every self, like every species, is a breed of life.

It is hard for computing to comprehend this, but not for the minds of people programming and using computers.

Computing and programming require that we think of both in explicit ways, and in explicit terms. Yet our knowledge of the world is mostly tacit. "We know more than we can tell", Michael Polanyi says, and that's a near absolute understatement. It applies to everything we think and say. For example: even if I've made full sense to you in this column so far, you probably won't be able to repeat it back to me verbatim. And if you could, it would owe more to memorization than comprehension. Short-term memory is an amazing grace of human nature. It forces us to communicate meaning more than words. Consider how often, in the midst of explaining something, we don't remember exactly how we started the sentences we are now speaking, or exactly how we will finish them, yet somehow we'll say what we mean, and others will understand it, even though they can't repeat exactly what we said.

That's because, when we communicate with each other, we don't deliver an explicit cache of words. Instead we cause meaning to form in the mind of another person. Meaning is most of what we take away from any conversation. The same goes for any course in school, any book or any experience. The meaning we take is mostly tacit. It is also mostly

unquestioned, once we make it our own.

Here's how I put it many years ago in a chapter of *Open Sources 2.0* (http://programmer.97things.oreilly.com/wiki/index.php/Open_Sources_2.0/Beyond_Open_Source:_Collaboration_and_Community/Making_a_New_World):

> Several years ago I was talking with Tim O'Reilly about the discomfort we both felt about treating information as a commodity. It seemed to us that information was something more than, and quite different from, the communicable form of knowledge. It was not a commodity, exactly, and was insulted by the generality we call "content".
>
> Information, we observed, is derived from the verb *inform*, which is related to the verb *form*. To inform is not to "deliver information", but rather, to form the other party. If you tell me something I didn't know before, I am changed by that. If I believe you and value what you say, I have granted you authority, meaning I have given you the right to author what I know.
>
> Therefore, *we are all authors of each other*. This is a profoundly human condition in any case, but it is an especially important aspect of the open-source value system. By forming each other, as we also form useful software, we are making the world, not merely changing it.

So now look at authoring as something all of us do—or should be able to do—all the time, in many more ways and contexts than our pyramidal centralized systems would allow.

Consider the possible purposes of both our heterogeneousness and our enormous capacity to communicate and learn, throughout our lives. Why are we that way? Are those very human natures not insulted by systems built to subordinate individuality to categories in databases? Is the full promise of heterarchy (http://www.linuxjournal.com/content/opening-minds-spheres-among-us) not a price we pay for making nothing but hierarchies, over and over again, because that is what our tools and mental models are biased to do?

That we come in many colors, sizes and body shapes—all with different faces that also change as we grow and age—is a grace meant to help us recognize every person as distinctive and separate. Not just so we can typify each other by any one of those characteristics. None of us is just black or white, male or female, tall or short. We are sovereign selves with complete souls that cannot be reduced to any one characteristic, no matter how easy it is to do that, especially with research and computers.

I bring this up because I believe it is also worth considering that the best case for distributed systems and networks is that they take advantage of the countless differences and originalities among us. Distributed systems, more than any other kind we can name—make possible recognizing that our greatest resources are each other—and ourselves.■

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

RETURN TO CONTENTS