Write Concurrent Apps in C with libmill

LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

WATCH: ISSUE OVERVIEW

AUGUST 2016 | ISSUE 268
http://www.linuxjournal.com

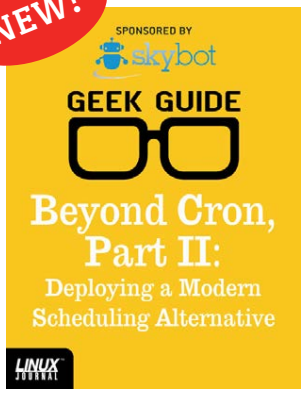
CREATE YOUR OWN TINY INTERNET

Make Qubes Even More Secure

+ Technology Travel Tips

A New Project for Linux at 25

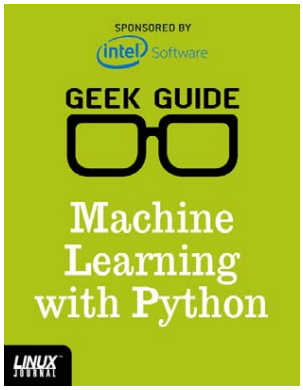
NEW!



Beyond Cron, Part II: Deploying a Modern Scheduling Alternative

Author: Mike Diehl

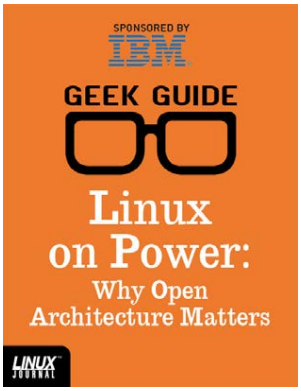
Sponsor: Skybot



Machine Learning with Python

Author: Reuven M. Lerner

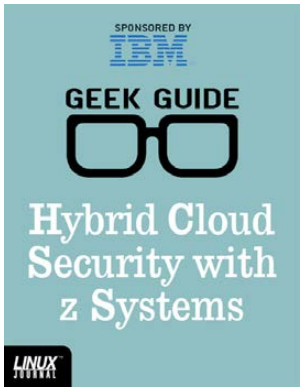
Sponsor: Intel



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

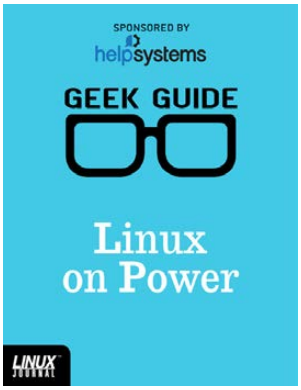
Author: John S. Tonello

Sponsor: IBM

Ceph: Open-Source SDS

Author: Ted Schmidt

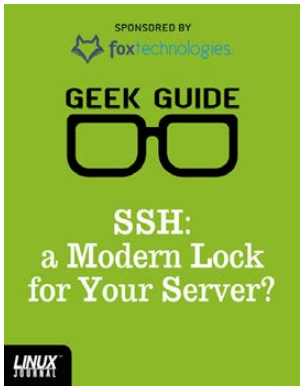
Sponsor: SUSE



Linux on Power

Author: Ted Schmidt

Sponsor: HelpSystems



SSH: a Modern Lock for Your Server?

Author: Federico Kereki

Sponsor: Fox Technologies

CONTENTS

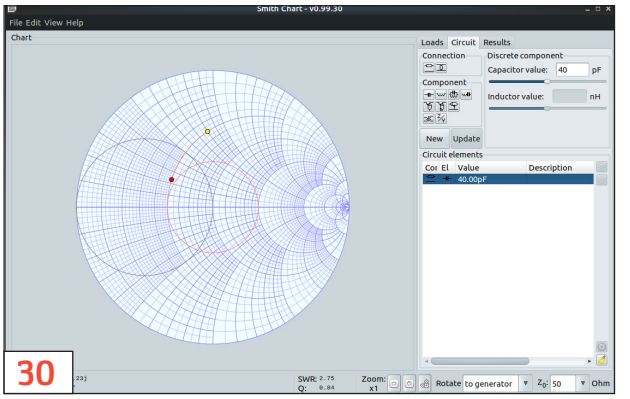
AUGUST 2016
ISSUE 268

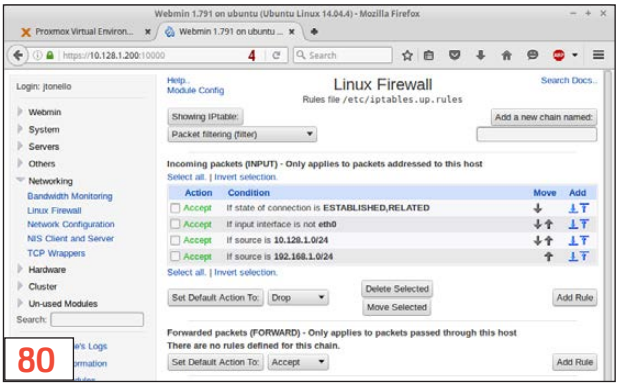
Cover Image: © Can Stock Photo Inc. / crstrbrt

FEATURES

COLUMNS

IN EVERY ISSUE





ON THE COVER

- Write Concurrent Apps in C with libmill, p. 112
- Create Your Own Tiny Internet, p. 80
- Make Qubes Even More Secure, p. 52
- Technology Travel Tips, p. 60
- A New Project for Linux at 25, p. 134

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

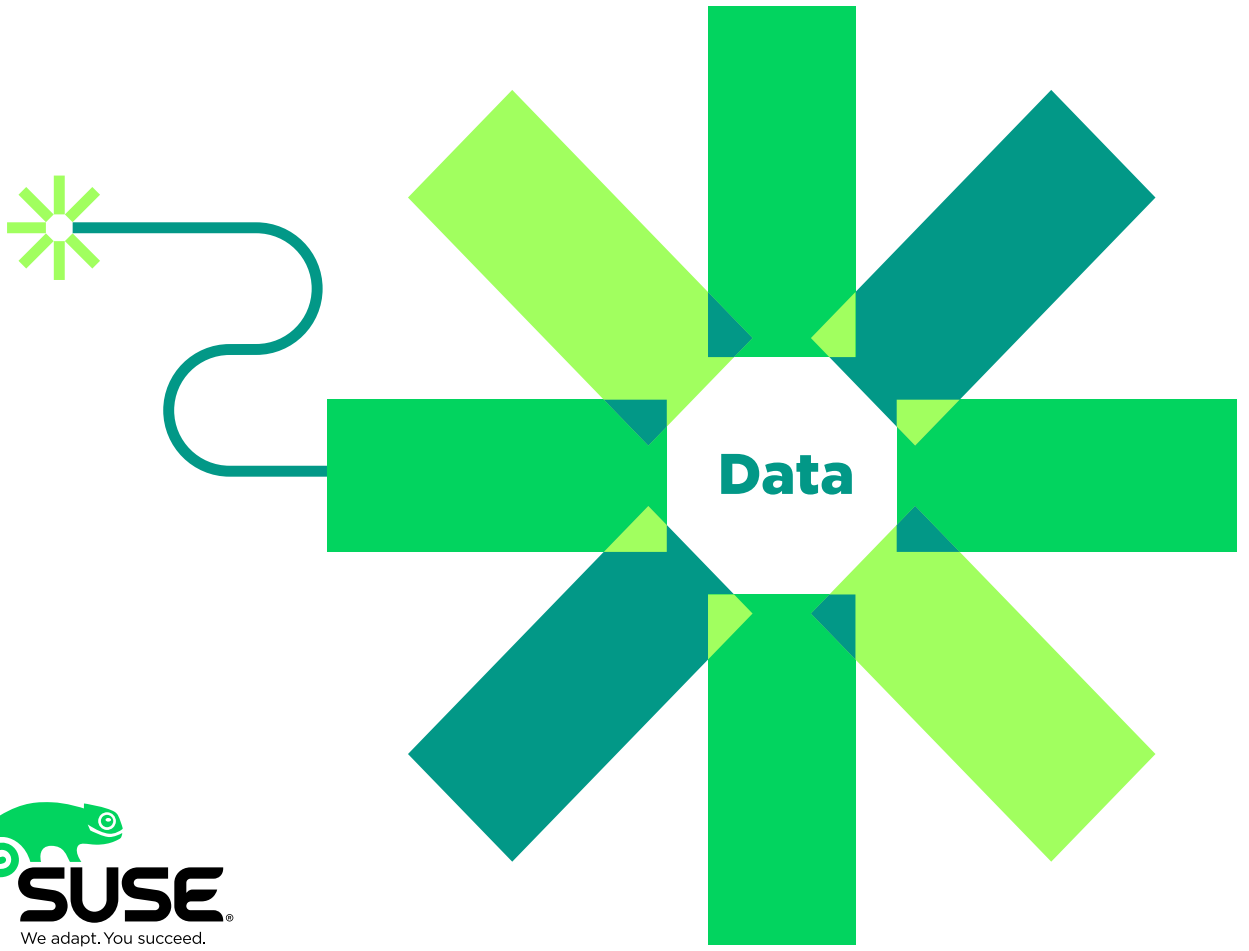
STORAGE
REDEFINED:

You cannot keep up with data explosion.

Manage data expansion with SUSE Enterprise Storage.

SUSE Enterprise Storage, the leading open source storage solution, is highly scalable and resilient, enabling high-end functionality at a fraction of the cost.

suse.com/storage



SUSE
We adapt. You succeed.

Linux: Just Do It!

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.

I recently had a conversation with a person in the tech world who does a lot of hiring. He started our conversation with a pretty open-ended request: "Shawn, talk to me about Linux and how it matters to people looking for a job." I assumed he was asking me why people should or shouldn't add Linux expertise to their résumés. Or, perhaps he was trying to get me to explain why a Linux professional is a value even in an environment largely containing Microsoft products. I was absolutely incorrect. The value of Linux already was assumed. He was asking how people should go about gaining training and experience! Let me say that again, *the value of Linux was already assumed*. It's like we're living in the future! On that note, let's learn about Linux.

Reuven M. Lerner starts off this issue with a how-to on migrating from Python 2 to Python 3. The transition was such that code doesn't automatically work in version 3 if it worked in version 2. That's not to say it's not worth moving to 3, just that it requires work. Reuven gives tips on working through that process incrementally.

Next, Dave Taylor finishes his series on "spinning" text, creating automated, personalized messages. It's a fascinating look at how computers are being used to do the sorts of things that used to require a human touch. Plus, it's just really awesome to see in action!

VIDEO: Shawn Powers runs through the latest issue.

Let me say that again, *the value of Linux was already assumed*. It's like we're living in the future!

Continuing his series on Qubes, Kyle Rankin is back this month explaining how to secure the security-focused distribution even further. Figuring out the advanced features of Qubes is just one more way to make your secure environment customized for your security needs.

My column this issue is more informational than educational. I often am asked about what I use on a daily basis, and I'm traveling this month, so I figured it was the perfect time to talk about how I work when traveling. I'm probably drastically different from anyone else, but perhaps one or two things I do might be useful for my fellow nerdy travelers.

John S. Tonello provides part 3 of his "Tiny Internet Project", where he teaches how to create your own diverse network infrastructure using virtualization and elbow grease. This month you'll learn how to get those installed systems going, so you can do whatever experimentation or learning you desire. If you've been following along with this project, you won't want to miss this conclusion.

Finally, Amit Saha delves deep into development with a look at coroutines and channels in the C language. If you are writing concurrent applications in C, you'll want to check out the libmill library, and Amit walks through its usage.

Just like every month, we have extensive looks at new products, reviews of existing technology, tips and tricks. Plus, we have a look at the implications of Linux and open source in our society with Doc Searls' EOF. Whether you are just learning Linux to make your résumé sparkle or are trying to learn even better ways to utilize Linux in your environment, this issue is bound to please. We hope you enjoy it as much as we've enjoyed putting it together!■

RETURN TO CONTENTS

PREVIOUS
Current_Issue.tar.gz

NEXT
UpFront

Doc Searls' June 2016 Column

I agree with most of Doc's comments. I started with IBM almost 50 years ago, and I have seen the computer infrastructure expand and contract in an endless cycle of centralized and distributed extremes.

As an early user of CompuServe, I can recall the frustration with both the expense and limitations of its idea of "email" and "networking".

Although I can see some positive attributes of the social-media engines, to me, they don't really have a product other than trapping me into their maze of advertising.

Even the internet is becoming more and more frustrating. I would much rather subscribe to my favorite sites than deal with the barrage of unwanted ads. Many commercial sites are becoming nearly unusable as they try to second-guess my intentions.

Keep up the good work, I have enjoyed your ideas for many years.

—John Crunk

Apache vs. nginx

Regarding Reuven M. Lerner's nginx article in the June 2016 issue: I use both nginx and Apache, and they both are very good servers. However, I would like to point out an inaccuracy in Reuven's article. He states that nginx's worker model is diametrically opposed to Apache's (i.e., async

event model vs. process per user). That's actually true for the old default installation of Apache, and it's also true if you install mod_php under Apache. However, Apache has several different concurrency systems you can choose from (they are called MPMs). The MPM Reuven is referring to is one of the old ones: mpm_prefork. There are other MPMs (like mpm_event) that actually work the way Reuven is describing that nginx works, and they have much better performance with high concurrent connection counts. The reason people use the old mpm_prefork (needlessly in my view) is because a lot of people run PHP under Apache, and they use mod_php to do it. mod_php isn't compatible with mpm_event, because mod_php isn't really believed to be thread-safe. So, what you do in this case is precisely what you do in the nginx case; you run PHP-FPM and use the FastCGI interface.

So, what really should be compared is performance of Apache running mpm_event talking to php-fpm vs. nginx talking to php-fpm. The big difference I see there is that Apache actually permits an inefficient solution, while nginx does not. Apache suffers from having a lot of old blog posts on the internet that teach old bad habits; nobody should be running an app server inside the web server process anymore.

—Daniel Waites

Reuven M. Lerner replies: *Your points about comparing apples to apples is a very good one. But—and I'm embarrassed to admit this—I somehow managed to miss the existence of the event-based MPM in Apache! This probably demonstrates how I've moved toward nginx for all but my most trivial projects.*

The very bright minds working on Apache haven't been resting on their laurels, and they have taken advantage of the MPM architecture to create an event-based system. I should have done some more investigation before basically accusing Apache of being old-school technology.

Comparing the event-based MPM with nginx would have been a fairer and more appropriate apples-to-apples comparison.

Screenshot Alternative

Regarding Shawn Powers' Non-Linux FOSS piece in the June 2016 issue: unfortunately, I too have grown accustomed to Snipping Tool, due to a proprietary OS at work, but home is a different story. Open a terminal within your GUI and enter `scrot -s`. The magic of this allows you to select an individual screen (if you have a dual-monitor setup) or draw a box around the item with the mouse to select the preferred item specifically.

—Eion Williamson

Shawn Powers replies: *Cool! Several versions of my home OS ago (Xubuntu), I had compiz set up so that I could hold down a fairly simple key combination and take a screenshot at will. I haven't ever gotten that working again since compiz isn't really used anymore. I always have at least one (or 15) terminal windows open though. Thanks!*

Appeal to Media: Help Stop This False Sense of Security from Spreading Any Further

Biometric authentications are good for physical security, but they ruin the security of password protection and generate a false sense of security in cyberspace. More specifically, deployed with a fallback password against a false rejection, they provide a level of security that is even poorer than a password-only authentication and yet trap people by giving the wrong impression that security is better than with the password-only authentication.

There is nothing wrong with a biometric product that is operated with a fallback password when that product is offered as a tool for increasing convenience. However, it would not only be foolish but also unethical and antisocial to make, sell and recommend such a product as a tool for increasing security, thereby spreading a false sense of improved security.

Take a few minutes to watch this short video, "Biometrics in Cyber Space—'below-one' factor authentication" (https://youtu.be/wuhB5vxKYlg), and you will certainly have no difficulty in realizing how the false sense of security has been generated. You might, however, reckon that this fact may well be very inconvenient to the media and reporters

who have, perhaps unknowingly, lent a hand to spreading this misconception and false sense of security.

This is not an issue of the relative comparison between "good" and "better", but the absolute judgment of "harmful" against "harmless". Something must be done before such critical sectors as medicine, defense and law enforcement are contaminated in a horrible way.

Furthermore, according to the article "Biometric Market Set to Skyrocket to $30Bn" (http://www.infosecurity-magazine.com/news/biometrics-market-set-to-skyrocket), the revenues of biometrics companies are expected "to reach more than $30 billion by 2021". Biometric solutions are used for physical security, including both forensic and cyber security. The budget for physical security might be well spent, but it is not the case for cyber security.

Assuming that the market for cyber security is no smaller than that for physical security, the figure of $30 billion tells us that no less than $15 billion would be wasted by 2021 for making negative contributions to cyber security, while making criminals and despotic regimes delighted. What a waste! What a folly! Becoming liberated from such a wasteful fate, the $15 billion could be better spent elsewhere for productive, constructive and ethical ends.

A dozen media outlets, including Elsevier, have started to help blow away this false sense of security generated by the misuse in cyberspace of biometric technology ("Misuse in Cyberspace of Biometrics Discussed on Media", http://www.slideshare.net/HitoshiKokumai/discussed-on-elseviers-btt-62502162)). Please consider joining them quickly as one of the front-runner media outlets and reporters.

—Hitoshi Kokumai

Shawn Powers replies: *Hitoshi, I think security must be one of the biggest areas we focus on as technologists in the present and near future. Thank you for the info, and our readers will be able to read your links as well. It seems our very concept of security, especially as it pertains to*

authentication, is broken. It's 2016, and I still see passwords on sticky notes attached to monitors. It's scary.

Secure Desktops with Qubes: Compartments

Thank you Kyle Rankin for another article about Qubes OS in the June 2016 issue. I've been using it since your first article in the April 2016 issue. The third article provided me a bit more information about organizing Qubes, and I'll try it.

One thing about Qubes that really impresses me is the available documentation to customize and do more things beyond what you get from the original installation. I am using an HP ProBook 6450 series with i5, 8GB RAM and 300GB HDD. It works fine except with Wi-Fi that has its button feature disabled from the BIOS to work properly. Recently I installed Win 7 with tools and Win 8.1 (no tools available yet) as standalone VMs. I am aware that there are more things to improve; however, for my daily use, it's been very stable and usable at my work, including some testing while visiting some public Wi-Fi networks. Great job Kyle!

—Antonio Misaka

The Tiny Internet Project, Part I

Regarding John S. Tonello's Tiny Internet Project beginning in the May 2016 issue: I am really excited about this project! The problem with Linux is that there are too many choices. That is a good problem to have, but it is a problem—especially if you don't know what you are doing and don't have time to try everything out. That is why I am excited about this project. When I am finished with the project, I expect to know enough to put together a small network. That is a valuable tool for almost anyone who enjoys Linux as a hobby, wants to teach it or just needs a working, secure network. Thanks John Tonello!

—Mike J. Nordyke

John S. Tonello replies: *I'm glad to hear you're diving into the Tiny Internet Project! For new users in particular, it can be frustrating to know just which flavor of Linux to adopt. If you're like most, once you commit*

to one, you tend to live with it (and favor it) for the rest of your Linux life.

Nowadays, there are essentially two main tracks that are most easily identified by their package types: .rpm or .deb. For the former, it's Red Hat, Fedora, CentOS, SUSE or Mandrake. For the latter, it's Debian, Ubuntu and derivatives like Linux Mint, Xubuntu or Lubuntu. The more significant differences are related to how the filesystems are organized, so if you start with Fedora, you may get a little lost on Ubuntu—and vice versa.

For the Tiny Internet Project, I use Ubuntu, which is popular for both servers and desktops. It's also very popular with developers. I personally use Linux Mint as my daily driver and manage a number of Linux servers running Ubuntu 14.04, so everything is seamless. But even Linux Mint comes in four different flavors: Cinnamon, MATE, KDE and Xfce. Each provides a different desktop experience, but the underlying systems are the same.

I'd recommend you make a couple bootable live USB drives, using a couple flavors of Linux Mint and a couple flavors of Fedora. You can test them without installing them, and once you explore a bit, you'll get a feel for the differences. Then you can more confidently decide on your "forever Linux".

Comments on the June 2016 Issue

Here are a few comments that the June issue elicited as I read.

initrd (see the Letters section):

Traditionally, an initrd was built using a loop-mounted file on which a filesystem was made and to which dirs and files were added. It took a few steps, but worked well.

The more modern method is to use an initramfs; no block device is needed, no loop-mounting, no filesystem. And, an initramfs is really trivial to create:

1) Create a directory (for example, /root/myinitramfs).

2) In that directory, place the standard directories and files Linux needs to run. Include as much or as little as you wish.

3) Change to that directory and execute `find . | cpio -o -H newc | gzip > /root/myinitramfs.gz`).

Pretty much every installed Linux kernel today uses an initramfs. It really is that easy to create an initramfs.

ANSI 3.64 (see Dave Taylor's Work the Shell "Publishing the wegrep Wrapper Script"):

This is just a pedantic nit. ANSI 3.64 and ECMA-48 were merged and became ISO-6429; ANSI 3.64 was withdrawn in the mid-1990s. But ANSI 3.64 will live decades longer, just as the async serial spec hasn't been a Recommended Standard since 1984, and has been twice revised—it is now TIA-232-E.

Doc Searls' EOF "What's Our Next Fight?":

We can start the fight by accelerating the change to IPv6 because that protocol's built-in multicast capabilities will further enhance peoples' ability to form their own common-interest groups.

We can continue the fight by producing simple, free software that makes it trivial to make peer-to-peer cross-platform voice and video connections over the internet where "peer-to-peer" explicitly means that no third-party "service" is needed or involved, and cross-platform means the program is readily available for MacOS, iOS, Android, Windows, Linux and other systems. People don't mind paying to use the information highways and streets, but they sure do get annoyed when they have to pay for that use *and* pay tolls every time they turn onto a different street.

—Neal Murphy

Note for Shawn Powers' "Build Your Own RPi Camera" in the June 2016 Issue

You've probably heard this from many others already, but just in case you haven't, you can disable the red LED on the camera by adding the line:

```
disable_camera_led=1
```

to the file /boot/config.txt. At least that works for me.

—Roger

Shawn Powers replies: *Ha! Actually, Roger, no one has sent that valuable information my way. Thank you!*

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

LINUX JOURNAL

At Your Service

SUBSCRIPTIONS: *Linux Journal* is available in a variety of digital formats, including PDF, .epub, .mobi and an online digital edition, as well as apps for iOS and Android devices. Renewing your subscription, changing your email address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly online: **http://www.linuxjournal.com/subs**. Email us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, PO Box 980985, Houston, TX 77098 USA. Please remember to include your complete name and address when contacting us.

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

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at **http://www.linuxjournal.com/contact** or mail them to *Linux Journal, PO Box 980985,* Houston, TX 77098 USA. Letters may be edited for space and clarity.

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

FREE e-NEWSLETTERS: *Linux Journal* editors publish newsletters on both a weekly and monthly basis. Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on **http://www.linuxjournal.com**. Subscribe for free today: **http://www.linuxjournal.com/enewsletters**.

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

PREVIOUS
Letters

NEXT
Editors' Choice

diff -u

What's New in Kernel Development

One of the difficulties with "live patching" a running kernel is the desire to remove the older version of a patched module, once the patch has been applied. Even if all new invocations of patched functions are sent to the new code, how can you be sure that some piece of code in that older version isn't still running because it was called before the live patch occurred?

Miroslav Benes recently posted some patches to resolve this issue. His code essentially implemented a reference counter, which would mark existing users of a given module as "safe" if they were running on the new patched code. Any code running in that module that was not yet marked "safe" would be running on the old version. Once all users were marked "safe", the older version of that module would have no more users and could be removed from the running kernel safely.

Josh Poimboeuf had also worked on this code and offered some technical criticisms of Miroslav's patch. In particular, he pointed out that users could mess with the reference count for a given module, simply by disabling and re-enabling it in rapid succession within the kernel. Miroslav replied, "That is unfortunately true. I

don't have a solution in my pocket that would be 100% reliable. At the same time I don't see a practical problem. Yes, refcount could overflow, but that shouldn't be a problem, should it? Anyway, I'll note it in the changelog."

Apparently it's one of those bugs that occurs only under pathological circumstances, and so it doesn't really need a solid fix.

Other issues required further debate, and **Jiri Kosina**, who had the responsibility to accept the patch once it was ready and feed it up to **Linus Torvalds**, joined the discussion. There were various sequences of user actions that potentially could crash the system, and the code had to be able to deal with each of them before it could be accepted into the kernel. Also, the code had to avoid too many time-consuming checks along the way; it had to just work. So, for example, as Jiri put it at one point:

> My understanding is that the concern here is that walking through the complete linked list every time sysfs node is accessed, just to figure out whether we're able to find a klp_patch entry that points back to the particular kobject that's being passed to the sysfs callback, isn't really super-efficient. I personally wouldn't worry *that* much about that particular aspect (sysfs operations are hardly considered time critical anyway), but I'd have to think a bit more whether this is really safe wrt Deadlocks between kernfs locks and klp_mutex; but so far it seems to me that klp_mutex always nests below kernfs, so it should be OK.

In a related post, **Jessica Yu** pointed out

> If any of the sysfs functions get called, we would have to take care to ensure that the klp_patch struct corresponding to the kobject in question actually still exists. In this case, all sysfs functions would require an extra check to make sure the matching klp_patch is still on the patches list and return an error if it isn't found.

She said this would be simple to implement, but complex to conceptualize, and she suggested other alternatives as well.

Ultimately, there don't seem to be any major roadblocks standing

in the way of this code, but there do seem to be a number of finicky nuances that everyone involved will need to consider very carefully. But, the code does seem on track to getting into the kernel.

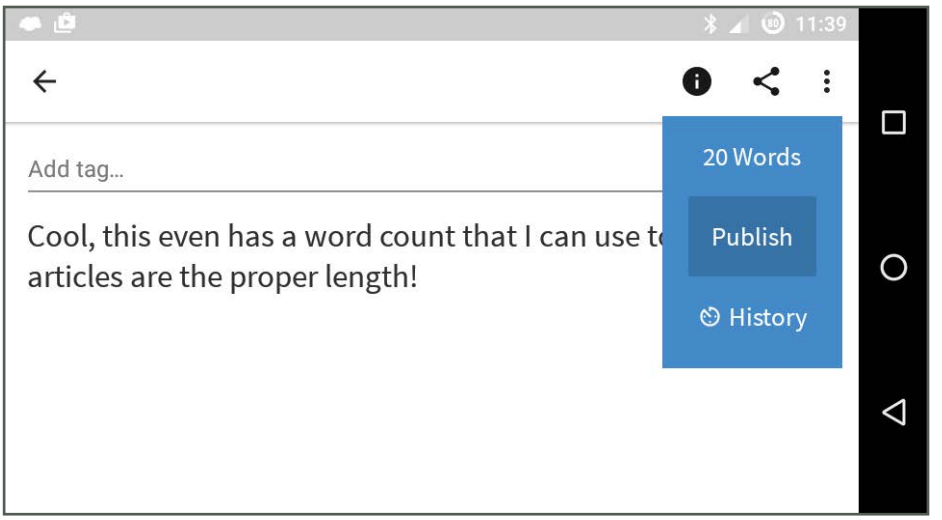
Sometimes legacy features are brought back to life when hardware manufacturers begin to depend on them once again—in this case, **ISA** (Industry Standard Architecture), the 16-bit bus used on **IBM** systems in the 1980s. Recently, **William Gray** posted some patches to deal with modern hardware, such as PC/104 cards, which are modular, embeddable systems that run on the ISA bus. William's code enabled ISA support on a per-driver basis. As William put it:

> This patch introduces the ISA_BUS_API and ISA_BUS Kconfig options. The ISA bus driver will now build conditionally on the ISA_BUS_API Kconfig option, which defaults to the legacy ISA Kconfig option. The ISA_BUS Kconfig option allows the ISA_BUS_API Kconfig option to be selected on architectures which do not enable ISA (e.g. X86_64).
>
> The ISA_BUS Kconfig option is currently only implemented for X86 architectures. Other architectures may have their own ISA_BUS Kconfig options added as required.

Linus Torvalds responded to William's post with a criticism of the way the dependencies were structured. He felt that the ISA symbol was defined for non-x86 architectures, and so the behavior of William's code on those architectures should be the same as on x86 systems. That is, Linus said that ISA_BUS_API should exist on all other architectures that depend on ISA, if that's the behavior we'd expect under x86.

After a few more versions of his patch, William posted one that was intended to be accepted into the kernel. This time, Linus replied, "This version seems fine and safe. I didn't see the other patches in the series (not cc'd to me), but at least this one would seem to do the right thing and expose part of the ISA code without causing other architectures to possibly lose it."—Zack Brown

Android Candy: Teach an Android to Take Notes



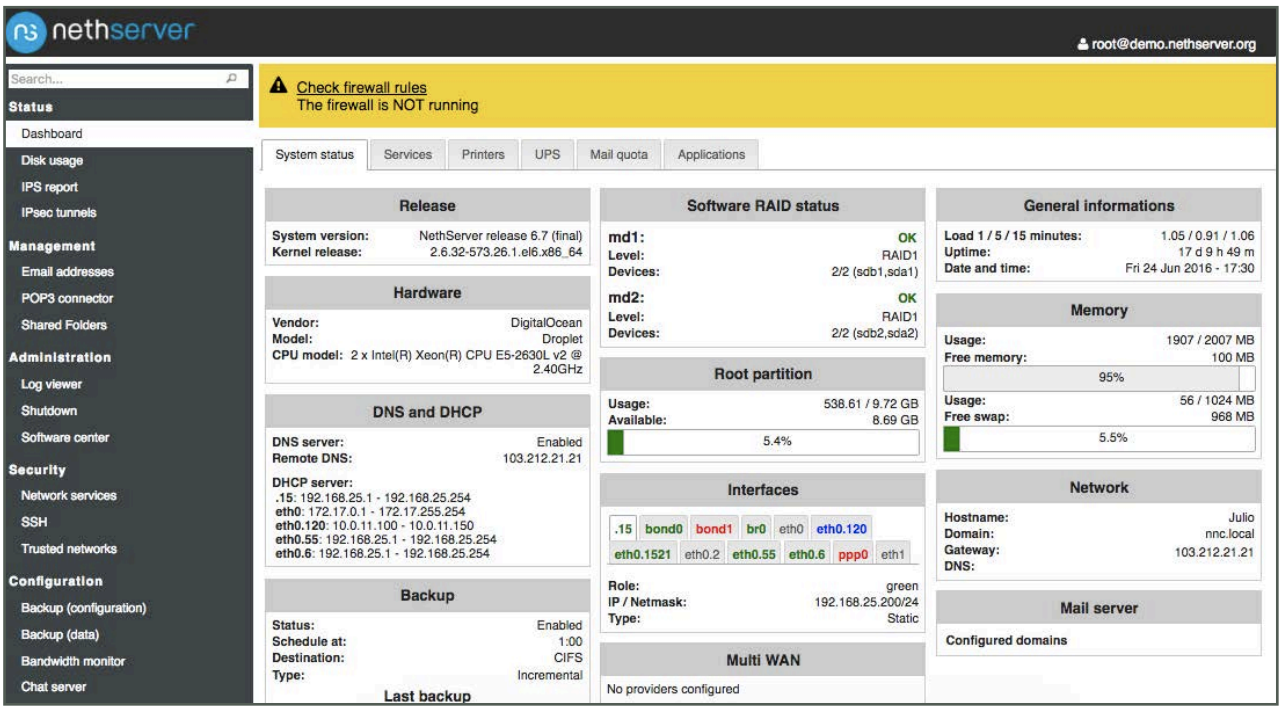
In my "Non-Linux FOSS" piece in this issue, I talk about the newly open-sourced Simplenote application from the folks at Automattic. One of the other fairly new additions to their cross-platform note-syncing platform is that an Android app is available for free in the Google Play Store.

The Android app does pretty much all the same things the desktop or web-based app does, and it syncs between devices automatically. With many "syncing" apps, I find it frustrating that the syncing isn't reliable or consistent. I have to admit, this one seems pretty solid. Perhaps it's because it does text and only text—I'm not sure.

To be completely honest, Simplenote doesn't do nearly as many things as Evernote does; however, that seems to be by design. It is a "simple note" program, and it handles simple notes extremely well. Along with syncing to other devices, it also allows you to publish notes publicly on the web so others can see them (for example, http://simp.ly/publish/L75FXz). On top of that, it allows you to share notes between a group of users simply by tagging the note with their email addresses.

Simplenote may not be as robust as Evernote, but what it does with text is amazing. From my limited experience with it, it's also fairly reliable. If you're not using Simplenote for taking simple notes, you should give it a try. Just search for Simplenote in the Google Play Store. It's from the folks at Automattic, the same company responsible for WordPress.—Shawn Powers

NethServer: Linux without All That Linux Stuff



Okay, that title really isn't fair. NethServer has all the Linux stuff, it's just that you don't have to interact with it in the traditional way in order to reap the benefits. NethServer is a web-based management software package built on top of CentOS. You can download it as a separate distribution, but truly, it's just software on top of CentOS. In fact, the installation methods are either "install the NethServer distro" or "add the NethServer repository to your existing CentOS install". I really like that.

The concept behind NethServer isn't a new one. Lots of distributions are designed to simplify managing a server. I've written

about ClearOS, Untangle and several others in the past. Plus, you always can just install Webmin on your server and get a "roll your own" web-administered system. The thing I like about NethServer is how well it allows you to configure services while not doing anything proprietary underneath. I think the interface is simple and intuitive as well.

Tons of features are available in the free community version of NethServer, but a few of the more advanced features require you to purchase a license. If you'd like to give NethServer a try, you either can download the ISO, install the repository on your existing server, or try the live Docker-based demo! Check it out at http://www.nethserver.org/demo-running-on-docker.

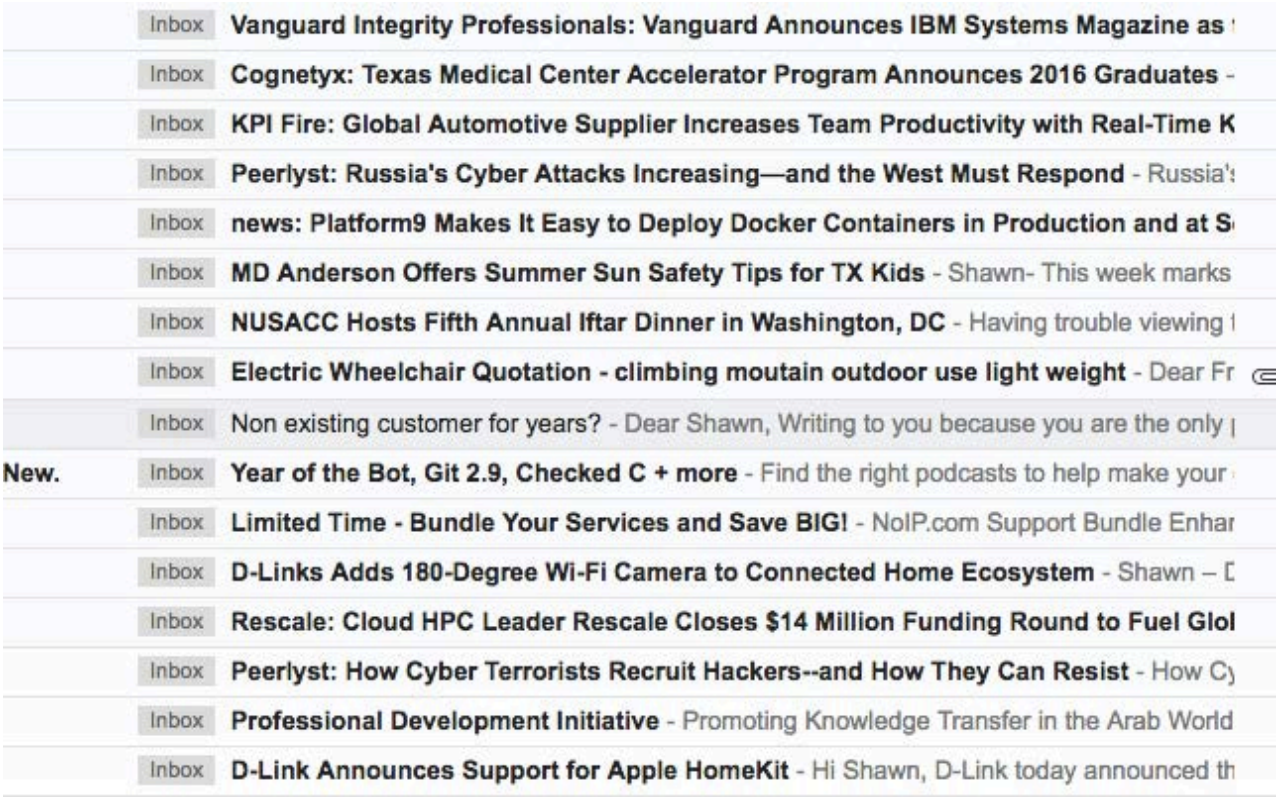
—Shawn Powers

How to Make Me Dance

I feel a little weird sharing to a large audience the best way to get my attention when trying to pitch an idea or project. Still, the amount of ineffective email I get on any given day is overwhelming, so maybe it will be for the best. (See the screenshot, that's from yesterday afternoon.)

Last month, the community manager for Nethserver, Alessio Fattorini, contacted me via email. The message appeared to be written in earnest, so it got much farther than most email I get. Generally, I never make it past the subject line. Anyway, his message was a plea for help in learning how best to reach the media and journalists he wanted to write about his project. No one ever had asked me that before, so I wrote him back. Here's my message:

> Alessio,
>
> I've never been on your side of the equation, so I'm not sure how companies get contact info. I can tell you my process for deciding what to look at (the first step in deciding what to write about). I can also tell you what turns me off. This is just off the top of my head, and honestly, since this isn't my main job, I probably don't function like the folks who depend on such information on a daily basis. For me, I have 4–6 items a month that I highlight, so I tend to be picky.
>
> 1) I hate standard PR email messages. They tend to sound like they were written by a marketing department (they often are), and I personally don't care about how successful the CEO has been and so on. I want to know why a product announcement would be interesting to my readers, and I want to know early in the email. If your subject is "Blah blah announces new version...", I generally don't even read it.
>
> 2) Keeping in mind the last sentence of the previous point, if you "trick" me into reading an email with a clever, deceptive title, I will



not write about your product. I probably will dislike your product on principle and think you're a horrible person. (DirecTV has this new campaign where it sends out envelopes that look like greeting cards and appear to have handwritten addresses. When you open it, it's a cheesy "card" where the CEO or whomever is excited to tell you about the great prices DirecTV offers. I absolutely hate DirecTV now and never, ever will subscribe to its service!)

3) I like to know about new features that make your product unique, or if I don't know your product (assume I don't, it's a big world—for example, I have no idea what NethServer is), I want to know what makes it worth learning about. There's really no magic formula for this, because if there were, it already would be overused and wouldn't work anymore. But don't start your subject with "RE:", as if we've been having a conversation about it, because that's starting to be a little too much like DirecTV.

4) This may be the most important: remember what your audience (press, in this case) is looking for. We want to entertain and inform our readers; we don't want to sell your product. I think of my readers as my friends (seriously, I'd totally buy you a cup of coffee if you were in the area), and I want to tell my friends about things that are interesting, useful and fun to talk about. I would never say to my friends, "Hey, there's a highly successful CEO that has pioneered a new product that will revolutionize the way we create virtual machines!" If I did, that friend should punch me in the face. I'd rather say something like, "Dude! Have you ever heard of NethServer? It's like Webmin, but it doesn't suck!" (I'm basing that on your email information, I have no idea if that's really what your product is like.)

The truth is, if you're excited and passionate about your product, don't hide it. Make sure I know you're excited about it and tell me why. At the end of the day, I'm just a guy with a lot of nerdy friends, and I love to tell them about cool stuff. If you have cool stuff, tell me about the stuff, not the company or the CEO or last quarter's earnings.

The response describes honestly how I go about picking subjects to share. If you have a project you'd like to share with the *Linux Journal* community, drop us an email at ljeditor@linuxjournal.com and tell us about it. But if I see "Dear $NAME" in the subject line, I might send sign you up for a Christmas card from DirecTV.—**Shawn Powers**

Smith Charts for All

I've covered several different programs that are useful when doing electrical engineering in the past. In this article, I want to look at a program called linsmith (http://jcoppens.com/soft/linsmith/index.en.php) that helps you do calculations or see how different parameters behave.

Linsmith allows you to generate Smith charts for problems in electrical engineering, especially RF (radio frequency) circuits. Smith charts are a graphical way of representing the rather complex interactions that can happen when dealing with multiple nonlinear electrical components. You can use them to see how they interact and what happens if you vary some of the parameters.

Now, let's look at how to use linsmith to try to make this task a little easier. Throughout this article, I am assuming that you know enough about electrical circuits that I won't need to explain too many of the terms I'm using. If you want to learn more, a good place to start is the Wikipedia

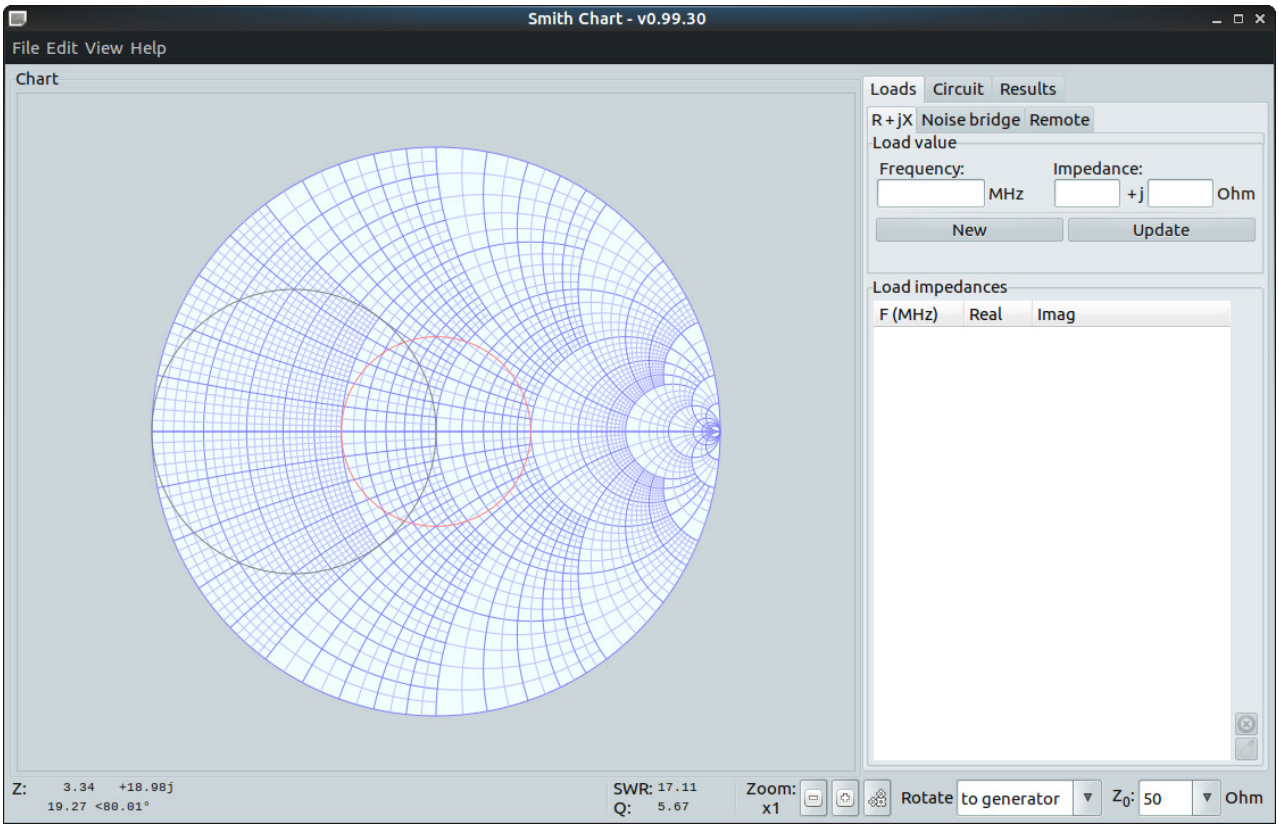


Figure 1. When you first start linsmith, you will see a blank Smith chart, ready for you to use.

page for Smith charts: https://en.wikipedia.org/wiki/Smith_chart.

First, you will want to install it on your system. It should be in the package management system for your preferred distribution. For example, you can install it on Debian-based distributions with the command:

```
sudo apt-get install linsmith
```

Once it's installed, you can start it either by finding it within the menu system for your desktop environment or by running the `linsmith` command within a terminal window.

This program is strictly a graphical one, so you need to be running X11 in order to use it.

When it first starts, you will see a blank Smith chart, ready for use. On the right-hand side of the main window is a set of tabbed panels where you can enter the details of the electrical problem you are working on. This section is broken into loads that you can apply to the system, a

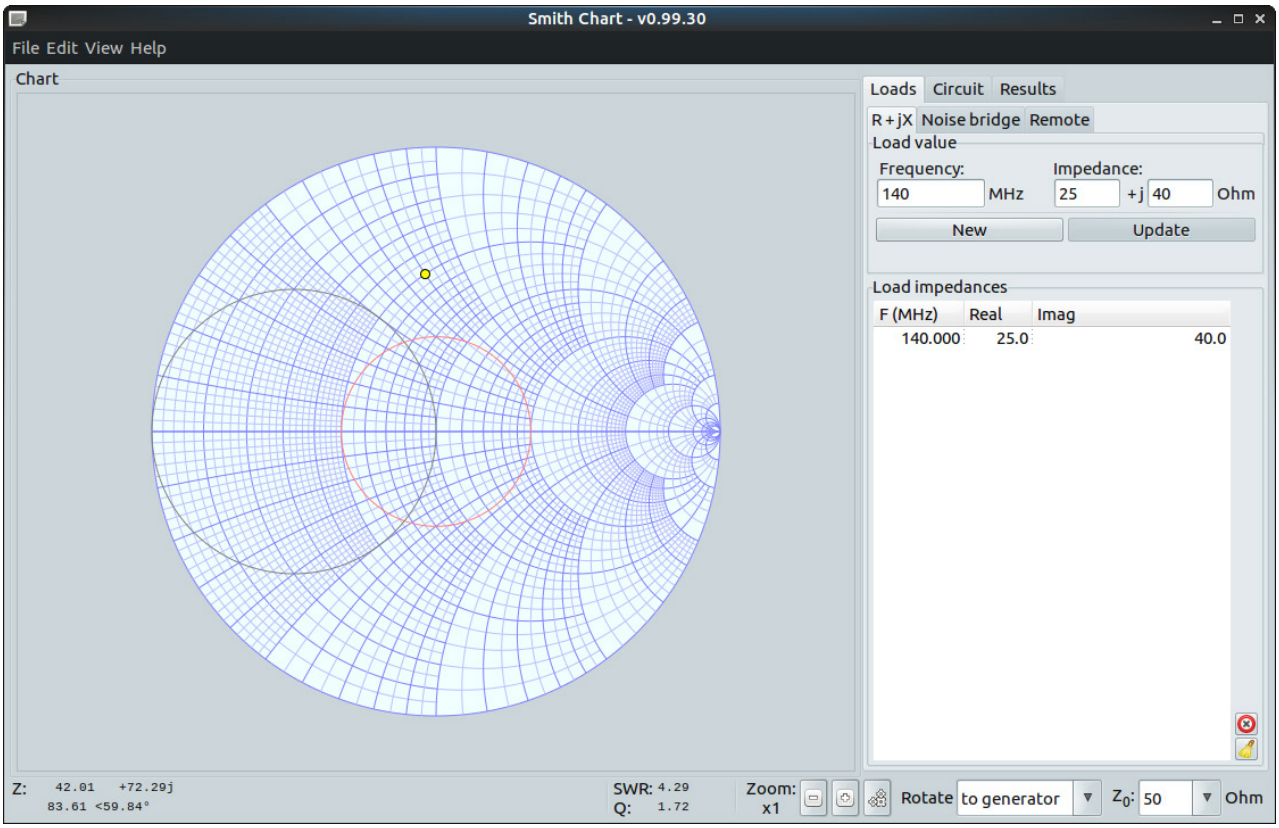


Figure 2. Adding a new load adds a yellow point to the Smith chart.

circuit tab where you can define discrete circuit elements that are part of the problem, and a results tab where you can find a running log of the calculations being made.

Now, let's look at what you can do in terms of applying loads to the system using the Loads tab. This section is actually broken down further into three more sub-tabs. The first one is labeled "R + jX". This tab allows you to enter a load characterized by a frequency, given in mega-Hertz, and an impedance, given as a complex-valued number of Ohms. For example, you could enter a load of 140MHz, with an impedance of (25+j40) Ohms, by entering these values in the appropriate boxes and then clicking the button labeled New directly below them. This will place a new load value in the table of load impedances, and it will display a new yellow dot representing this load on the Smith chart.

Now that you have a load, you can alter it by adding in extra components. You can do this by clicking on the tab labeled Circuit. In the Component section, you can select from elements, such

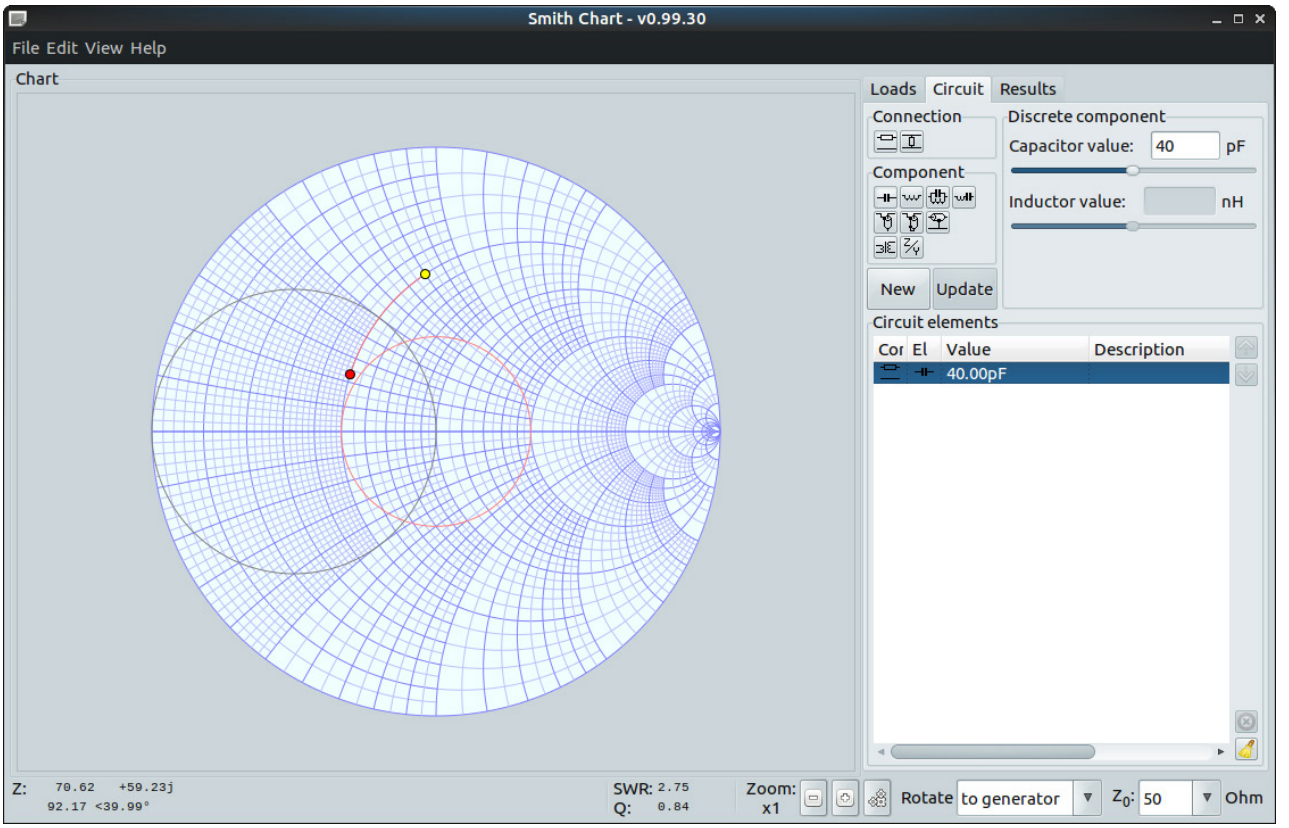


Figure 3. You can add components, such as capacitors.

as capacitors, inductors or even transformers. For each of those components, there is a different set of parameters that you can use to define your new component.

As an example, you could add a capacitor of 40pF by clicking on the capacitor button and entering the value in the "Capacitor value:" text box. If you don't recognize the different component symbols, you can simply hover over them to see a text box appear.

Once you have the values all set, you can click on the New button just below the component section to add this element to your Smith chart. This will add a red line and a red end point on the Smith chart.

In all cases, you can select an element that already exists to edit its values. When you select an item, you can edit its parameters in the appropriate boxes in the top of the pane. Once you're done, you need to click on the Update button to apply the changes to your Smith chart. These elements can be added either in series or in parallel. This option is in the section of the right-hand side panel labeled Connection.

Clicking on the Edit→Preferences menu item will open a new window where you can set several preferences for how linsmith can work. Several tabs covering several sections of options are available. The first tab, General, sets the most basic of parameters. The screen tab lets you set the image used as the background of the Smith chart, along with what colors you want to use for the various elements. The

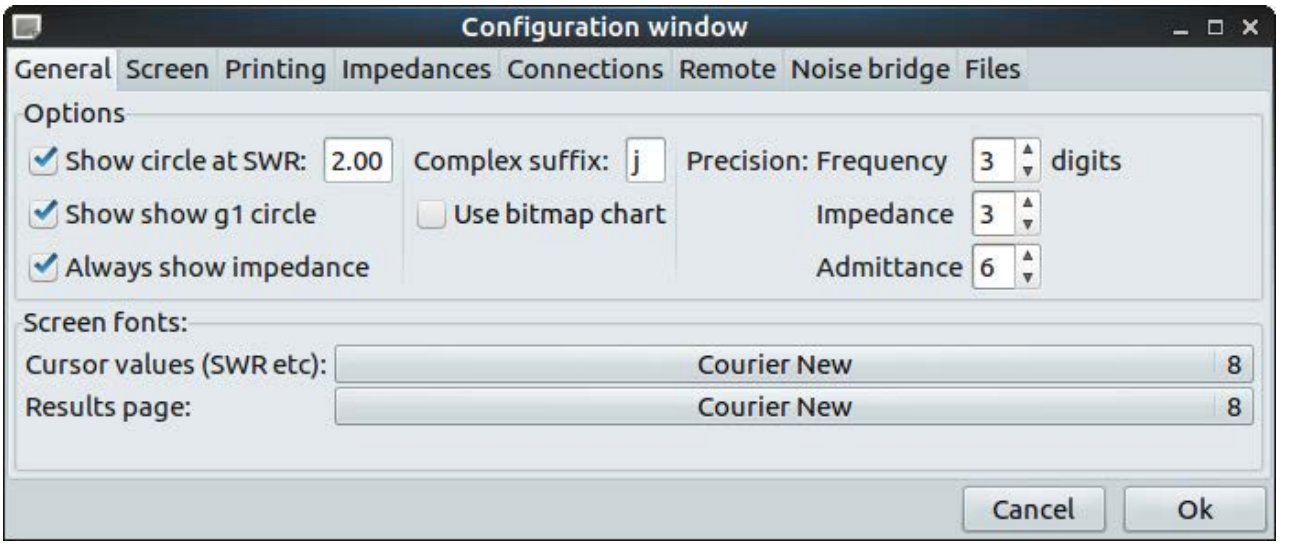


Figure 4. You can set a number of parameters using the preferences window.

printing tab lets you set the paper size and margins, along with the colors to use for each of the parts of your Smith chart.

In the main display, you should see a number of informational displays for your problem on the bottom bar. This is a Z Smith chart, so the values at the far left-hand side of the bottom bar are coordinates within the chart. On the right-hand side, there is a display of the SWR (Standing Wave Ratio) as well as the Q value for the problem. After this, there is a set of zoom buttons where you can zoom in on sections of the chart to see further details. There also is a button to recalculate the results of the Smith chart based on changes you may have made.

The last two options allow you to change the base values of the Smith chart. You can rotate the chart either to the load or to the generator. You also can change the normalization impedance from the default 50 Ohms to one of 75, 150, 300 or 600.

Once you're done, you can save your work in a few different ways. Under the File menu item, you can select to save either circuit details or load details as a separate file. Each of these sub-menus allows you to save data or load previous data. You also can import data from CSV files and s2p files. To save the final results, you can print the resultant Smith chart either by pressing Ctrl-P or by selecting the File→Print menu item. This way, you can save the chart to a PDF file. Hopefully, linsmith will be a useful tool for electrical engineers to add to their toolboxes.—Joey Bernard

THEY SAID IT

Nothing ever goes away.
—Barry Commoner

Never tell people how to do things. Tell them what to do and they will surprise you with their ingenuity.
—George S. Patton

Nothing is particularly hard if you divide it into small jobs.
—Henry Ford

Another belief of mine: that everyone else my age is an adult, whereas I am merely in disguise.
—Margaret Atwood

Act as if it were impossible to fail.
—Dorothea Brande

PREVIOUS
UpFront

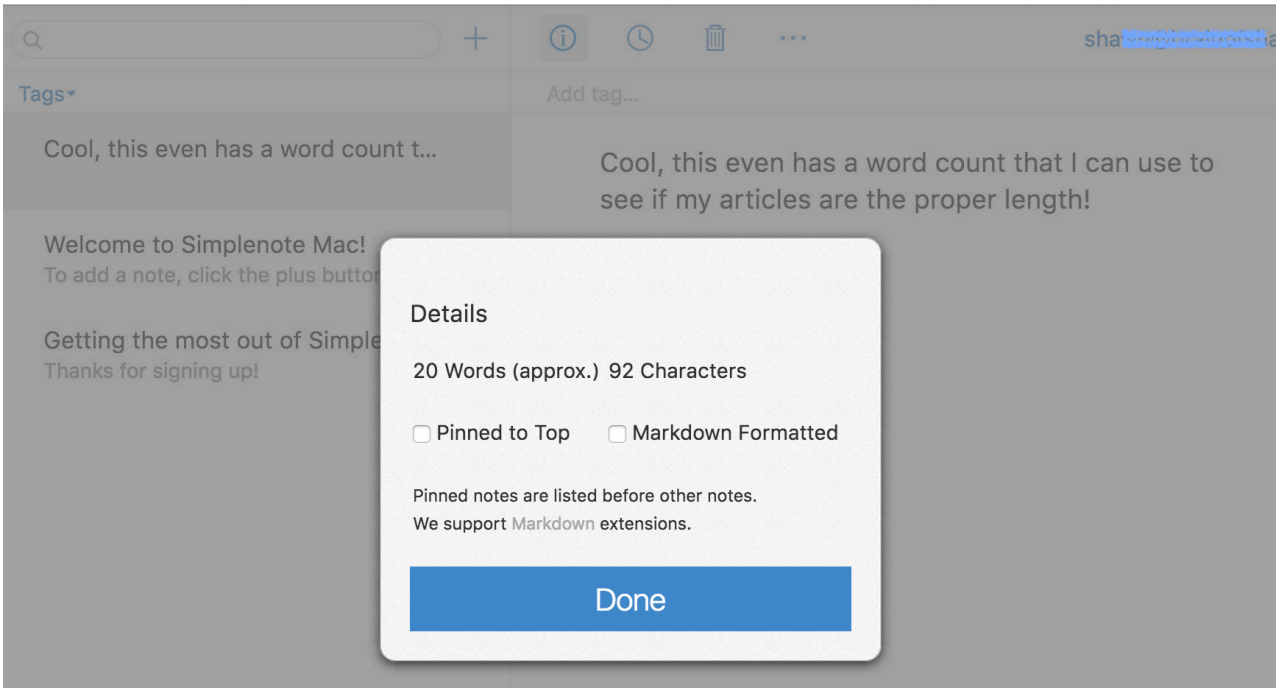
NEXT
Reuven M. Lerner's
At the Forge

Non-Linux FOSS Simplenote, Simply Awesome!

I'm a big Evernote user. It's a powerful commercial program that allows you to sync text, photos and documents across multiple devices. Sadly, there's no native Linux client. Also, it's a proprietary software package, and that bums me out.

Simplenote has been an alternative to Evernote for quite some time now. It's created by Automattic, the folks behind WordPress. It's designed to sync only text-based notes, but that's usually all I want anyway. Recently, the developers at Automattic decided to release Simplenote as open source! They also compiled binaries for just about every platform out there, including Linux!

The data is synced into their cloud, so if you're not keen on someone else keeping a copy of your data, Simplenote might not be for you. The convenience of multi-platform synchronization is worth it for me. Since the software itself is open source, a person could look and see exactly what they do with your data as well.



Thanks mainly to its cross-platform capabilities, including multiple apps for mobile devices, I'm giving the new open-source Simplenote this month's Editors' Choice award. Grab a copy today from http://simplenote.com, or head over to its GitHub page and snag the source code: https://github.com/Automattic/simplenote-electron.

(For a look at the Android-specific version of Simplenote, check out my Android Candy piece also in the UpFront section of this issue.)—Shawn Powers

RETURN TO CONTENTS

Transitioning to Python 3

Still using Python 2? Unable to upgrade? Try an incremental approach.

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 PYTHON LANGUAGE, which is not new but continues to gain momentum and users as if it were, has changed remarkably little since it first was released. I don't mean to say that Python hasn't changed; it has grown, gaining functionality and speed, and it's now a hot language in a variety of domains, from data science to test automation to education. But, those who last used Python 15 or 20 years ago would feel that the latest versions of the language are a natural extension and evolution of what they already know.

At the same time, changes to the language—and particularly changes made in Python 3.x—mean that Python 2 programs won't run unmodified in Python 3. This is a known issue, and it was part of the process that Python's BDFL (Benevolent Dictator

for Life) Guido van Rossum announced back when the "Python 3000" project was launched years ago. Guido expected it would take time for organizations to move from Python 2 to Python 3, but he also felt that the improvements to the language were necessary.

The good news is that Python 3, which at the time of this writing exists in version 3.5, is indeed better than Python 2. The bad news is that there still are a lot of companies (including many of my training and consulting clients) that still use Python 2.

Why don't they just upgrade? For the most part, it's because the time and effort needed to do so aren't seen as a worthwhile investment of developer resources. Most differences between Python 2 and 3 are easily expressed and understood by people, but the upgrades aren't completely automatic. Moving a large code base from Python 2 to 3 might take days, but it also might take weeks or months.

That said, companies will soon be forced to upgrade, because as of the year 2020, there will be no more support for Python 2. That's a risk many companies aren't going to want to take.

If you have to upgrade, but can't upgrade, that puts you in a terrible spot. However, there is another option: upgrade incrementally, modifying just 1–2 files each week so that they work with both Python 2 and 3. After a number of months of such incremental changes, you'll be able to switch completely to Python 3 with relatively little investment.

How can you make your code compatible with both? In this article, I provide a number of suggestions on how to do this, using both an understanding of Python 3's changes and the tools that have been developed to make this transition easier. Don't wait until 2019 to start making these changes; if you're a Python developer, you already (in mid-2016) should be thinking about how to change your code to be Python 3-compatible.

What Has Changed?

The first thing to ask is this: what exactly changed in Python 3? And, how easily can you move from Python 2 to Python 3? Or, how can you modify your Python 2 programs so they'll continue to work in Python 2, but then also work unmodified in Python 3? This last question is probably the most important one for my clients, and possibly for your business as well, during this transition period.

On the face of things, not very much actually changed in Python 3. It's a cleaner, more efficient and modern language that works like more modern Python developers want and expect. Things that Python developers were doing for years, but that weren't defaults in the language, are now indeed defaults. Sure, there are things I'm still getting used to after years of bad habits, such as failing to use parentheses around the arguments passed to `print`, but on the whole, the language has stayed the same.

However, this doesn't mean that nothing has changed or that you can get away with not changing your code.

For example, you almost certainly never wanted to use Python 2's `input` built-in function to get user input. Rather, you wanted to use the `raw_input` built-in function. So in Python 3, there is no equivalent to Python 2's `input`; the Python 3 `input` function is the same as Python 2's `raw_input`.

A more profound change is the switch in the behavior of strings. No longer do strings contain bytes; now they contain Unicode characters, encoded using UTF-8. If 100% of your work uses ASCII, you're in luck; nothing in your programs will really need to change. But if you use non-ASCII characters, and if you do so in the same program as you work with the contents of binary files, you'll have to make some adjustments. Python 2's `str` class is now a `bytes` class, and Python 2's `unicode` class is now the `str` class.

A number of other changes have been made that make Python more efficient. For example, Python 2 has the `range` function (which returns a list of integers) and the `xrange` function (which returns an iterator). Python 3's `range` function is the same as Python 2's `xrange`, because it's so much more efficient, and there really are few reasons to prefer the old `range`. But if your program expects to get a list back from `range`, you might be in trouble when you move to Python 3.

Another problem, which has become far less acute in the last year or two, is that of third-party libraries. If you're using packages from PyPI, you need to make sure not only that your own code works with Python 3, but also that all of those packages do. For a long time, I would argue that these packages were the bottleneck stopping many people from upgrading. But nowadays, most popular packages support Python 3, as you can see at http://py3readiness.org, which tracks such information.

Identifying Problems

So, how can you take a Python 2 program and modify it so that it'll work under both Python 2 and 3? You could go through the code line by line and try to find changes, but there are tools that can make the process much easier.

The first is an old friend of Python developers, the `pylint` program, which normally checks your code for Python style and usage. Modern versions of `pylint` have a `py3k` option you can apply that checks your code to see how compatible it is with Python 3. For example, let's assume you have written the (terrible) program shown in Listing 1. How can you find out which parts of it aren't going to work? You can run this:

```
pylint --py3k oldstuff.py
```

And, you'll get the following output:

```
************* Module oldstuff
W:  3, 7: raw_input built-in referenced (raw_input-builtin)
E:  4, 0: print statement used (print-statement)
E:  5, 0: print statement used (print-statement)
E:  6, 0: print statement used (print-statement)
W:  8, 9: raw_input built-in referenced (raw_input-builtin)
E: 10, 4: print statement used (print-statement)
W: 10,48: division w/o __future__ statement (old-division)
E: 14, 4: print statement used (print-statement)
W: 16, 4: range built-in referenced when not iterating
 ➥(range-builtin-not-iterating)
E: 17, 0: print statement used (print-statement)
```

The output contains both errors ("E") and warnings ("W"). The example program is using `print` as a statement, rather than a function. It's using `range` when not iterating. And, it's using `raw_input`. What can you do about it, and how can you improve things? `pylint` won't tell you; that's not its job. But if nothing else, you now have a list of things to fix and improve, so that it'll at least run under Python 3.

Listing 1. oldstuff.py

```
#!/usr/bin/env python

name = raw_input("Enter your name: ")
print "Hello, ",
print name,
print "!"

number = raw_input("Enter a number: ")
for i in [2,3,5]:
    print "{} / {} = {}".format(int(number), i, int(number) / i)


for i in range(10):
    print i

x = range(10)
print x[3]
```

If you have written a Python package with a requirements file, you can download and install `caniusepython3` from PyPI. Running `caniusepython3` against your requirements file will indicate what will work and what won't. If you don't want to download and install `caniusepython3`, you actually can go to http://caniusepython3.com and upload your requirements file there.

Fixing Problems

Python has come with a program called `2to3` for some time that looks over your Python 2 code and tries to find ways to make it work with Python 3. So, you can run:

```
2to3 oldstuff.py
```

and get unified diff-style output, indicating what changes you'll need to make in order for your program to work under Python 3. The problem is that this is a one-way conversion. It tells you how to change your program so it'll work with Python 3, but it doesn't help you make your

program compatible with both 2 and 3 simultaneously.

Fortunately, there's a package on PyPI called `futurize` that not only runs `2to3`, but also provides the import statements necessary for your code to run under both versions. You can just run:

```
futurize oldstuff.py
```

and the output is (as with `2to3`) in diff format, so you can use it either to create a file that's compatible with both or to read through things.

What if you have Python 3 code and want to make it backward-compatible with Python 2? The same people who make `futurize` also make the amusingly named `pasteurize`, which inserts the appropriate `import` statements into code.

How do you know if your code really works well under both Python 2 and 3 after you have applied futurize's changes? You can't, and there is no doubt that these automatic tools will get some things wrong. For this reason (among others), it's crucial that you have a good test suite, with good coverage of your Python 2 code. Then you can run your tests against the Python 3 version and ensure that it works correctly there as well. Without these tests, you shouldn't think that your upgrade has worked; even 100% test coverage is never a guarantee, but it at least can tell you that the risk of failure has been minimized.

What if you're doing all sorts of serious and deep things with Python 2 that `2to3` can't notice, or that you can't paper over? A great package on PyPI is `six`, which papers over the differences between Python 2 and 3. For example, let's say you want to create a new object of the type used for text, such that things will be compatible across versions. In Python 2, that's going to be `unicode`, but in Python 3, that's going to be `str`. You don't want to have an "if" statement in your code each time you do this. Thus, using `six`, you can say:

```
import six
s = six.text_type()
```

Now you can be sure that "s" is an object of the appropriate type.

`six` defines an amazing array of things that have changed, which you might need to keep track of in your code. Want to check something in

the `builtins` namespace (aka `__builtins__` in Python 2)? Want to re-raise exceptions? Want to use StringIO (or BytesIO)? Want to deal with metaclasses? Using `six`, you can write a single line of code, which behind the scenes will issue the appropriate "if" statements for the version of Python you're using.

Even if you don't use `six` in your code, I recommend that you read through its documentation just to see where things have changed in Python 3. It'll open your eyes (as it did to mine) regarding the behind-the-scenes changes that often aren't discussed in the Python 2/3 world, and it might give you more insights into how to write your code so that it can work in both.

Conclusion

If you're starting to write some new Python code today, you should use Python 3. And if you have Python 2 code that you can upgrade to Python 3, you should do that as well. But if you're like most companies with an existing Python 2 code base, your best option might well be to upgrade incrementally, which means having code that works under 2 and 3 simultaneously. Once you've converted all of your code, and it passes tests under both 2 and 3, you can flip the switch, joining the world of Python 3 and all of its goodness.■

RESOURCES

Much has been written about the changes in Python 2 and 3. A great collection of such information is at the http://python-future.org website. That site offers the `futurize` and `pasteurize` packages as well as a great deal of documentation describing the changes between versions, techniques for upgrading and things to watch out for.

The `six` package is documented at https://pythonhosted.org/six. Even if you don't use `six` for 2/3 compatibility, I strongly suggest that you look through its capabilities.

Finally, if you're a web developer using Django, you definitely should read the Django-specific documentation regarding moving to Python 3 at https://docs.djangoproject.com/en/1.9/topics/python3.

This is especially important because of Django's handling of strings, bytes and Unicode strings, the names of which changed a bit over the years. Django actually includes a copy of the `six` library, modified slightly to suit its needs for internal use.

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

RETURN TO CONTENTS

Finishing Up the Content Spinner

In which {writer|columnist|hacker} Dave Taylor helps you become a spammer. Sort of.

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* (new edition coming out this summer!). 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 /

YOU'LL RECALL THAT IN MY LAST ARTICLE I shared a long, complex explanation for why spam email catches my attention and intrigues me, perhaps more than it should. Part of it is that I've been involved in email forever—I even wrote one of the most popular old-school email programs back in the day. But, there's also just the puzzle factor of taking a massive data set of millions of records and trying to produce "personalized" messages on such a large scale.

The easy version of this is to have named data fields like ${firstname}, so you can open your email with "Dear ${firstname}, I heard you went to ${college}? Me too!" and so on.

But, I'm more interested in the "spinning" side of things—the production of prose that has built-in synonyms, as exemplified by:

```
The {idea|concept|inspiration} is that each time you'd use a
{word|phrase} you instead list a set of {similar words|synonyms|
alternative words} and the software automatically picks one
{randomly|at random} and is done.
```

I know, you're likely shaking your head and wondering "what the deuce happened to Dave?", but humor me, let's explore this together as a text-processing puzzle.

In my June 2016 column, I presented the core building blocks of the article spinner, a script that could identify the {} surrounded choices, isolate them, count how many options were present and display it to the user as debugging output.

So, the above would be displayed as:

```
$ sh spinner.sh spinme.txt
The
3 options, spinning --- idea|concept|inspiration
is that each time you'd use a
2 options, spinning --- word|phrase
you instead list a set of
3 options, spinning --- similar words|synonyms|alternative words
and the software automatically picks one
2 options, spinning --- randomly|at random
and is done.
```

That's a good start, but this time, let's finish the job and actually pick randomly from the set of choices each time, output only the selected option and reflow the text to make it all look good.

Pick a Card, Any Card

The basic way to work with random numbers in Bash is to use the special `$RANDOM` variable. Each time it's referenced, it returns a randomly chosen number between 1 and MAXINT (32767). I constrain it to a specific range

by using the modulus function, so this will generate a random number between 0 and MAXVALUE:

```
randomnum=$(( $RANDOM % $MAXVALUE ))
```

The double-parent notation triggers mathematical evaluation, but you already know that, right?

To make the bottom be the value 1 instead of zero, I just add a bit more math to the equation:

```
randomnum=$(( $RANDOM % $MAXVALUE + 1 ))
```

The script already can identify how many choices are in a specific cluster (for example, "{one|two|three}"), and now we have a simple one-liner to help randomly pick one of the values. The challenge, of course, is to pick the actual string value, not just show a number!

I know, I know—work, work, work.

Halfway through the `spinline()` function (which I'll show in its entirety in just a sec), `$choices` stores the count of how many options are in the cluster, and `$source` is the set of choices, minus the open and close curly brackets.

Here's my first attempt at the random word extraction:

```
pick=$(( $RANDOM % $choices ))
wordpick=$( echo $source | cut -d\| -f$pick )
```

But, that generates an error message when run. It's not because of a typo, however—it's legit to use `cut` and specify the pipe symbol as the field delimiter—but because I haven't compensated for the 0..n selection of the random number generator: request field `-f0` from `cut`, and it complains because, well, there is no field zero.

That's easily fixed now that I understand the problem, however, and so here's version two:

```
pick=$(( $RANDOM % $choices + 1 ))
wordpick=$( echo $source | cut -d\| -f$pick )
```

Remember that modulus returns 0..(n-1) for its values, so when there are three choices, for example, `$RANDOM % 3` returns 0, 1 or 2. Add one to each, and it's back on track with the values 1, 2 and 3.

With a few useful debugging lines, here's the function in its entirety:

```
function spinline()
{
  source="$*"
  choices=$(grep -o '|' <<< "$*" | wc -l)
  choices=$(( $choices + 1 ))
  echo $choices options, spinning --- $source
  pick=$(( $RANDOM % $choices + 1 ))
  wordpick=$( echo $source | cut -d\| -f$pick )
  echo I pick choice $pick which is $wordpick
}
```

Yeah, code. Let's see what happens when I run it with the test sentence as input:

```
$ sh spinner.sh spinme.txt
The
3 options, spinning --- idea|concept|inspiration
I pick choice 2 which is concept
is that each time you'd use a
2 options, spinning --- word|phrase
I pick choice 1 which is word
you instead list a set of
3 options, spinning --- similar words|synonyms|alternative words
I pick choice 2 which is synonyms
and the software automatically picks one
2 options, spinning --- randomly|at random
I pick choice 2 which is at random
and is done.
```

It's close, actually—really close!

In fact, let's get rid of those superfluous debugging `echo` statements

(actually, I always just comment them out instead by prepending # on each line, so that if I develop the script further, and things start to go sideways, I can simply uncomment the lines and figure out what's going on).

Here's the result:

```
$ sh spinner.sh spinme.txt
The
idea
is that each time you'd use a
word
you instead list a set of
synonyms
and the software automatically picks one
at random
and is done.
```

The magic really becomes apparent when the entire output is piped through the handy `fmt` command to put all the puzzle pieces back together on the line:

```
$ sh spinner.sh spinme.txt | fmt
The idea is that each time you'd use a word you instead list a set of
synonyms and the software automatically picks one randomly and is done.
```

Run it a second time, and it's the same concept being discussed, but the specific word choices are different:

```
$ sh spinner.sh spinme.txt | fmt
The idea is that each time you'd use a phrase you instead list a set of
alternative words and the software automatically picks one randomly and
is done.
```

So that's the program—mission accomplished.

Don't Bug Me, Man!

It turns out that there's a bug in the script; however, it's a subtle one

that is nonetheless tricky to solve: if the text to spin includes a word cluster followed immediately by punctuation, the punctuation ends up being broken.

For example, consider if I slightly modified the spinme text like this:

```
The {idea|concept|inspiration} is that each time you'd
use a {word|phrase}, you instead list a
set of {similar words|synonyms|alternative words} and the
software automatically picks one
{randomly|at random} and is done.
```

See the added punctuation immediately after the word cluster on the second line? Here's what happens if I run this through the spinner script:

```
The inspiration is that each time you'd use a phrase , you instead list
a set of similar words and the software automatically picks one randomly
and is done.
```

See the problem? There shouldn't be a space before the comma. That's easily fixed with a `sed` statement, but it's an instance of a bigger problem, so rather than `sed 's/ ,/,/g'`, I'm going to leave it to you, dear reader, to try to come up with a more generalized solution that takes into account all punctuation, including sequences like:

```
({cat|dog})
```

so that they'll be formatted properly in the final output.

And, that's a wrap for this article. For my next article, I'll look at, um, something or other. Perhaps it's time to start another game script?■

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

RETURN TO CONTENTS

Secure Desktops with Qubes: Extra Protection

Find out how to make Qubes even more secure with a few advanced features.

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

THIS ARTICLE IS THE FOURTH IN MY SERIES ABOUT THE QUBES OPERATING SYSTEM, a security-focused Linux distribution that compartmentalizes your common desktop tasks into individual VMs. In the previous articles, I gave a general introduction to Qubes, walked through the installation process and discussed how I personally organize my own work into

different appVMs. If you haven't read these earlier articles, I suggest you do so before diving in here. In this article, I focus on some of the more advanced security features in Qubes, including split-GPG, the usbVM and how I control where URLs open.

Split GPG

One of the concerns I've always had with my Linux desktop was with the fact that I had a copy of my GPG key on the desktop so I could use it for my email. Of course, my key was password-protected, but I still worried that if my computer ever were compromised, an attacker still could grab it and try to brute-force it.

You can use a number of different techniques to protect GPG keys, including methods that store the master key offline in an air-gapped system while you use subordinate keys. Qubes offers a pretty novel approach to GPG key security with its split GPG system that acts kind of like a poor man's Hardware Security Module. With split GPG, you store your GPG key in a highly trusted appVM, such as the vault VM Qubes installs by default. The vault has no network card at all and is designed to store sensitive files like GPG keys. When an application wants to access the GPG key to encrypt or decrypt a file, instead of calling GPG directly, it calls a wrapper script that is included in the Qubes default templates. This script asks for permission to access the vault GPG key for a limited amount of time (which means you get an unspoofable prompt on your desktop from the vault), and if you grant it permission, it will send the GPG input (data to encrypt or decrypt, for instance) to the vault. The vault then will perform the GPG operation and send the output back to the appVM. In this way, the key always stays within the vault, and the appVM never sees it.

Split GPG is relatively simple to set up with the main challenge being that because it relies on a wrapper script, any application that calls out to GPG will need to be changed to point to a script like qubes-gpg-client or qubes-gpg-client-wrapper (the former works on the command line, which preserves environment variables, and the latter works better in programs that don't preserve them well, like email clients). Qubes has great documentation on split GPG on its website that includes some diagrams, detailed explanations of how it works and its limitations, and a how-to

guide with sample configs for Thunderbird and Mutt.

The first step is to make sure that the qubes-gpg-split package is installed in your appVMs (you should be able to use your regular package manager to install it if it isn't) and that qubes-gpg-split-dom0 is installed in dom0 (if not run `sudo qubes-dom0-update qubes-gpg-split-dom0` in a dom0 terminal).

Once the software is installed, put your GPG keyring into the appVM you want to use as your vault. I recommend the default vault unless you have more advanced and specific reasons you want to use another appVM. Execute some GPG command-line commands from a terminal in the vault (such as `gpg -K`) to confirm that the GPG key is installed correctly.

Now to use split GPG, just set the `QUBES_GPG_DOMAIN` environment variable to the name of the appVM that has your GPG key, and then you should be able to run qubes-gpg-client from that appVM with the same kind of arguments you normally would pass GPG. For applications like mail clients that may not be able to load that environment variable, you must use qubes-gpg-client-wrapper instead. This script is configured to read the contents of the /rw/config/gpg-split-domain file to determine which appVM to use, so be sure it contains the name of your vault:

```
$ sudo bash -c 'echo vault > /rw/config/gpg-split-domain'
```

That's pretty much it for basic GPG usage. The one major use case it doesn't cover is importing keys from an appVM back to the vault. You want to import keys in a trusted way, so Qubes provides a different script for this purpose that will prompt you from the vault in an unspoofable window before it imports the key. To import a key, just use this:

```
$ export QUBES_GPG_DOMAIN=vault
$ qubes-gpg-import-key somekey.asc
```

That should be enough to get you started with split GPG, but if you need particular split GPG configuration examples for applications like Thunderbird, Mutt, and Git, I recommend checking out the Qubes split GPG documentation page at https://www.qubes-os.org/doc/split-gpg.

USB VM

One of the major risks of compromise against a personal computer is the USB port. You can find a large number of stories on the internet about organizations (including governments) who were compromised because someone plugged in an untrusted USB key. There are even some fun hardware projects out there like the USB Rubber Ducky that provide what looks like an innocent USB thumbdrive but can act like a USB input device when you plug it in, and with its scripting language, you can program it to type whatever compromising keystrokes you want against your victim (including waiting some time later before unleashing your payload).

Given that just about anyone can create a malicious USB device now, you definitely want to be careful about what USB devices you plug in. Even Qubes installs may suffer the same risk, because by default, the dom0 VM is assigned the USB PCI controllers, so if you mistakenly plug in an infected USB key, it could potentially compromise your whole machine. Thankfully, Qubes provides a countermeasure for this with the option of creating a special USB VM that is assigned all of your USB PCI devices. With the USB VM in place, if an attacker plugs a malicious USB device in to your computer while you are away (or you plug it in yourself), the damage is contained to the USB VM.

Of course, if all of your USB devices are now assigned strictly to one VM, how can you use them on your other appVMs? For input devices like mice and keyboards, Qubes provides an input proxy service that will proxy input devices to the rest of the appVMs provided the user accept a prompt when the devices are plugged in. When you plug in a USB storage device, it shows up only in the USB VM for starters, and you then can assign it to other appVMs in the Qubes VM Manager by right-clicking on the appVM and selecting the device from the attach/detach block devices menu (be sure to detach it before you unplug it, otherwise Xen has been known to get confused about the state of the block device).

If you do want to enable the USB VM, the sys-usb USB VM shows up as an option during the install on the screen where you select which default appVMs to load. Otherwise, if you want to try it out post-install, you can run the following commands from the dom0 VM (Qubes 3.1 or newer):

```
$ qubesctl top.enable qvm.sys-usb
$ qubesctl state.highstate
```

These commands will run through an automated Salt script the Qubes team has put together that will configure the sys-usb VM appropriately. Of course, if you want to do this all by hand, you also could just create your own sysVM (I recommend not giving it a network card if you can help it), and in the Qubes VM Manager, go into that VM's settings and identify and assign your PCI USB controllers to it.

Now, there's a reason that sys-usb is disabled by default in the installer. Although desktop computers still offer PS/2 ports and many laptops use PS/2 as the interface for their main keyboard and mouse, some laptops (such as current MacBooks for instance) use a USB interface for the main keyboard. If that's the case, you can end up with a situation where you are locked out of your computer, because your USB keyboard will be assigned to your USB VM at boot, and you won't be able to log in. Another downside is that although there are services to share input devices and storage devices with other appVMs, any other USB devices (such as webcams or network cards) cannot be shared and can be used only from applications within the USB VM. Finally, the USB VM is unstable on some kinds of hardware, depending on how well it supports Qubes.

By default, only mice are allowed through the Qubes input proxy (and then only if you accept a prompt). Keyboards are not allowed through by default, because of the extra risk a malicious keyboard input device can pose to a system, including the fact that the USB VM can then read anything you type on that keyboard in other appVMs (such as passwords) or could enter its own keystrokes. If you are willing to accept this risk, you still can provide a level of protection by ensuring that you are prompted before an arbitrary USB keyboard is shared with the rest of the environment. In a dom0 terminal, add the following line to /etc/qubes-rpc/policy/qubes.InputKeyboard:

```
sys-usb dom0 ask
```

In this case, I specified sys-usb, but if you use a different appVM as your USB VM, use its name here instead.

The one big challenge you may find if you use a USB VM is in video conferences. Since basically every webcam shows up as a USB device, and there currently is no way to share that USB device with other VMs, you

are stuck using your webcam with apps in the USB VM only. A further challenge is that, by default, the sys-usb VM has no network access, and because it's a system VM type, you can't just point it to your sys-net VM to grant it network access. Instead, if you want to video conference with a USB VM, you unfortunately must replace the existing sys-usb VM with a new one that is either an appVM or proxyVM so it can get network access. Then you will have to install and run your video conferencing software from within that VM.

The major downside to this approach is that the USB VM is traditionally considered an untrusted VM much like sys-net, yet video conferencing is a somewhat trusted application, because you have to provide login credentials. Whether the security risk is worth the benefit is something you will have to decide for yourself based on your threat model. Fortunately in the future, there is hope that once a new USB virtualization feature hits Xen (and Qubes), you may be able to treat webcams like input or storage devices within Qubes.

URL Handlers

One final enhancement you can make with your Qubes desktop that not only helps with security but also ease of use is the Qubes CLI tools to open URLs in different appVMs. One common use case for this is to configure your email client to open attachments in a disposable VM (something Qubes covers for mail clients like Mutt on its main documentation page). In addition to that, I like to set the default URL handler for appVMs that aren't meant to run a web browser to be my general-purpose untrusted appVM. The untrusted appVM has no personal files or anything really of value to me on it, and I don't enter any login credentials into that appVM. Therefore, I can destroy it and re-create it at any time.

The default URL handler is a setting that you have to make in each appVM. The way I set it is to go to the shortcuts editor for that appVM (by clicking the "add more shortcuts..." option in that appVM's desktop menu) and adding the Preferred Applications program to that appVM's shortcuts. This is the same program that you use on a regular desktop to choose Firefox or Chrome as your default web browser. Once I've added the shortcut, I then launch the program, and in the section where I can

set the web browser (Figure 1), I choose Other so I can type in my own custom command, and in that field, I type:

```
qvm-open-in-vm untrusted "%s"
```

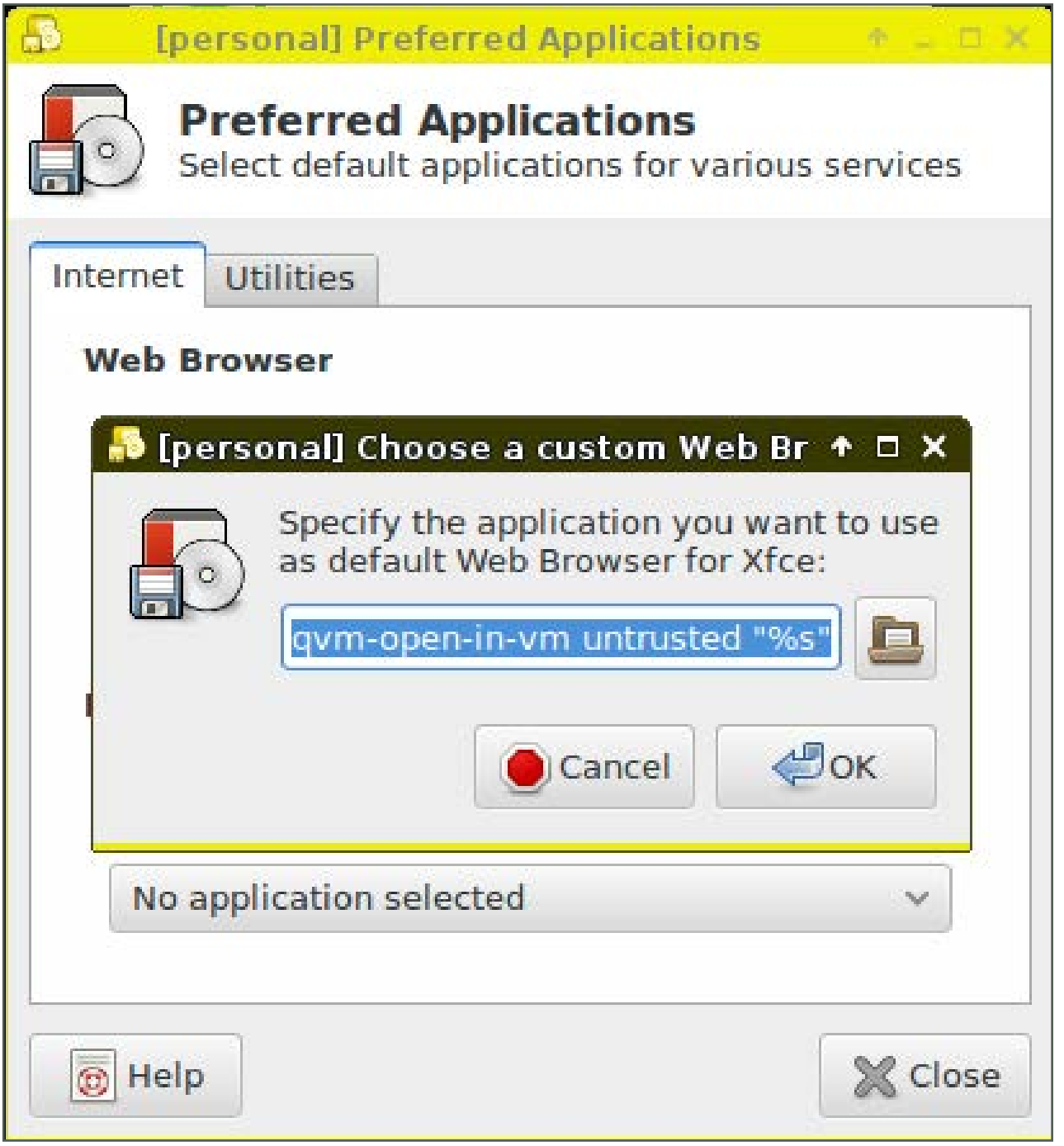


Figure 1. Changing the Default URL Handler

Once I save that, the next time I click on a URL within that appVM (for instance, in a chat session), I will get a prompt from the untrusted appVM to allow this appVM to open a URL there. You either can allow that every time or select "Yes to All" to allow this appVM to open URLs in untrusted without prompting permanently. I use a similar method to open URLs from my KeePassX password manager in my personal-web appVM instead of untrusted by going into the KeePassX settings and configuring a custom URL handler there instead. If you would rather open URLs in a disposable VM, just change this command to be:

```
qvm-open-in-dvm "%s"
```

Although Qubes definitely is very secure by default, these additional settings will help you lock it down even further, and each one gives a prime example of how the principle of compartmentalization can help you isolate and protect your computer. This is the last article I intend to write in this Qubes series for now; however, I likely will come back to more specific Qubes how-tos in the future.■

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

RETURN TO CONTENTS

Sometimes My Office Goes with Me

Traveling doesn't have to mean time off work.

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

LAST YEAR ABOUT THIS TIME, I wrote a short article about my "Network Go Bag". During the past year, I've gotten lots of email about that bag and actually quite a few questions about working while traveling in general. This month, I'm traveling again, so I thought I'd put together the "How I Do It" information into a single article. I often write about what I use, so you've probably already heard of some of these things, but nonetheless, here it goes.

Work Area

Usually, if I'm staying at a hotel while traveling, there will be some sort of desk in the room that I can take over as a temporary workstation. The room I'm in this week doesn't actually have a desk, so I had to run

Figure 1. Yep, that's a party-size bag of saltwater taffy—and about 50 empty wrappers.

to the local department store and buy a table. Figure 1 shows the table I bought. It was $29, and I have no idea how I'm going to fit it into the car to bring home. You can't see the chair in the photo, but it's a folding chair I "borrowed" from a conference room. Metal folding chairs hurt your rear end after a while, so for a cushion, I'm sitting on a bag of dirty clothes. Yes, I travel in style, and I'm super classy.

The other side of the room has a little nightstand (Figure 2), which was oddly placed nowhere near the bed. That's okay, because I used the table to set up a coffee station. The room also didn't have a coffee pot, so I picked up this single-serve Keurig-compatible coffee pot for $29. It's terrible. Seriously, if I reviewed coffee pots for a living, this would be my example of the worst coffee pot I've ever used. Still, I can't work in a room without coffee, so for the week, I'll just suffer. But I'm leaving this

Figure 2. Coffee and ramen—the fancy creamers are for my wife.

pot here; I don't want it at home. (And yes, I don't eat terribly healthy on the road. Ramen is about it. You can cook only so many things with a subpar, single-serve coffee pot.)

Hardware

I won't go back over all the stuff in my Network Go Bag, but you can read about it in the August 2015 issue if you're really interested (http://www.linuxjournal.com/googlesearch?s=My%2520Network%2520Go-Bag). Basically, I carry a router, a Wi-Fi extender, network cabling and extension

cords. I've been in a situation where I tied my Wi-Fi extender inside a grocery bag with a lithium-ion battery and happily had Wi-Fi access in a cabin outside of range. I was my own superhero.

Once the gasping stops (yes, that's an 11" MacBook Air laptop shown in Figure 1). It's the best laptop I've ever owned, and I'm willing to take the shame for using an Apple product. First, it has the nicest keyboard I've ever used on a laptop. (I can't say the same for the newer MacBook 12" keyboard.) It's tiny, has 8GB of RAM, an i7 processor and a big 512GB SSD. I can run VMware Fusion and host Ubuntu MATE 16.04, and it runs better than it does natively on any other laptop I've owned. Plus, I'd be lying if I said I didn't use MacOS as well. Sometimes when I'm presenting, the only "projector" access is an AppleTV using AirPlay. With the MacOS running, I can share my screen and do all my demos inside VMware. Plus, the aluminum chassis is so thin, I probably could sword fight ninjas with it, should the need ever arise.

Apart from the laptop, I sometimes have to print handouts (I do this week), so I have a USB laser printer I tote along (also shown in Figure 1). I know there are truly portable printers, but since I almost always drive, something a little bigger isn't a problem, and it's a whole lot cheaper. This model is a Brother HL-L2320D, and it prints fairly fast. It also has a duplexer, so really, for less than $100, I'm not complaining.

For mobile data when Wi-Fi isn't available, I used to carry a MiFi device everywhere I went. It was awesome. Since at that time I worked for a school, I could literally get *unlimited* 4G data, and it cost the school $35/month. That unlimited deal was available only for schools, and thanks to people like me, it's probably not available even for schools anymore. Nowadays, I just tether to one of my phones. I always carry two. My day job provides an iPhone 6s on the Verizon network, and I have a personal Moto-E using Cricket Wireless (Figure 3). Since I have both AT&T and Verizon networks at my disposal, I almost always have connectivity. If I do much more traveling, I'll invest in one of those "Karma Go" devices (http://www.yourkarma.com), so I have access to the Sprint network in a pinch. So far, I haven't needed it, but if I ever decide to do an extended road trip, I'll probably have to do that.

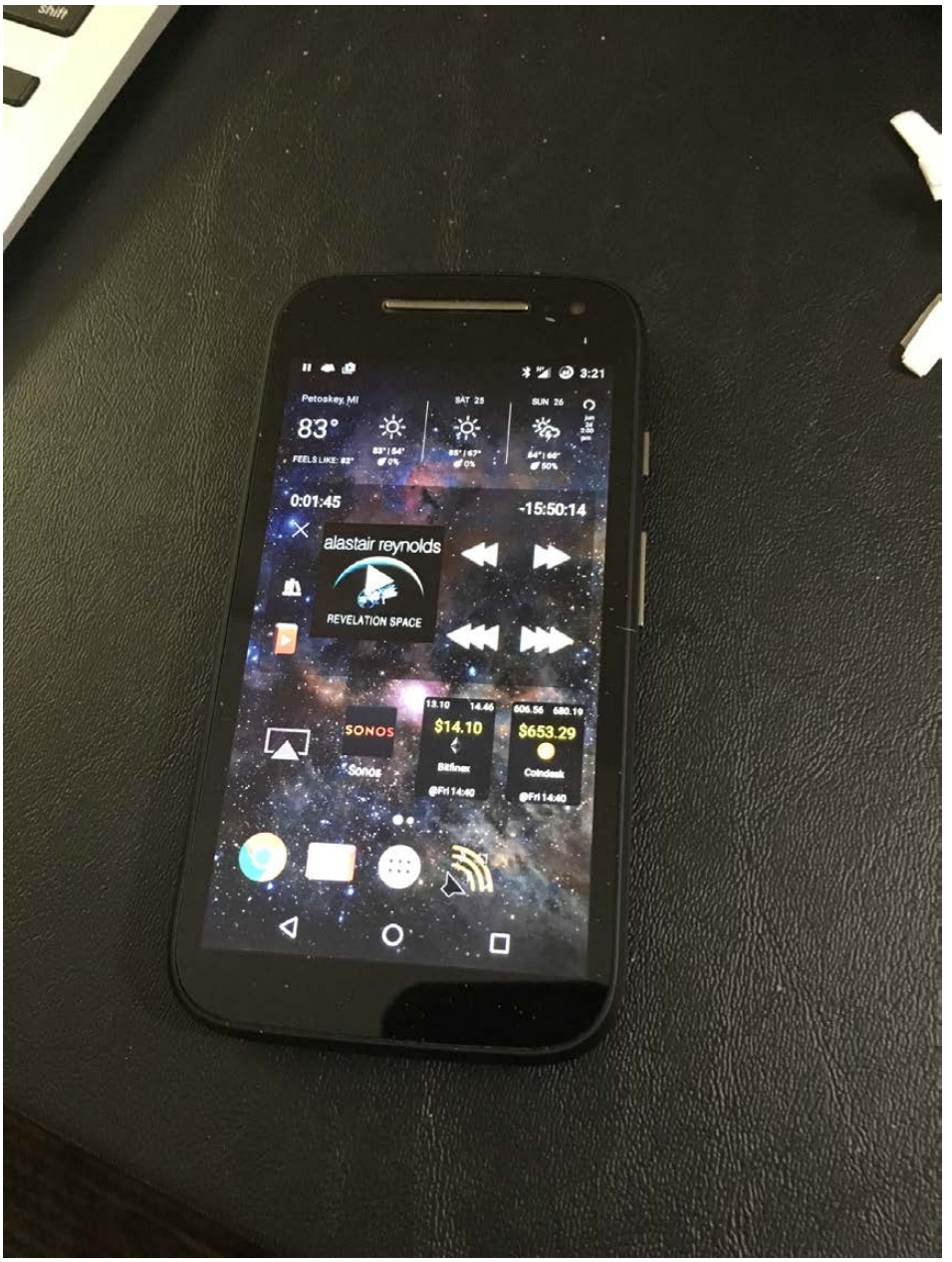


Figure 3. I've honestly never spoken on this phone. I don't even know if the phone part works!

The only other things I carry with me, hardware-wise, are power-related. I have a 12v inverter for the car that gives me 120v AC on the go. I also have a big 24,000mah lithium ion battery (I got it from woot.com on special) charged up and ready for dying devices. My Anker five-port USB charger (Figure 4) is still the charger I use for mobile devices, and with that combination, I've never run short on juice. I kind of want to get a portable solar panel so I can top off my batteries during sunny days, but I don't really need that. I just want it!

Figure 4. I reviewed this back in the September 2014 issue. It's the Anker 40w five-port charger, and it's awesome.

Software

I've talked about software before, but things change from time to time, so my most current batch of "must-have" software includes the following:

- BittorrentSync: this works amazingly well. I have it installed on all of my computers and laptops, and also on my big file server at home. My goal is to move all my documents to BittorrentSync.

- Dropbox: I'm still a little chicken, and I haven't moved all my

For some reason, two sentences into an article I find myself SSH'd into multiple servers, playing with Docker containers.

documents over to BittorrentSync. There's really no reason I haven't, except for the occasional time I need to share a file with a public link. BittorrentSync doesn't do that yet (I don't think it does anyway), and I've used Dropbox for so long, it's hard to switch. Plus, I have approximately 24GB of free Dropbox storage that I earned with a slightly shady Google Adwords campaign, so I feel like I should take advantage of my ill-gotten gain.

- Bean: yes, Bean the OS X text editor. I use Bean to write articles for two main reasons. First, it has a running word count on the bottom of the window, so I know exactly how much I've blabbered. I can't find any decent text editor that does the same on Linux. All the ones I've tried get annoying after a little while. Second, when I try to do work on Linux, I get distracted. For some reason, two sentences into an article I find myself SSH'd into multiple servers, playing with Docker containers. When I'm using the native MacOS, I tend to get more actual work done.

- Crashplan: because backups aren't just important, they're vital. Did you notice that in last month's issue there wasn't an Open-Source Classroom column? That's because I had a system crash, and my work hadn't been backed up yet. The world might never know the mysteries of running a syslog server, because I don't think I have the strength to write my article over again.

- Final Cut Pro: I know, I know, you're losing faith in my Linux passion. Here's the deal, Kris Occhipinti—I've mentioned him often through the years, as he's a fellow reader and friend (Figure 5)—somehow gets his Linux video editing software to work amazingly well. Either I expect too

Figure 5. He's even better looking than me. You can find his videos at http://www.filmsbykris.com.

much out of an editing package, or he is a much better technologist than I am. Since all I need from an editor is to overlay some text for the intro video every month, I suspect he's just better than I am. Nonetheless, for now, I use Final Cut Pro because it works.

- TeamViewer (free version): this works on Linux, Mac and Windows, and it works amazingly well through NAT services. I know there recently was a data breach where all the logins and passwords were stolen, but as long as you don't leave it set for unattended access, it's a perfect way to help family members while you're away. In fact, my home router stopped forwarding SSH traffic, and I was able to use TeamViewer to log in to my office computer with the help of my daughter and fix the router remotely.

- Evernote/Simplenote: I use Evernote for storing pretty much everything. The more I store, the more useful it is. I just discovered Simplenote, however, and for text, I might switch over. Based on my "switch" from Dropbox though, I'm not holding my breath, but I really do love the

simplicity of Simplenote. Plus, it's open source now, and that does matter to me.

Entertainment

It will likely come as no surprise that going out "partying" isn't really my cup of tea. A nice cup of tea is more my cup of tea. So when I'm traveling, I tend to bring along my own entertainment. I have a bunch of different options, because I tend to be fickle.

- Books: I usually take an actual dead-tree book or two with me whenever I go somewhere. I'm currently reading *How to Keep Your Volkswagen Alive* by John Muir because I recently bought a 1975 Volkswagen Beetle. It turns out I'm not much of a mechanic, so I'm reading a book about it—you know, like any nerd does. Can you learn to maintain a car by reading a book? I'll find out soon (Figure 6).

- Kindle Paperwhite: I know there's a new model of the Kindle available, but my Kindle Paperwhite works fine for me. I don't actually read as much as I'd like on it, but I keep forcing myself to use it, because I know it's the way of the future. I usually keep a ton of sci-fi and fantasy books on it, which I've stored, converted and uploaded via Calibre. I also keep my entire Calibre library on my laptop, so I always can add more books from my collection if I don't want to buy one from the Kindle store.

- Listen: most of my "reading" is done via audiobook. I love Audible, but I hate its app. Listen is hands-down the nicest, most incredible audiobook player I've ever used. It's an Android-only app and worth every penny. (I think it was $3.) I use FolderSync on Android to keep my audiobook folder up to date, and I just copy the audiobooks I want into a sync folder on my home server. Every night it does an SSH/rsync copy of books to my phone. Oh, and I use an LG HBS-730 Bluetooth headset for listening to the books. I imagine I could talk on the phone with the headset too, but I've never talked on that phone because who actually "talks" on phones anymore? (Okay, it's because my day-job phone has the number everyone calls.) Anyway, the combination of Listen, FolderSync and those LG headphones supplies about 90% of the entertainment on any given trip for me.

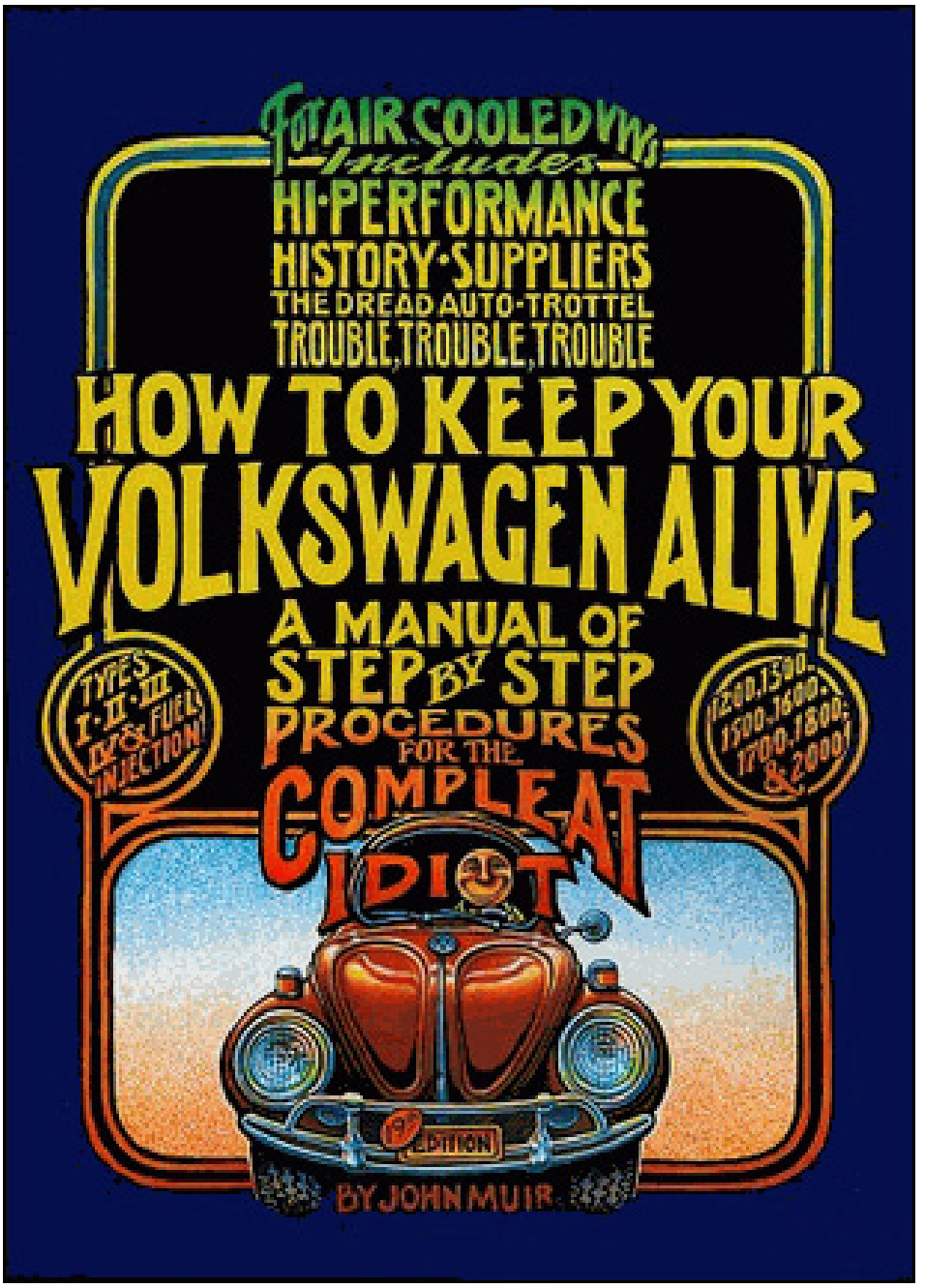


Figure 6. Only I could make auto mechanics nerdy.

- Plex: I seem to write about Plex every other issue. I'd apologize, but oh my goodness is Plex awesome. Now it even syncs photos from my family's phones, so we can all see each other's uploaded pictures if we want, even on the big screen. Nevertheless, Plex is perfect for remote entertainment, but beware of using cellular data to watch movies, because it eats up data quickly.

- Fox Sports Go: what? Sports? Yeah, I know. I'm going to have to

turn in my nerd card. First I use a Mac, and now I watch sports? It's really only one sport—baseball. I'm not sure if it's because of all the statistics, or because I can watch a baseball game while doing something else and not miss anything. I really love watching baseball, and the Fox Sports Go app might suck, but it lets me stream baseball games, even when I'm in the "blackout" areas.

- MLB App: sometimes I don't have the bandwidth to stream the video of a baseball game, so I use the MLB app to listen. Seriously, I have no idea why I like baseball so much, but I do. Go Tigers!

That and Pants!

Apart from regular mundane things like dad shorts (cargo shorts—I'm pretty sure dads everywhere have agreed to wear nothing else) and

Figure 7. Someday you'll be mine.

flip-flops, that's about all I take while I'm traveling. I do usually take a backup laptop in case something goes wrong and, of course, a pair of emergency pants. If you've ever presented to 1,000 people with coffee-stained pants, you know exactly what I mean.

One of these years I'd like to take an extended road trip with my wife and work from the road while we travel. I'm sure my go-bag will change significantly during that trip. All I need to do is buy that 1970s Volkswagen Westfalia camper (Figure 7) and convince my wife it'll be the trip of a lifetime! If you have any special travel oddities or tips, I'd love to hear about them. And heck, if you have a 1970s Westfalia camper, I'd love to hear about that too! Drop me an email at shawn@linuxjournal.com.■

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

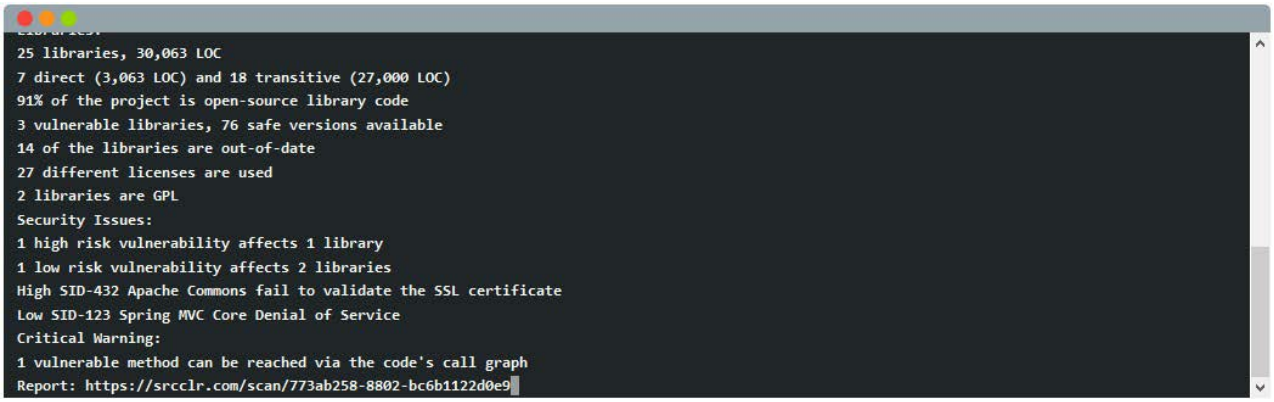
RETURN TO CONTENTS

PREVIOUS
Shawn Powers' The Open-Source Classroom

NEXT
Feature: The Tiny Internet Project, Part III

SUSE Enterprise Storage

The conundrum for many enterprise-level businesses is that the demand for data storage is significantly outpacing the shrinking price for storage. The upshot for many is that budgets for storage are growing faster than data demands. Fortunately SUSE wants to reduce your storage costs, which is why it released the new SUSE Enterprise Storage 3. SUSE's updated intelligent software-defined storage solution is touted as the first commercially supported solution based on the Jewel release of CephFS, which enables enterprises to transform storage infrastructure to reduce costs while providing unlimited scalability. SUSE Enterprise Storage users can transform their storage infrastructure and seamlessly adapt new technologies using cost-efficient, resilient and redundant storage infrastructures on commodity hardware. Other innovations in version 3 include multisite object replication to ensure replication at distance for improved disaster recovery and a new framework to simplify management by providing the foundation for an advanced GUI management tool (using openATTIC).
http://suse.com/storage



SourceClear's Commit Watcher

Someone accidentally commits private AWS keys to an open-source project and ends up handing candy to a bitcoin miner. Once committed, these secrets are easily discoverable through GitHub Search, which makes this accidental disclosure additionally dangerous. To combat this and other threats to safe use of open source, Source Clear announced Commit Watcher, a recently open-sourced tool that finds interesting and potentially hazardous commits—both accidental credential leaks and undisclosed security patches. The tool addresses two critical categories of issues found among open-source software that by nature are disclosed publicly but are also largely unknown. These are accidental disclosure of sensitive information (SSH keys, AWS credentials and so on) and security patches for vulnerabilities that are not explicitly disclosed. Companies can watch their own projects, public and private, for accidental disclosures and take remedial action as soon as possible. Commit Watcher is further backed by a comprehensive vulnerability database, SourceClear Registry, and complements SourceClear Open in the arsenal of products SourceClear has designed specifically for open-source developers.
http://srcclr.com

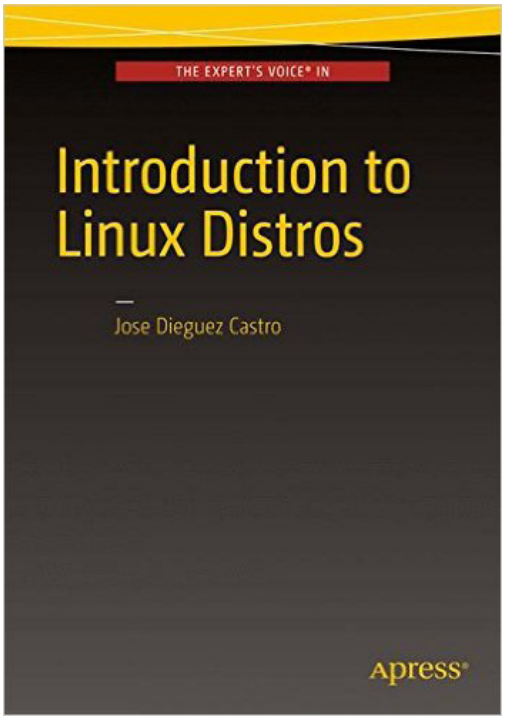
Juniper Systems' Geode

Juniper Systems' new Geode rugged sub-meter GNSS (Global Navigation Satellite System) receiver collects real-time professional-grade data but is intuitive enough for novices. Designed foremost for versatility, the Geode features one-button simplicity and can be paired with any of Juniper Systems' rugged handhelds as well as a wide range of Android devices. The Geode features a small, compact design and can be carried conveniently by hand or in a pack or mounted on a pole, depending on specific user requirements. Like other Juniper Systems' products, the Geode is built to Juniper Rugged standards and IP68-rated protection against dust and water for reliable performance in harsh environments. The receiver features an Overtime Technology battery for all-day power and a wide operating temperature range (very low and high), even conserving power in low temperatures.

http://junipersys.com

illusive networks' Deceptions Everywhere

illusive networks' bread and butter is its deception cybersecurity technology called Deceptions Everywhere whose approach is to neutralize targeted attacks and Advanced Persistent Threats by creating a deceptive layer across the entire network. By providing an endless source of false information, illusive networks disrupts and detects attacks with real-time forensics and without disruption to business. The latest illusive networks release is enhanced with Linux support, Advanced Forensics and Highly Interactive Deceptions to distract attackers from their targets further. The new forensics dashboard provides laser-focused forensics illuminating the minutes before, during and after an attack. illusive enables defenders to focus on high-quality information (as opposed to trudging through terabytes of data) and generate the report in real time.
http://illusivenetworks.com

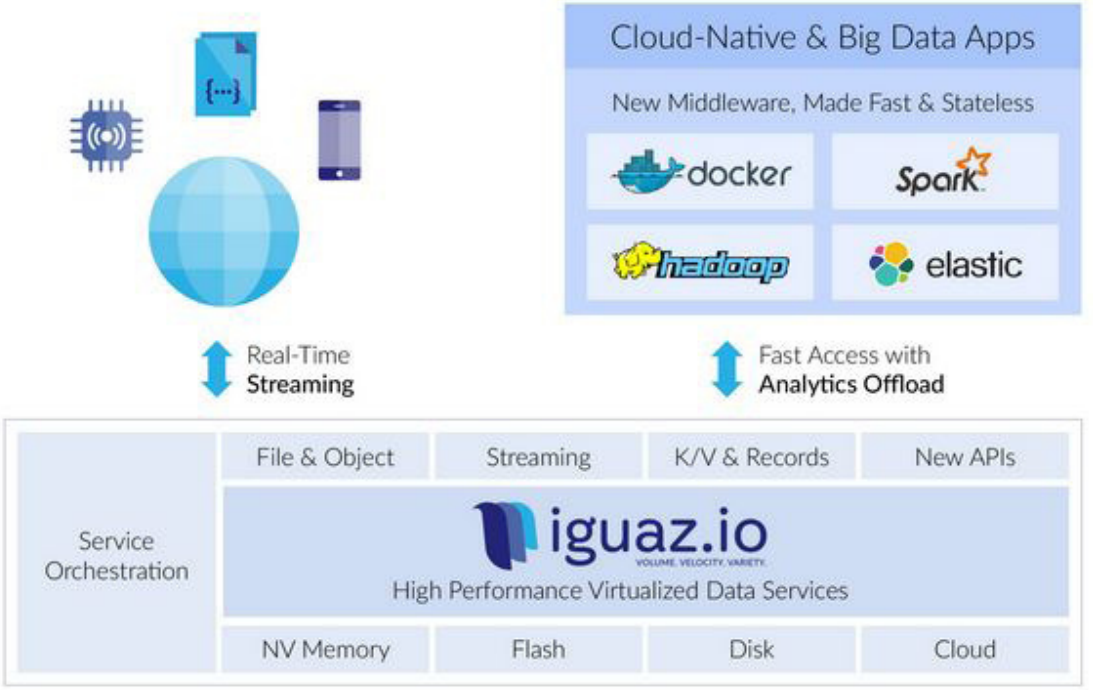


Jose Dieguez Castro's *Introduction to Linux Distros* (Apress)

Although Linux always has been a diverse ecosystem, once upon a time, just a handful of Linux distributions, or distros, existed. Do you recall Yggrasil, Trans-Ameritech and the original S.u.S.E. (which begat SuSE, SUSE and openSUSE)? Today, literally hundreds of Linux flavors exist, each with its own unique offerings. How do you choose the right one for you and your needs? A new book called *Introduction to Linux Distros* by Jose Dieguez Castro explores the pros and cons of the most frequently used Linux distributions in a concise step-by-step manner, so users can avoid hours of web surfing, countless downloads and ample confusion by new concepts and complex and marathon installation guides. Readers will benefit from the author's long-term experience working hands-on with each distro. In the book, Dieguez Castro also discusses the idea of a Linux "distro" and why so many exist, the criteria for finding the right distro for one's needs, the various Linux "family trees" and their unique "philosophies", as well as how to install, maintain and obtain support for each distro.
http://apress.com

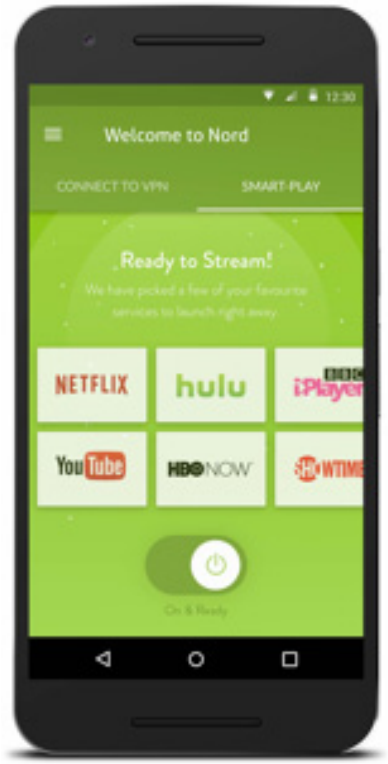
Contrast Security's Contrast Enterprise

The phrase with which Contrast Security describes the one-of-a-kind protection provided by the new Contrast Enterprise is "continuous application security". By this, Contrast Security means that Contrast Enterprise is "The first and only enterprise security software product that fully integrates the ability to find and fix application vulnerabilities during development, and monitor and block application attacks in production, all within one unified environment." Contrast Security points out that these two capabilities heretofore were available only in partially integrated solutions at best or siloed products from partnering vendors at worst. Meanwhile, Contrast Enterprise uses patented deep security instrumentation to weave vulnerability detection, threat visibility and attack protection directly into applications automatically, without requiring application changes or security experts. While these applications are running, highly accurate context is instantly generated about where applications are vulnerable and under attack. Contrast Security calls its approach "a revolution in application security for traditional development approaches, as well as for Agile and DevOps methodologies".
http://contrastsecurity.com



iguaz.io

An IT megatrend in progress involves the shift from legacy monolithic apps running on enterprise storage to systems of engagement that interact with users, collect real-time data from many sources and store it in elastic and shared data services. A self-described "disruptive" enterprise seeking to push this vision forward is iguaz.io, which recently announced a virtualized data services architecture for revolutionizing both private and public clouds. In contrast to the current siloed approach, iguaz.io consolidates data into a high-volume, real-time data repository that virtualizes and presents it as streams, messages, files, objects or data records. All data types are stored consistently on different memory or storage tiers, and popular application frameworks (Hadoop, ELK, Spark or Docker containers) are accelerated. In addition to a 10x to 100x improvement in time to insights at lower costs, iguaz.io says that its architecture provides best-in-class data security based on a real-time classification engine, a critical need for data sharing among users and business units.
http://iguaz.io



NordVPN for Android

The prospect of privacy protection and occulting your smartphone's IP address with a VPN are sufficient selling points, but the ability to watch your Spanish-dubbed Turkish telenovellas while on the beach in Tahiti should seal the deal for real. Better security and access to geo-blocked content are part and parcel of NordVPN, a new Android app designed to bring VPN (Virtual Private Network) services to a larger public with its simplicity to set up and use. NordVPN says that its apps have received wide acclaim from critics and users for their "breakthrough usability and design". The app can connect to the desired destination quickly by simply clicking on the country name, as it automatically selects the fastest server available. Additional NordVPN features include service on up to six devices with one account, a choice of more than 550 servers worldwide, 24/7 customer support, an automated kill switch, integrated access to SmartPlay (an encrypted SmartDNS), double VPN, Tor over VPN, anti-DdoS, Ultra Fast TV Servers, a Netflix shortcut and more.

http://nordvpn.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

THE TINY INTERNET PROJECT, Part III

Deploy DNS, mail, web and Linux mirror servers using VM templates.

JOHN S. TONELLO

PREVIOUS
New Products

NEXT
Feature: Coroutines and Channels in C Using libmill

In the May 2016 issue of *LJ*, I introduced the Tiny Internet Project, a self-contained Linux project that shows you how to build key pieces of the internet on a single computer using virtualization software, a router and free open-source applications. In the second installment in the July 2016 issue, I explained how to set up the host server using Proxmox and build a first basic Ubuntu 14.04 virtual machine. In this third installment, you'll learn how to set up an Ubuntu mirror, a DNS server, a mail server and a web server. If you missed Parts I and II, be sure to visit the *Linux Journal* archive and read them there.

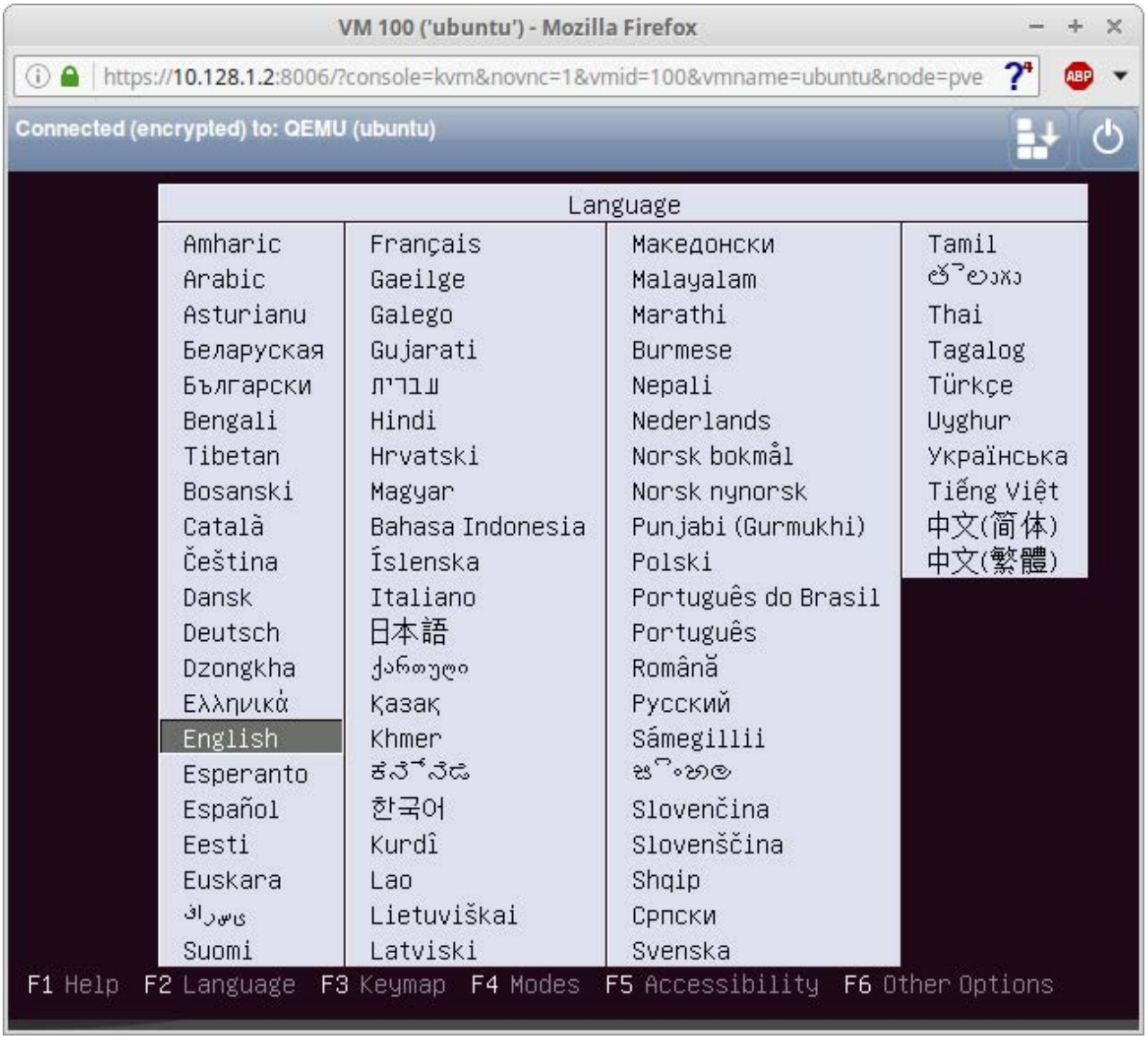


Figure 1. Ubuntu Installation Screen—Selecting Your Language

As you finished with Part II, you hopefully had just booted a raw Ubuntu 14.04 server VM. Now, I'll describe how to customize that VM with some user accounts and software, keeping it fairly generic, but ready to become a template for most everything else you'll build.

Initially, you'll do all your work from the Proxmox web interface on your Proxmox server: https://10.128.1.2:8006.

Log in and start the Ubuntu VM you made, which probably was named "100 (ubuntu)". Wait a moment for it to boot, and click the Proxmox Console button to launch what is essentially a web-based terminal.

When the shell opens, you'll see the Ubuntu installation screens. Select your language and choose "Install Ubuntu Server" from the action list.

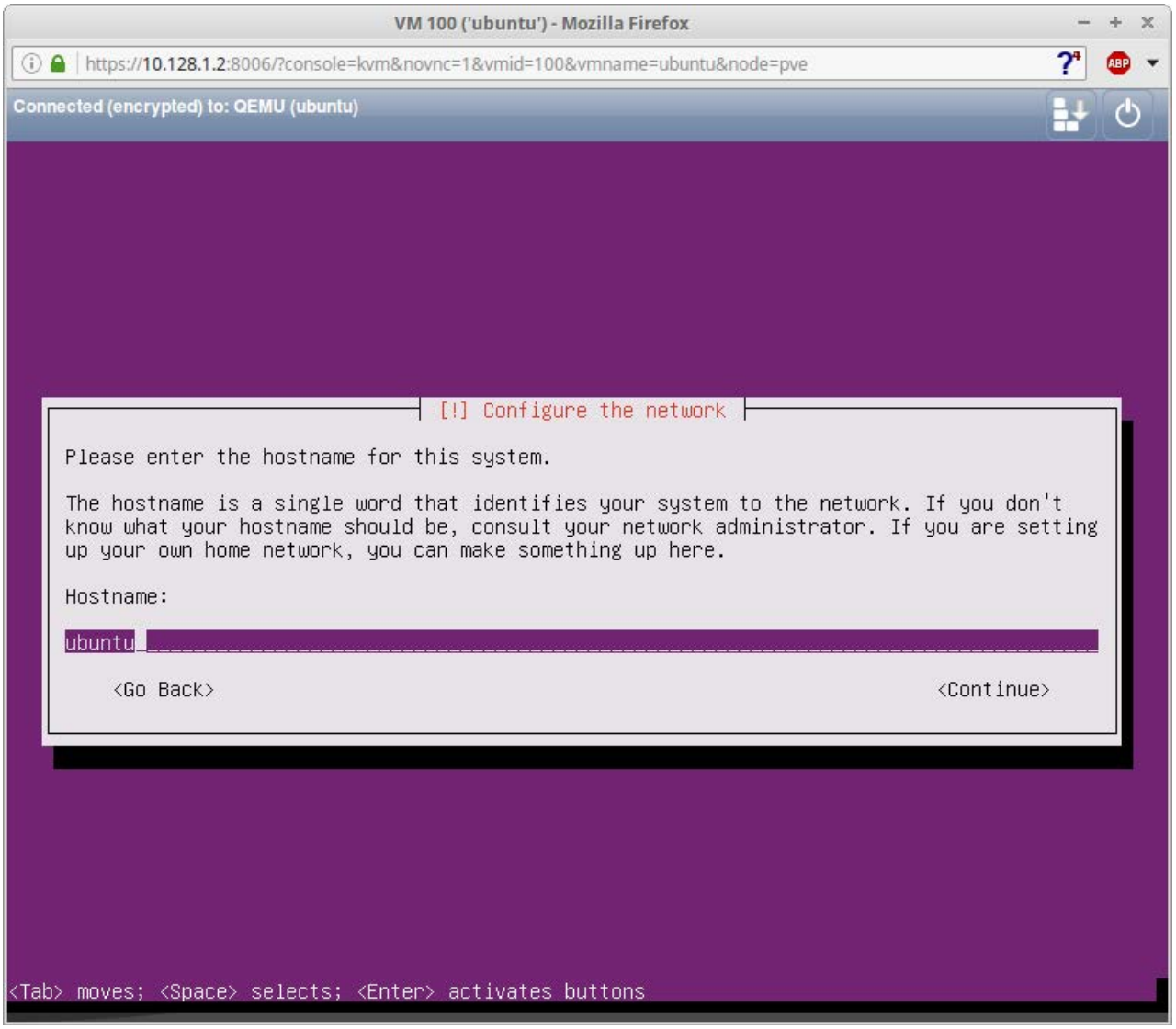


Figure 2. Entering a Hostname

You'll be prompted again for language choices and keyboard layouts; choose the ones that suit your needs. The installer will detect your network and prompt you to enter a hostname.

Since you'll be making this VM a template, give the machine a generic hostname like "ubuntu". That way, if you later deploy a different type of server (say, ArchLinux), you'll easily be able to tell them apart.

When you're asked to create a user name, choose something that follows a naming convention you can use for all future users, such as your first initial and your full last name. Then when you need to figure out user names (and email addresses) later, you won't have to guess.

Provide a password, add encryption if you like, set your time zone and

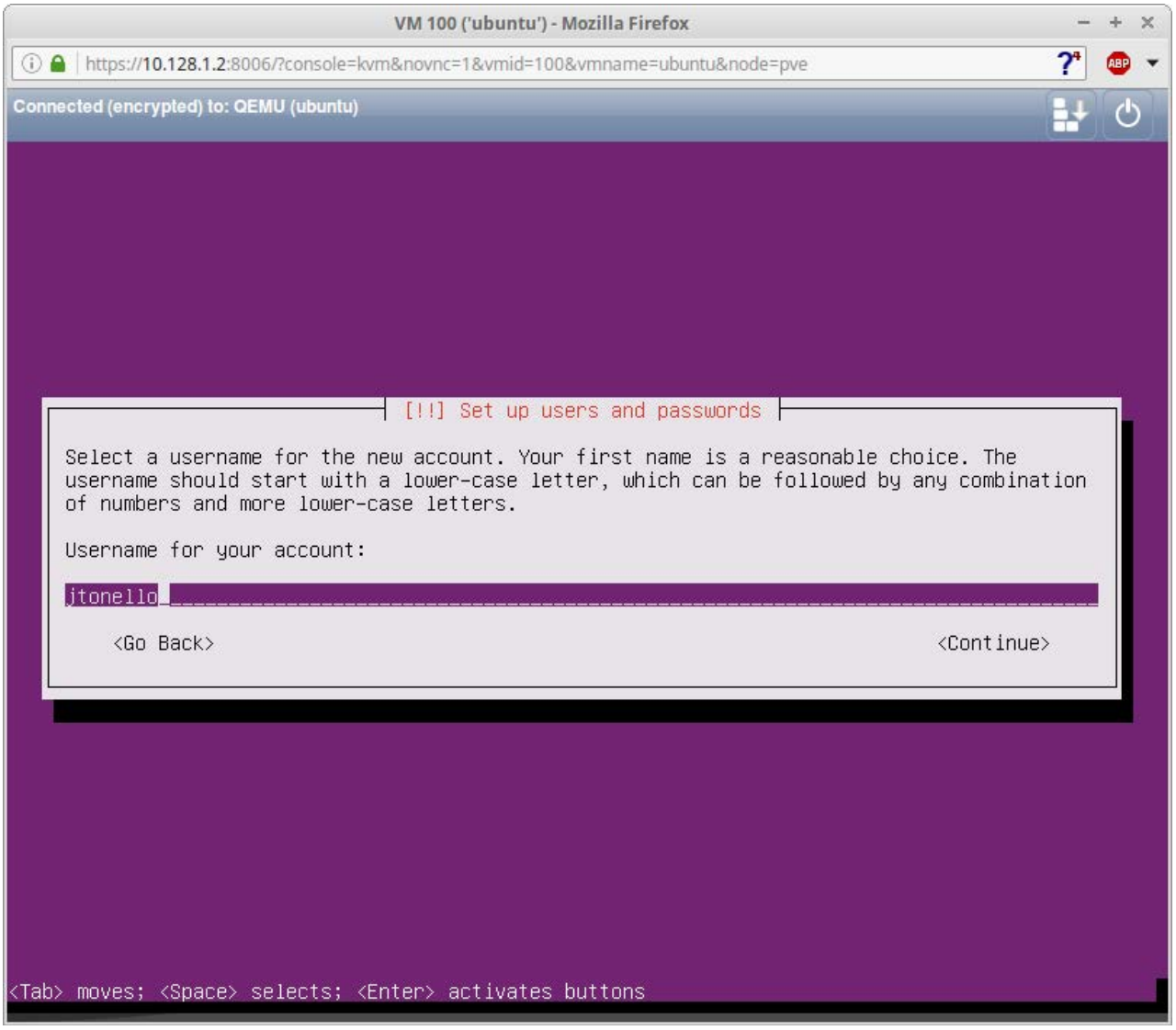


Figure 3. Selecting a User Name

proceed to the disk partitioning.

When you first created this VM under Proxmox, you gave it a main virtual disk, which is what the Ubuntu installer now sees. Select "Guided — use entire disk", not the default with the LVM option. Accept the configuration and then write the changes to disk.

The installer will set up the system, which takes a few minutes. If you've ever installed an operating system onto hardware from a DVD, this is the same thing.

When you're prompted for HTTP proxy information, leave it blank and continue unless your school or situation requires a proxy to access the public internet.

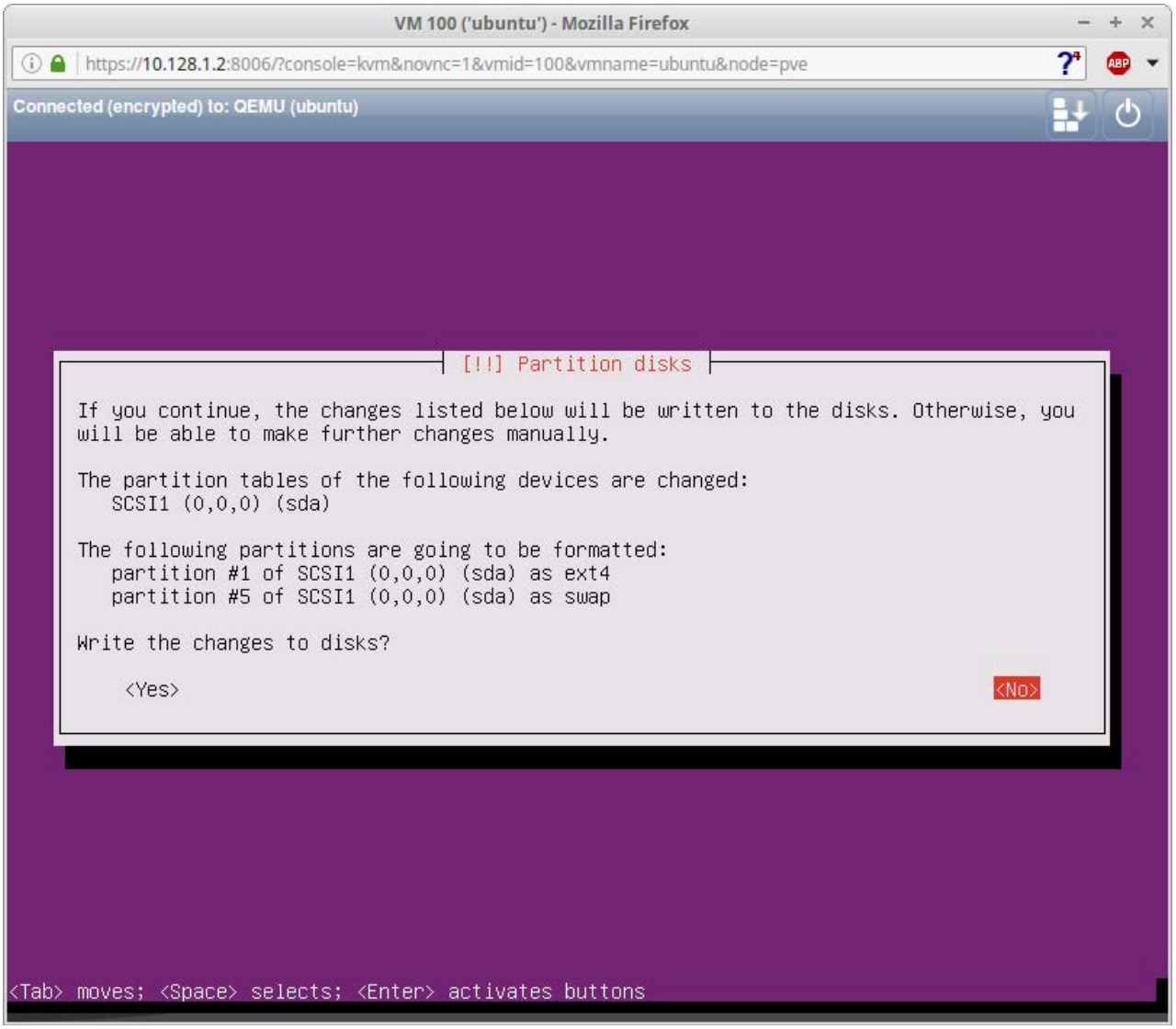


Figure 4. Disk Partitioning

Once the installer configures `apt`, it'll set up all the base software and prompt you about how to manage upgrades. Select "Install security updates automatically" and continue. On the Software selection page, select only "OpenSSH server" for now. Doing so will give this base VM template direct shell access and won't load up the machine with packages you don't need.

When prompted to install the GRUB bootloader to the master boot record, choose "Yes", and reboot when prompted. The VM will launch into the GRUB menu quickly, boot and drop you at the login prompt.

Log in using the user name and password you set up during the installation and check the system's IP address. It was dynamically

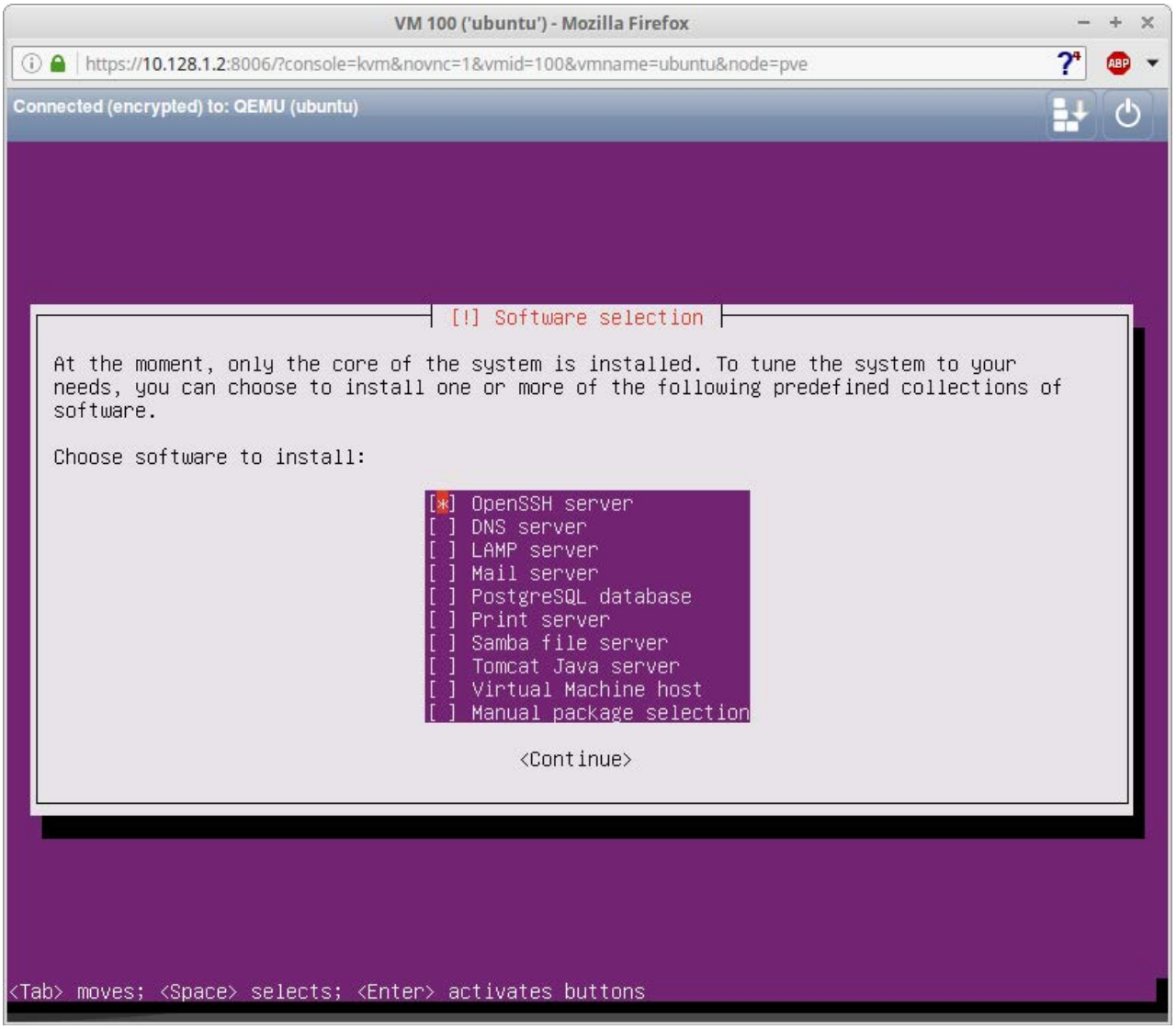


Figure 5. Software Selection—Selecting Only OpenSSH Server

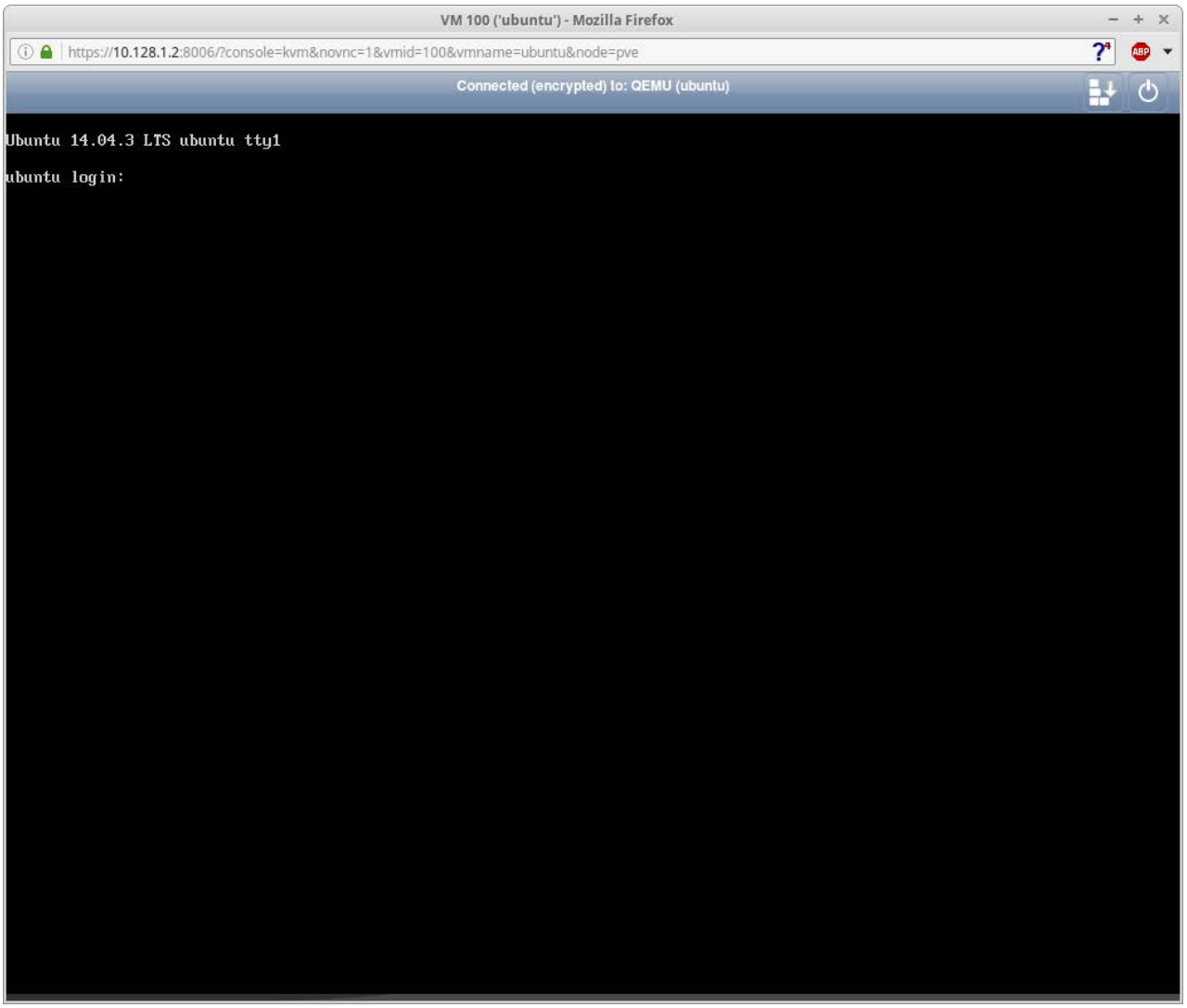


Figure 6. Login Prompt

configured with DHCP during the installation, and it will be handy to know so you can `ssh` into your new VM:

```
$ ifconfig
```

You'll see two entries, one for eth0 and one for lo (local). In my example, the automatically assigned address is 10.128.1.26. If your administration PC is a Linux box or Mac, open a terminal and `ssh` in to your new VM by typing:

```
$ ssh username@10.128.1.26
```

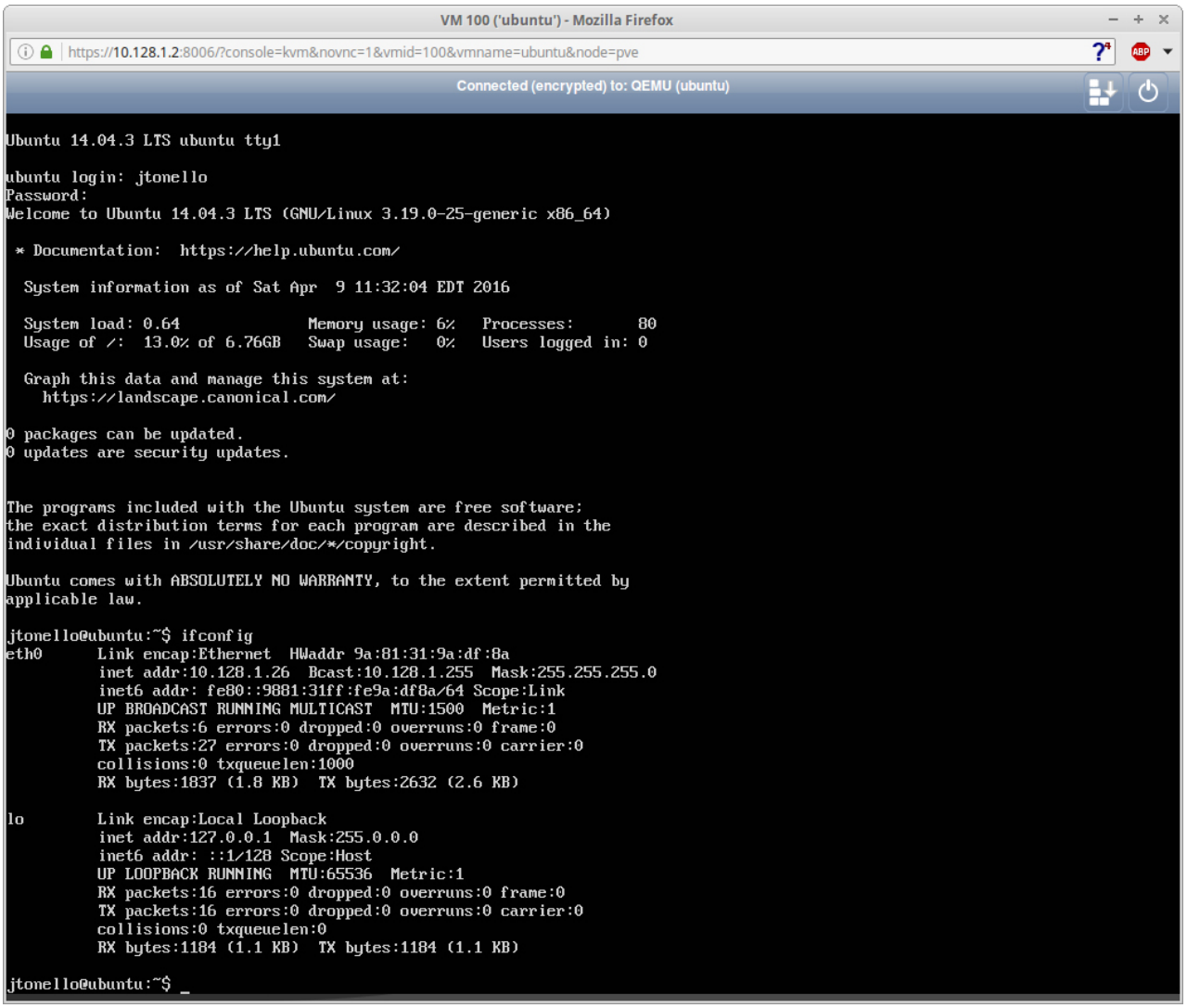


Figure 7. Logging in and `ifconfig` Output

If you're on Windows, `ssh` in using PuTTY or a similar tool. If you get a login prompt, you're good to go. You always can use the Proxmox console to connect to your VMs, but being able to `ssh` in directly is handy.

Customize the VM

With your Ubuntu VM up and accessible, it's time to make some customizations that will save time for all future deployments. Start by adding any other administrative users you want. That way, when you make a template out of this VM, all those users already will be set up. You're already an administrative (sudo) user yourself, but it might be

handy to have someone else with admin rights:

```
$ sudo adduser msmith
```

Follow the prompts and enter the user's full name and any of the rest you care to add. Next, add your new user to the sudoers group:

```
$ sudo adduser msmith sudo
```

These steps can be combined, but I think it's useful to see the output so you better understand what's happening under the covers.

Set a Static IP Address

All your servers will have static IP addresses so they can be mapped to DNS later, so this is a good time to change them by editing the network configuration file:

```
$ sudo vi /etc/network/interfaces
```

Change the eth0 entry from this:

```
auto eth0
iface eth0 inet dhcp
```

to this:

```
auto eth0
iface eth0 inet static
    address 10.128.1.200   # .200 is out of range of anything
                           # I might be adding soon
    netmask 255.255.255.0
    dns-nameservers 10.128.1.3
    dns-search tiny.lab
```

Save the file and reboot.

Add a Second Network Interface

The goal of your tiny internet is to make a self-contained system that doesn't rely on the public internet, but you'll need to cheat a little with this first VM and, later, with the mirror server. These VMs need a second NIC so they can access both your private and public networks simultaneously.

Once this VM is customized and updated, you can delete the second NIC by reversing the steps you're about to take.

In the main Proxmox web interface, click on the Hardware tab for your VM. Click Add and select Network Device from the menu. In the modal window, select "Bridge mode" and bridge the second network interface, in my example "vmbr1". Click OK, and for good measure, restart the VM.

When you type `ifconfig` at the prompt after you log in, you'll still see only eth0 and lo, so you need to activate the newly added NIC. In the console window (or shell), edit the network

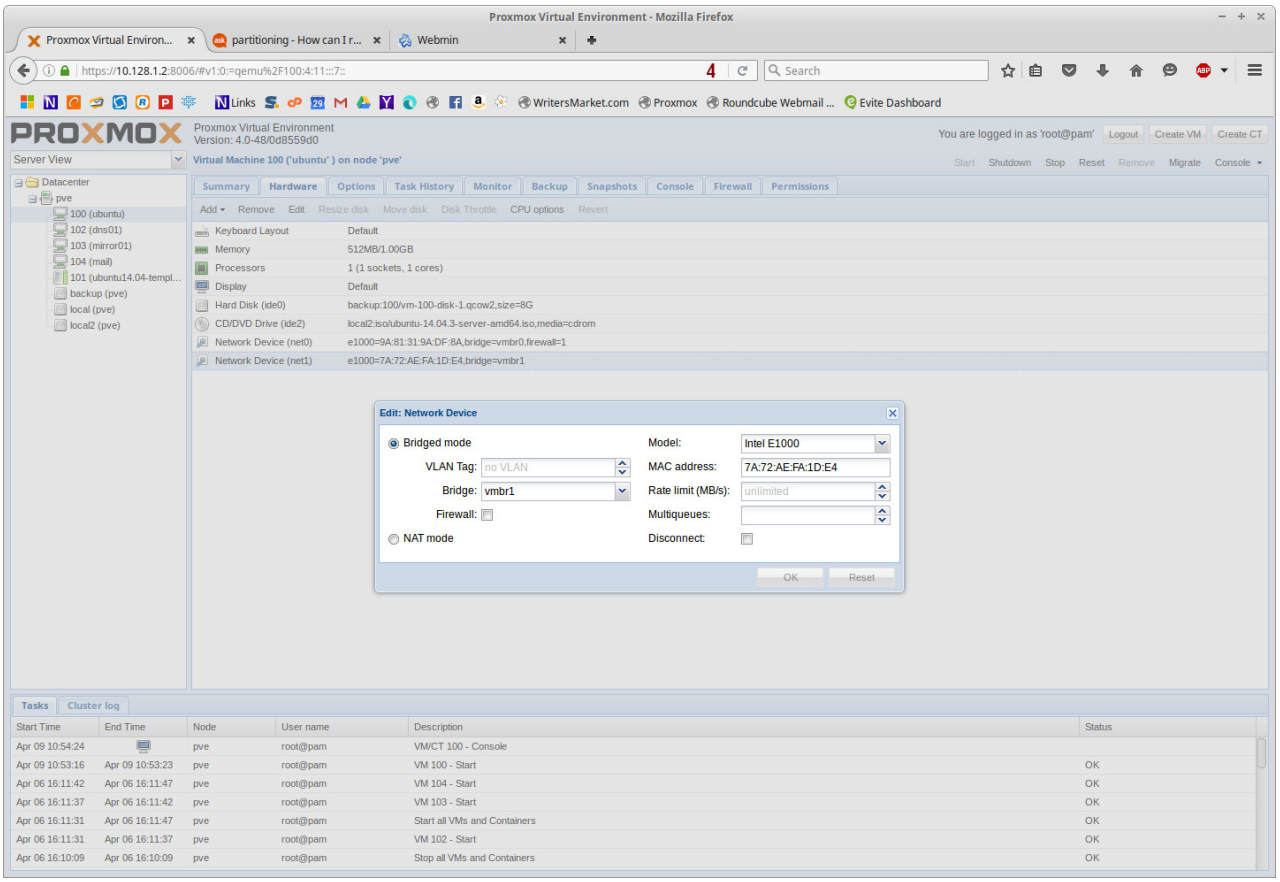


Figure 8. Bridge Mode

configuration file:

```
$ sudo vi /etc/network/interfaces
```

Create a static IP entry for the second NIC:

```
auto eth1
iface eth1 inet static
    address 192.168.1.200    # .200 is an available address on
                             # my network
    netmask 255.255.255.0
    gateway 192.168.1.1
    dns-nameservers 8.8.8.8 8.8.4.4
```

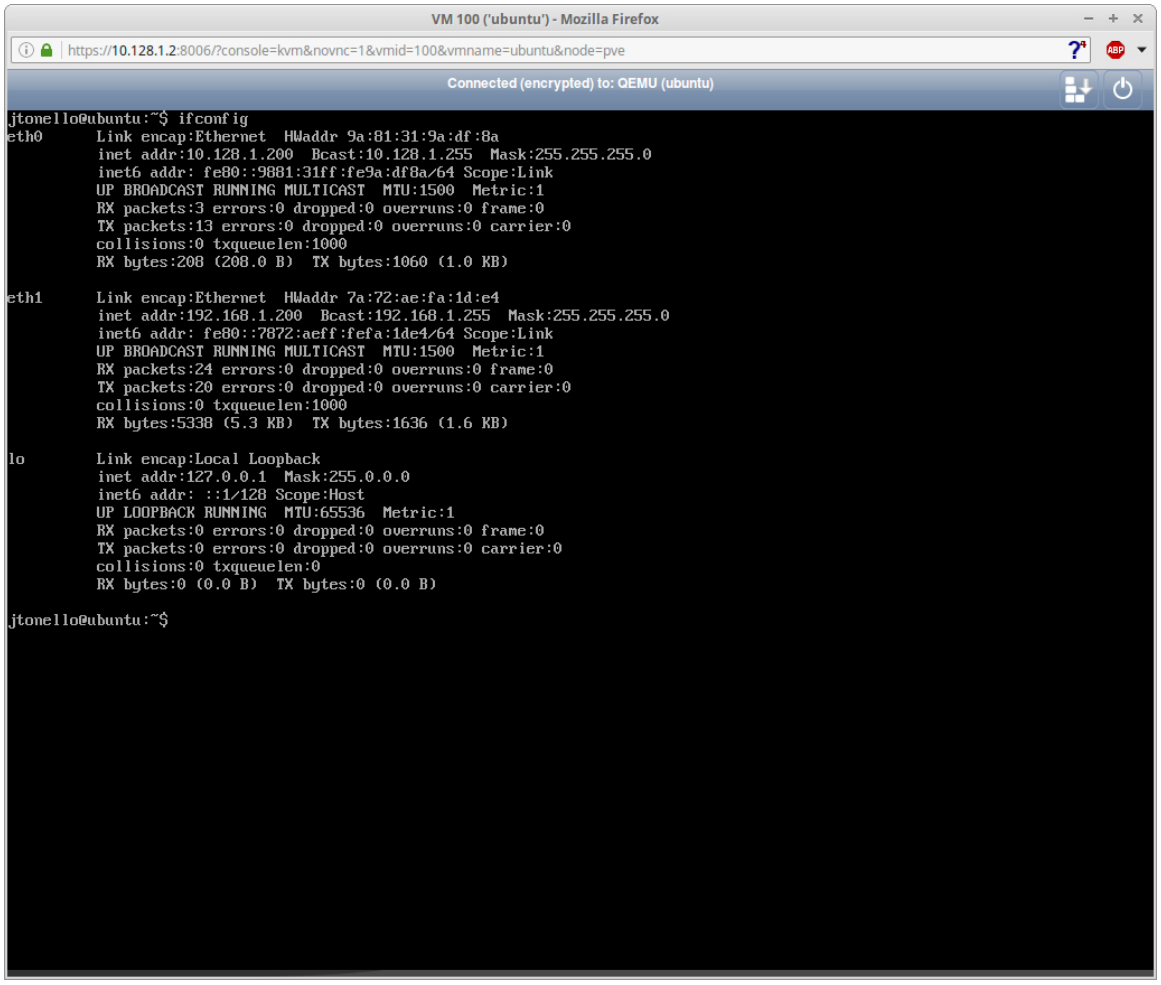


Figure 9. eth0 and eth1 are both active.

In this example, I choose .200 because it's a free address on my LAN. Assign one of your own that doesn't conflict with other machines on your network. The dns-nameservers are Google's.

Save the file and reboot the VM. When you log in and run `ifconfig` now, you should see that the eth0 and eth1 interfaces are active (Figure 9).

This VM now has access to your private network and the public internet so you can do updates and downloads. Run this update so you have the latest version of your software and kernel:

```
$ sudo apt-get update && sudo apt-get upgrade -y
 ➥&& sudo apt-get autoremove -y
```

Install Webmin

You're now ready to install Webmin, a browser-based tool for administering your Linux server. It's worth installing on the template VM so it's available on every server you build from here on out.

First, install a few packages Webmin needs to work properly:

```
$ sudo apt-get install libnet-ssleay-perl libauthen-pam-perl
 ➥libio-pty-perl apt-show-versions libapt-pkg-perl
```

When those packages are installed, copy the URL of the latest Debian installer from the Webmin downloads page (see the Resources section), and fetch it to your home directory on this VM:

```
$ cd ~
$ wget http://prdownloads.sourceforge.net/webadmin/
➥webmin_1.791_all.deb
```

Now, install it:

```
$ sudo dpkg -i webmin_1.791_all.deb
```

The installer will take a little while, so be patient. When it's done, it'll direct you to log in at https://ubuntu:10000, but since you don't have DNS set up yet, you'll have to use https://10.128.1.200:10000

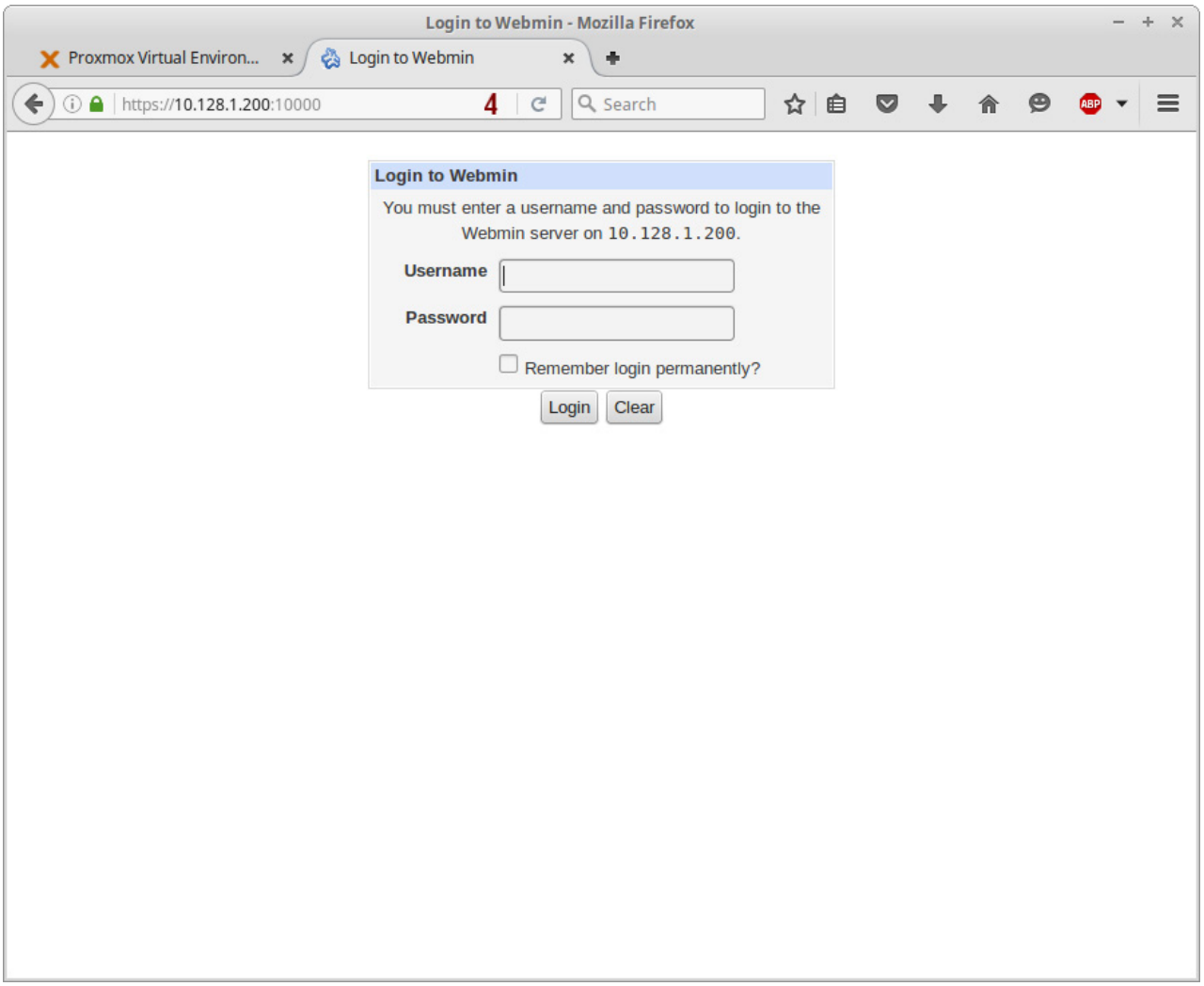


Figure 10. Logging in to Webmin

for now. Use your administration PC and log in with your user name and password.

Set Up the Firewall

If you want to enable a firewall on your template, this is the time to do it. It's easy with the Webmin interface—under Networking, click Linux Firewall. Check the box at the bottom that says "Enable firewall at boot time", and click the "Setup Firewall" button. Create four basic rules:

- Accept if state of connection is ESTABLISHED,RELATED.
- Accept if interface is not eth0.

- Accept if source is 10.128.1.0/24.
- Accept if source is 192.168.1.0/24.

These rules allow traffic only from devices on your two networks—the one you use to connect to the internet and the private one that makes up your tiny internet. Set the "Default Action To:" button to "Drop" on the first entry ("Accept" on the other two), and click Apply Configuration.

You can confirm these rules are active by running this simple command:

```
$ sudo iptables -L
```

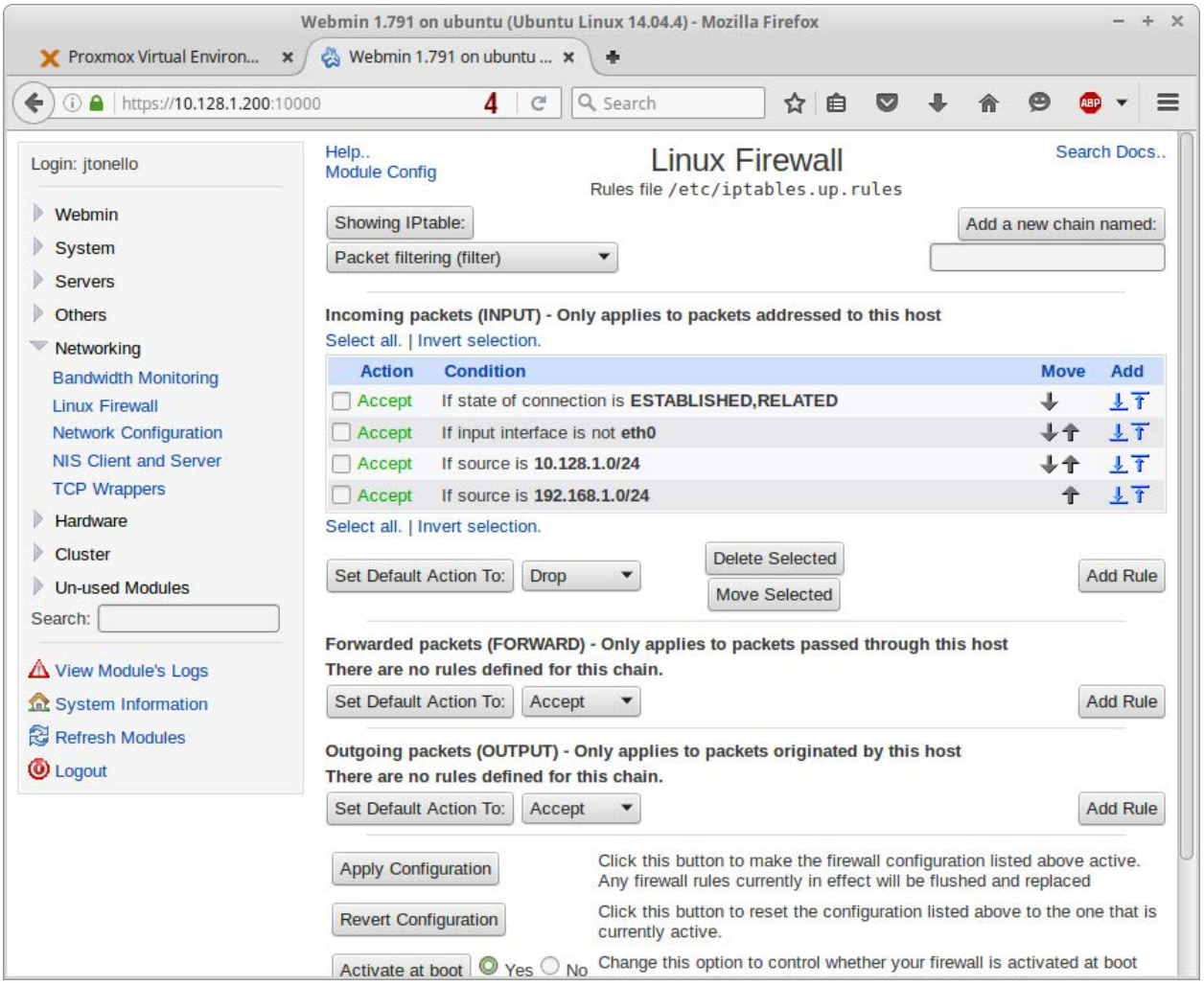


Figure 11. Setting Up the Firewall Rules

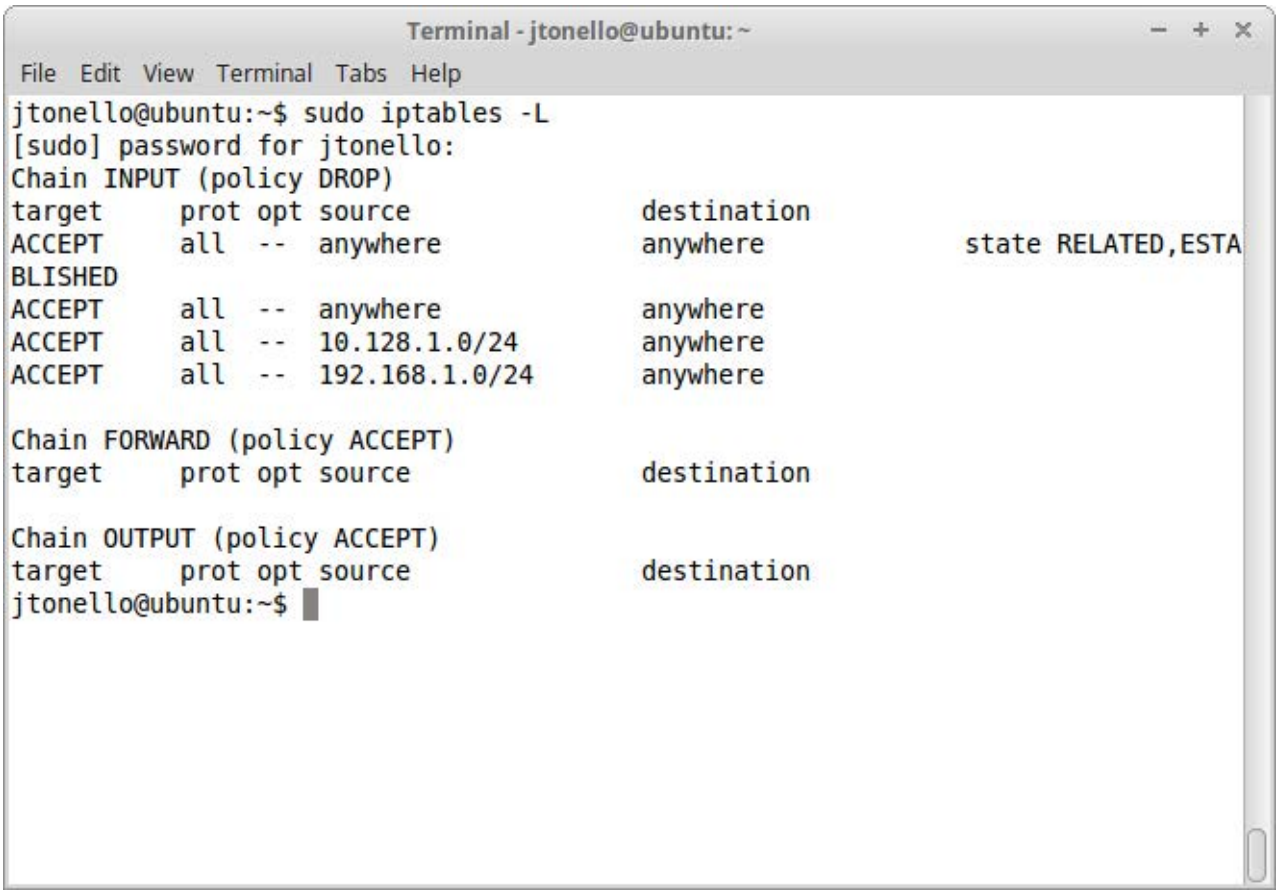


Figure 12. Output of the `sudo iptables -L` Command

Change Your Local sources.list

Next, you'll change the apt package repository listed in /etc/apt/sources.list from the Ubuntu default to your own. This will enable you to update all your VMs locally without them ever needing to access the public internet. Note that this won't work until your apt-mirror is fully operational (if you don't want to set up a mirror, skip this step):

```
$ mv /etc/apt/sources.list /etc/apt/sources.list.bak
$ sudo vi /etc/apt/sources.list
```

Enter the following three lines in sources.list and save it. The URL points to the VM you'll make called "mirror" on the domain you'll create called "tiny.lab":

```
deb http://mirror.tiny.lab/ubuntu trusty main restricted
 ➥universe multiverse
deb http://mirror.tiny.lab/ubuntu trusty-security main
 ➥restricted universe multiverse
deb http://mirror.tiny.lab/ubuntu trusty-updates main
 ➥restricted universe multiverse
```

Set Up a Proxy

If you're planning to build and use an HTTP proxy, edit /etc/environment to add the following lines. In this example, I used addresses and a port number created later by installing tinyproxy. After the PATH line, add:

```
no_proxy="127.0.0.1, localhost, *tiny.lab"
http_proxy="http://proxy.tiny.lab:8888"
ftp_proxy="http://proxy.tiny.lab:8888"
```

In this case, I don't want the system to use the proxy for anything on my private tiny internet domain (*tiny.lab), but it can for anything else that isn't local. If you're not planning to build your own mirror and plan to use a public repository (the default), you'll also need to edit /etc/apt/apt.conf to add a line telling apt to use your proxy to get to the repository:

```
Acquire::http::Proxy "http://proxy.tiny.lab:8888";
```

Convert Your VM to a Template

Now you're ready to convert this VM to a template. It's been customized with your credentials, static IP addresses that can be modified easily, Webmin for easy system management, simple firewall rules, a custom sources.list and proxy settings, if necessary.

To convert it, return to the Proxmox browser interface and shut down the machine. Right-click on the VM and select "Convert to template" from the menu.

After a few moments, the VM's icon will change, showing you that the machine is now purely a template. It no longer can be started as is.

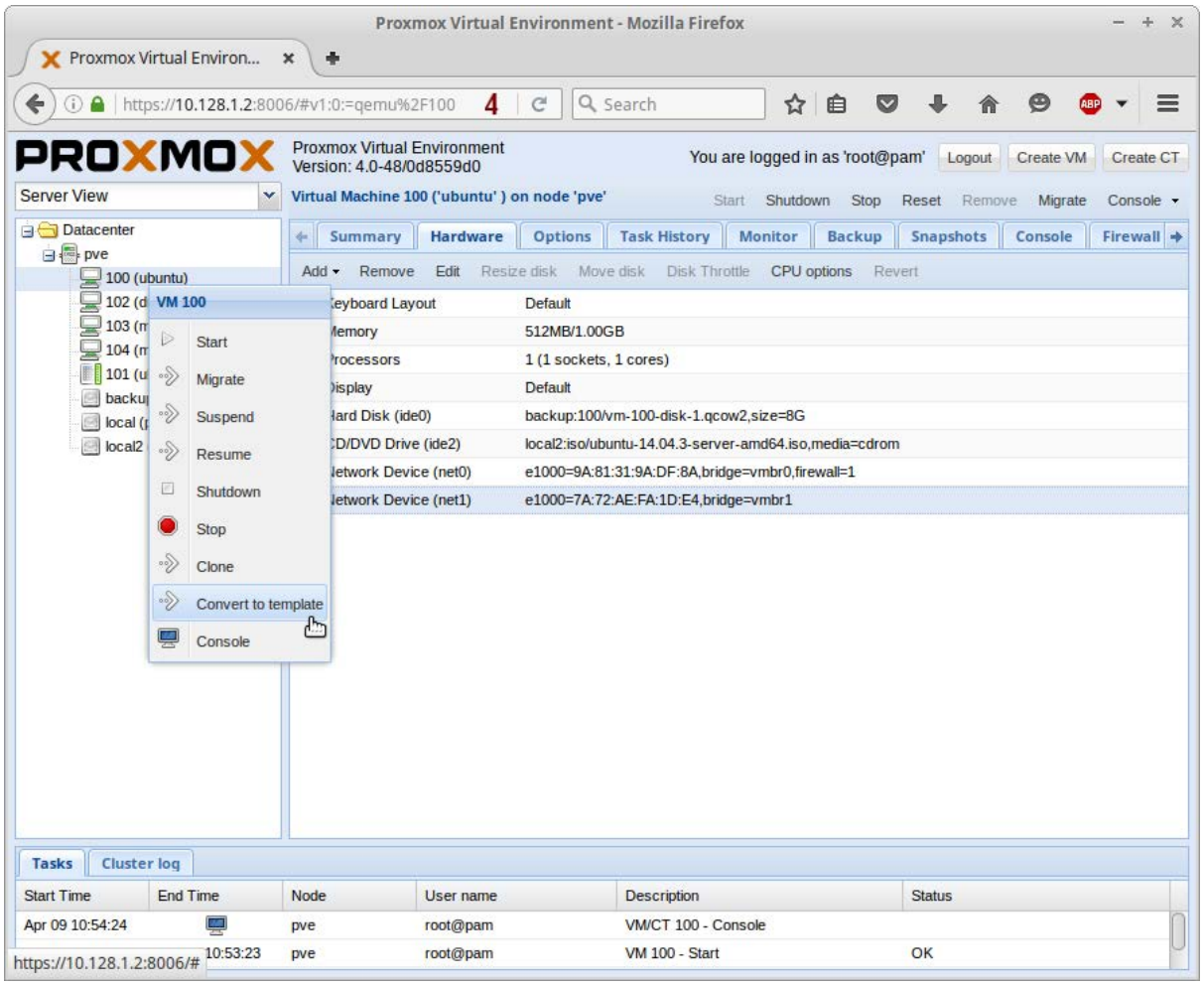


Figure 13. Converting the VM to a Template

If you installed a second hard drive on your Proxmox host server, now is a good time to back up this new template. Check the Proxmox website for more information.

Deploy a Linux Repository Mirror

You have two choices when it comes to deploying the Ubuntu repository: clone your new VM template and resize its 30GB virtual disk, or create an all-new VM with a larger disk. I explain the latter here.

Start by creating a new VM from the Ubuntu .iso file you used to create your template VM. This time, give the machine a virtual disk that's at least 200GB. Boot the machine and step through the server installation process as before, making sure to give the machine the hostname "mirror" and an IP address of 10.128.1.6. Use the same user name and password you set

on your template, and install only OpenSSH from the application list.

Follow the procedure for adding a second NIC, and be sure to leave the /etc/apt/sources.list unchanged for now. You'll need it to point to the default external Ubuntu repository.

Install apt-mirror

With the machine set with access to both your tiny internet and the public internet, install `apt-mirror` and the web server that will serve up packages:

```
$ sudo apt-get install apt-mirror apache2
```

The core of the `apt-mirror` configuration is the list of repository URLs. Make a backup of the default file and edit the original:

```
$ sudo cp /etc/apt/mirror.list /etc/apt/mirror.list.bak
$ sudo vi /etc/apt/mirror.list
```

Leave the `config` section at the top as is, and don't change the first three `deb` listings. However, to save disk space, remove all the `deb-src` entries and replace them with the 32-bit repositories. Your entries should look like this:

```
deb http://archive.ubuntu.com/ubuntu trusty main restricted
 ➥universe multiverse
deb http://archive.ubuntu.com/ubuntu trusty-security main
 ➥restricted universe multiverse
deb http://archive.ubuntu.com/ubuntu trusty-updates main
 ➥restricted universe multiverse

deb-i386 http://archive.ubuntu.com/ubuntu trusty main
 ➥restricted universe multiverse
deb-i386 http://archive.ubuntu.com/ubuntu trusty-security main
 ➥restricted universe multiverse
deb-i386 http://archive.ubuntu.com/ubuntu trusty-updates main
 ➥restricted universe multiverse

clean http://archive.ubuntu.com/ubuntu
```

Save the file and start `apt-mirror` with your edited mirror.list. Depending on your internet connection, this likely will take many hours. You'll get a hint of just how long when it tells you how many gigabytes will be downloaded. Be patient:

```
$ sudo apt-mirror /etc/apt/mirror.list
```

Continue your work during this download by opening another shell into the server. You can configure apache2 while `apt-mirror` is doing its thing by creating a symbolic link from the `apt-mirror` repository to the web directory that was created automatically when you installed apache2.

If you haven't already, test to see that the web server is up by pointing your browser to http://10.128.1.6.

If it works, create a new directory at the root of the web server:

```
$ sudo mkdir /var/www/html/ubuntu
```

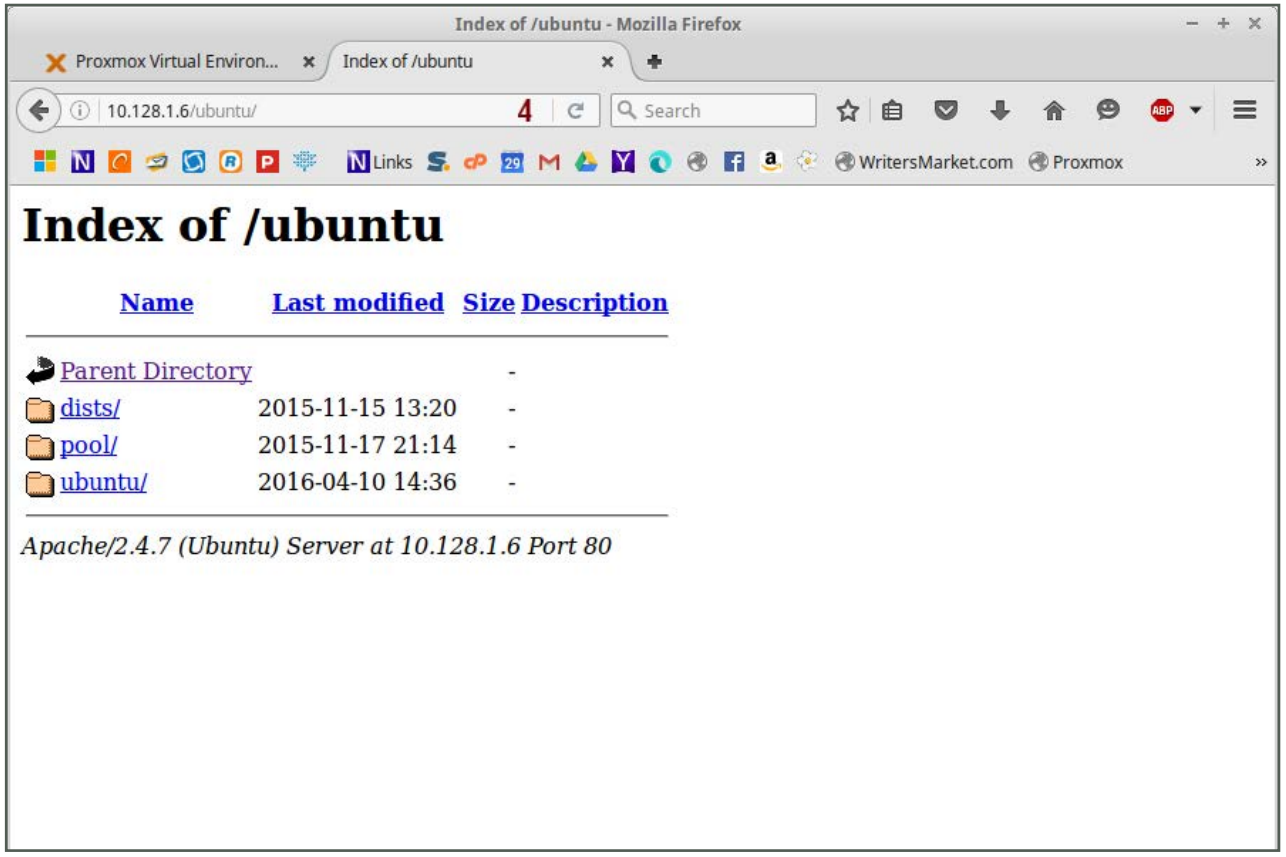


Figure 14. Installing apt-mirror

If you want to use domain names instead of IP addresses for reaching all your tiny internet machines, it's time to deploy a DNS server.

Create a symbolic link from the directory where `apt-mirror` stores the packages to the new directory:

```
$ sudo ln -s /var/spool/apt-mirror/mirror/archive.ubuntu.com/
➥ubuntu/pool/ /var/www/html/ubuntu
```

Test it by pointing your browser to http://10.128.1.6/ubuntu. If you see a list of directories, you're all set. Any VMs built with the local sources.list will be able to download packages from this local server once apt-mirror completes. (See the Resources section for more information.)

Set Up DNS (bind9)

If you want to use domain names instead of IP addresses for reaching all your tiny internet machines, it's time to deploy a DNS server. Use the addresses you established as part of your tiny internet schema. Here's a reminder:

- pve — 10.128.1.2
- dns01 — 10.128.1.3
- dns02 — 10.128.1.4
- mail — 10.128.1.5
- mirror — 10.128.1.6
- web01 –10.128.1.7

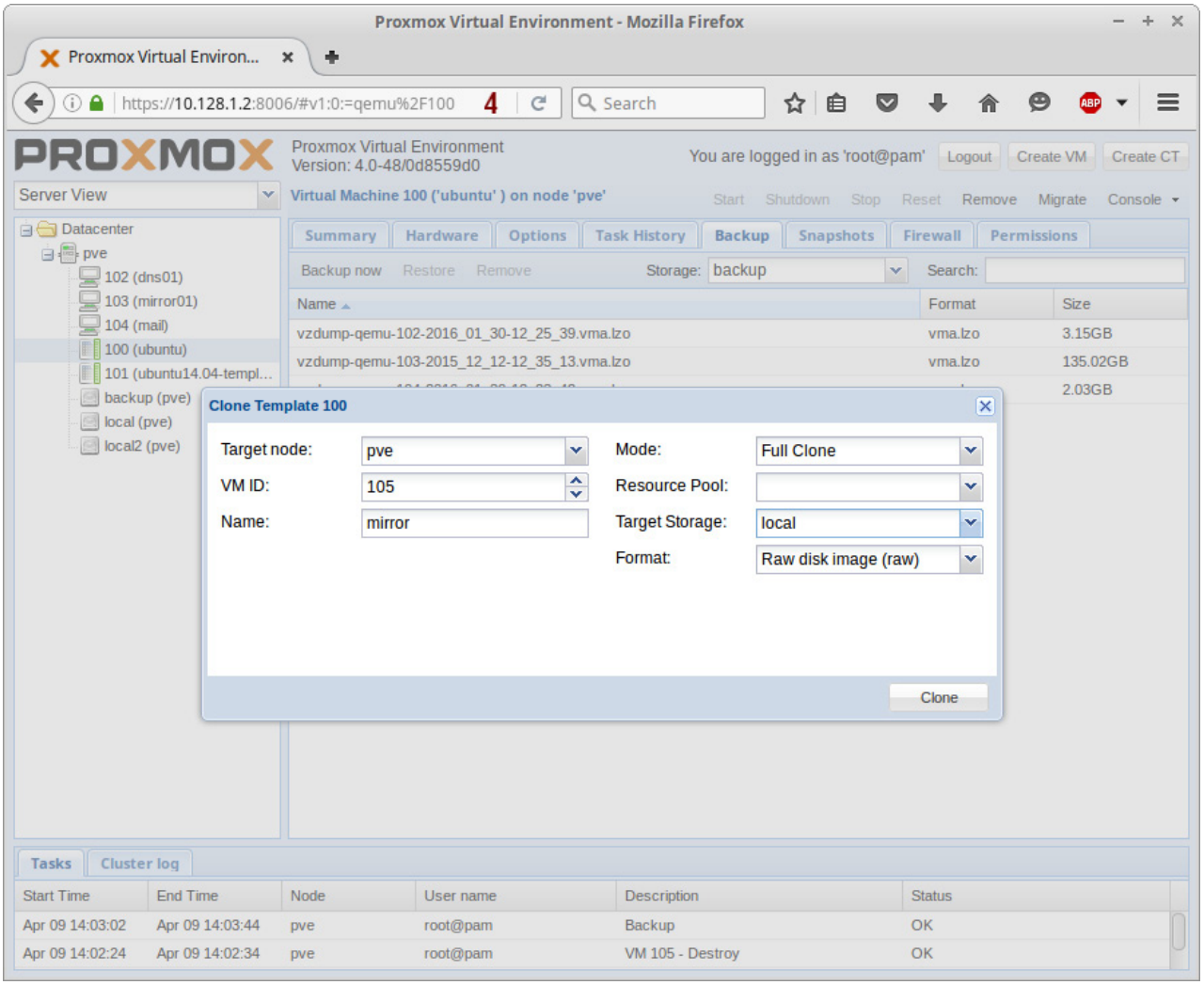


Figure 15. Cloning the Template

Start by cloning your VM template. Right-click on the template and select "Clone". The target node will default to "pve" (or whatever you called your Proxmox host). Set the VM ID and name to whatever you want. Leave the auto-incrementing ID as is, and give the VM a name that's the same as the hostname you'll assign. In my example, I used "dns01". Set the Mode to "Full Clone", and set the Target Storage to "local" with "Raw disk image" as the format.

It takes less than a minute to spawn a new VM from your template. In its current state, it's exactly like the original "ubuntu" VM—same IP address, same hostname. Of course, you'll need to change those before putting the machine into production:

- Edit /etc/hosts — change `127.0.1.1 ubuntu` to `10.128.1.3 dns01.tiny.lab dns01`.
- Edit /etc/hostname — change `ubuntu` to `dns01.tiny.lab`.
- Edit /etc/network/interfaces — change the `10.128.1.200` address to `10.128.1.3`.

Once you've made these basic changes, reboot, log in and install `bind9`:

```
$ sudo apt-get update
$ sudo apt-get install bind9 bind9utils dnsutils bind9-doc
```

The main DNS configuration is done in these three files:

- /etc/default/bind9
- /etc/bind/named.conf.options
- /etc/bind/named.conf.local

You'll finish by creating your zone files in /etc/bind/zones. I'm using the "tiny.lab" domain name in all these examples, but you can set the name to anything you want.

Start by adding an IPv4 option:

```
$ sudo vi /etc/default/bind9
```

Add the following to the end of the file:

```
OPTIONS="-4 -u bind"
```

Make backup copies of the next two files before editing them, then edit /etc/bind/named.conf.options and add your trusted hosts (one for each server and resource you have), and set the options:

```
acl "trusted" {
    10.128.1.1;
    10.128.1.2;
    10.128.1.3;
    10.128.1.4;
    10.128.1.5;
    10.128.1.6;
    10.128.1.7;
    10.128.0.0/16;
};

options {
    directory "/var/cache/bind";

    recursion   yes;   # enables recursive queries
    allow-recursion  { trusted; };   # allows queries from "trusted"
                                        # clients
    listen-on { 10.128.1.3; };   # dns01 IP address
    allow-transfer { none; };   # disable zone transfer by default

    forwarders {
        8.8.8.8;   # These are Google's DNS servers
        8.8.4.4;
    };

    ...
};
```

Save the file and create your zones by editing /etc/bind/named.conf.local. This is where you set your domain name, replacing "tiny.lab" with whatever you want:

```
zone "tiny.lab" {
    type master;
    file "/etc/bind/zones/db.tiny.lab";
    allow-transfer { 10.128.1.4; };  # Setting this for a
```

```
                                    # future secondary DNS server
};

zone "128.10.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/db.10.128";
    allow-transfer { 10.128.1.4; };
};
```

Now create the forward and reverse zone files, placing them in the /etc/bind/zones folder. If it doesn't exist, create it:

```
$ cd /etc/bind
$ sudo mkdir zones
```

Copy the default DNS forward and reverse zone config files into that folder, renaming them to match your domain name and IP subnet:

```
$ sudo cp db.local ./zones/db.tiny.lab
$ sudo cp db.127 ./zones/db.10.128
```

Edit /etc/bind/zones/db.tiny.lab, and enter your current and future hosts. The file I set up includes comments at the top to remind me of changes I make. I also created entries for my router (10.128.1.1) and a proxy server, which I put on the same box as my dns02. Each time you make modifications, increment the Serial entry before saving. Also note the ".?" after each name. Don't leave those off. You can find more information about DNS in the Resources section at the end of this article, but this will get you started:

```
; BIND data file for local loopback interface
;
;       20150505        JST     Modified proxy address
;       20160505        JST     Added web01

$TTL    604800
```

```
@       IN      SOA     dns01.tiny.lab admin.dns01.tiny.lab. (
                             12         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
; name servers -- NS records
        IN      NS      dns01.tiny.lab.
        IN      NS      dns02.tiny.lab.

; name servers -- A records
dhcp.tiny.lab.          IN      A       10.128.1.1
pve.tiny.lab.           IN      A       10.128.1.2
dns01.tiny.lab.         IN      A       10.128.1.3
dns02.tiny.lab.         IN      A       10.128.1.4
proxy.tiny.lab.         IN      CNAME   dns02.tiny.lab.
mail.tiny.lab.          IN      A       10.128.1.5
mirror.tiny.lab.        IN      A       10.128.1.6
web01.tiny.lab.         IN      A       10.128.1.7
```

Save the file and edit the /etc/bind/db.10.128 reverse zone file. The IP addresses for each server under "PTR records" are truncated-looking and can be confusing. Imagine each leading off with an invisible "10.128." to envision the addresses. Again, be sure to increment the Serial entry each time you make a change:

```
; BIND reverse data file for local loopback interface
;
; 20160505      JST     Added cname for proxy
; 20160505      JST     Added mirror01

$TTL    604800
@       IN      SOA     tiny.lab. admin.tiny.org. (
                             11         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
```

```
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
; name servers -- NS records
        IN      NS      dns01.tiny.lab.
        IN      NS      dns02.tiny.lab.

; PTR records
1.1     IN      PTR     dhcp.tiny.lab.
1.2     IN      PTR     pve.tiny.lab.
1.3     IN      PTR     dns01.tiny.lab.
1.4     IN      PTR     dns02.tiny.lab.
1.5     IN      PTR     mail.tiny.lab.
1.6     IN      PTR     mirror.tiny.lab.
1.7     IN      PTR     web01.tiny.lab.
```

Save the file and check the syntax of your files by running:

```
$ sudo named-checkconf
```

If everything is correct, you'll get no output and no errors. Check the configurations further with `named-checkzone`:

```
$ sudo named-checkzone tiny.lab /etc/bind/zones/db.tiny.lab
$ sudo named-checkzone 128.10.in-addr.arpa /etc/bind/zones/db.10.128
```

You'll see "OK" if everything checks out. If not, edit the files. Leaving off the trailing ".?" is a common mistake.

Restart bind to get it up and running:

```
$ sudo service bind9 restart
```

When `bind9` restarts, do a quick check with the `dig` utility, or simply open a browser and navigate to your mirror server at http://mirror.tiny.lab:

```
$ dig mirror.tiny.lab
```

If you see 10.128.1.6 in the `dig` output, you've succeeded. DNS is working. You can complete your DNS setup by deploying a second VM or installing `bind9` on a physically separate machine on your tiny internet network, but it's not strictly necessary at this point. You also can set this VM to start automatically when your Proxmox host starts, so you have DNS running whenever your tiny internet is up.

Deploy a Mail Server

Mail servers have two key components: a service that transfers mail and a service that serves up mail. Postfix is a common transfer agent (SMTP), and when coupled with Dovecot, it will provide you with all you need to send and receive mail via IMAP (or POP if you're so inclined).

Deploy another clone of your "ubuntu" template, this time naming it "mail", making sure the Mode is "Full Clone" and your Target Storage is "local". Once it's generated, start it up, open a console, and update the same basic information you did for your mirror server: set the static IP address for `eth0` as 10.128.1.5, and change the `hostname` to `mail.tiny.lab`.

Reboot and install Postfix and Dovecot:

```
$ sudo apt-get update
$ sudo apt-get install postfix
```

During the install, you'll be prompted to select the type of installation you want. Choose "Internet Site". Follow that by setting the "System mail name" as "mail"—the same as the hostname.

When it's done, install `mailutils` and Dovecot tools:

```
$ sudo apt-get install dovecot-imapd dovecot-pop3d
```

Reply "Yes" to install the self-signed certificate, set the hostname as "mail" and allow the install to complete.

To configure Postfix, run this command:

```
$ sudo dpkg-reconfigure postfix
```

You can confirm the entries you made during the Postfix install, and then proceed to set your user name for the "Root and postmaster mail recipient"—in my case, "jtonello". Look over the other destinations from which to accept mail, and add your domain ("tiny.lab"). Don't force synchronous updates on the mail queue, but under the local networks, be sure to add "10.128.1.0/24" to the list. That's the scope you defined for all your machines. If you leave this out, the mail server will reject all incoming mail.

Set the mailbox limits to suit your needs, and set "all" as the internet protocols to use. You'll start with IPv4, but having IPv6 enabled offers future flexibility.

With the basics now in place, check out the main Ubuntu Postfix and Dovecot pages listed in the Resources section for more information.

You'll be able to use any email client with your new mail server,

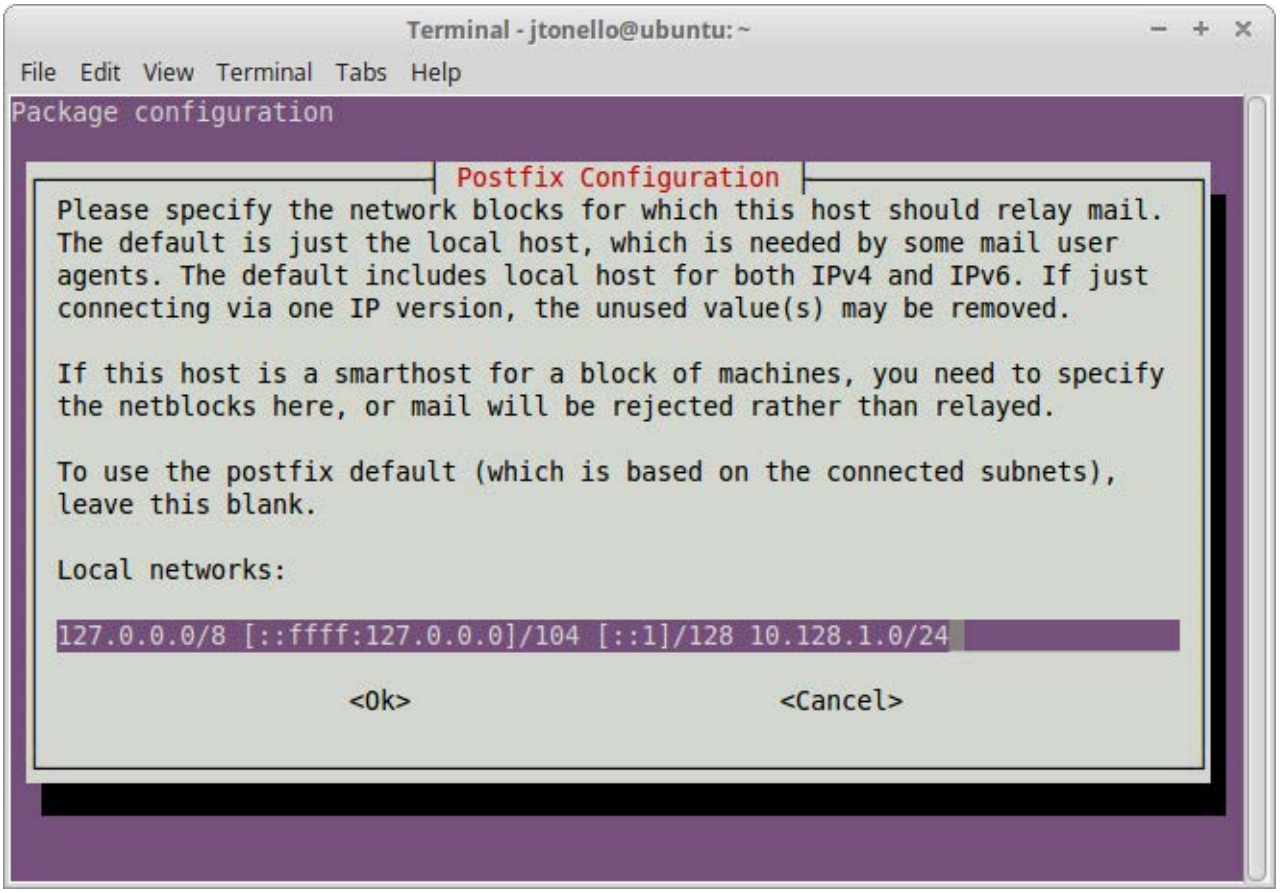


Figure 16. Postfix Configuration

You'll be able to use any email client with your new mail server, including Thunderbird and Evolution, two common tools that come pre-installed on many Linux distributions.

including Thunderbird and Evolution, two common tools that come pre-installed on many Linux distributions. You also might consider installing a web-based mail tool like Roundcube. It provides a great interface and plenty of features that work well with Postfix and Dovecot.

If you want to dig in to the mail configuration, use Webmin. Log in at https://mail.tiny.lab:10000 and look under the Servers tab for the Postfix and Dovecot entries. From there, you can explore and manage the servers easily.

Set Up a Web Server

So far, without possibly being aware, you've installed apache2 on the mirror and mail servers. These allow web connections to those services, but you'll want a more full-featured LAMP stack (Linux, Apache, MySQL and PHP) for building robust websites.

When you first installed Ubuntu from the .iso, you may have noticed a LAMP choice during the applications install step. You definitely can create a VM from scratch and check that box; it'll give you everything you need. However, since you have a nice pre-made VM template, you can use that and add LAMP to it.

Start by cloning your Ubuntu template. Follow the same steps you did with previous clones—change the IP address (10.128.1.7) and hostname (web01). Reboot and open a terminal to install the LAMP components:

```
$ sudo apt-get install lamp-server^
```

The caret (^) is important; don't leave it off. As the installers proceed, you'll be asked to create a root password for the MySQL database. Jot it down or put it in your password safe so you don't forget it.

The installation will create the /var/www/html folder and the default index.html. If you point your browser to the machine (either by IP or DNS name if you set it up), you'll see the default Apache2 page. Test the PHP installation by creating a new file in the same directory:

```
$ sudo vi /var/www/html/phpinfo.php
```

Add these lines to the file:

```
<?php
        phpinfo();
?>
```

Point your browser to the page at http://web01.tiny.lab/phpinfo.php. If you see a page with information, your server is successfully serving up PHP.

To make it easier to work with the MySQL database, install phpMyAdmin:

```
$ sudo apt-get install phpmyadmin
```

Select "apache2" as the server to reconfigure automatically, and answer "No" to the next question (because the database is already configured). When the installation is complete, you'll have a very robust web-based tool to manage all your databases. Log in at http://web01.tiny.lab/phpmyadmin with the MySQL user name "root" and the password you set.

If you plan to deploy multiple web servers, go ahead and convert this web01 VM to a template. That way, it will be full-featured and ready with only a few small changes. If you want each VM to use WordPress or another content-management tool, install that before you make your template.

Conclusion

You now have a fully operational tiny internet, complete with a Linux repository; mail, DNS and web servers; and a number of useful tools. Use this setup to explore and learn about Linux. If you mess up a VM server,

just deploy another one from your template. Most important, share what you've learned and get others involved, and help spur the curiosity of the next generation of Linux enthusiasts.■

John Tonello is the Director of IT for NYSERNet Inc., New York state's regional optical networking company. He's been a Linux user and enthusiast since building his first Slackware system from diskette 20 years ago. Since then, he's developed web and IT solutions for major universities, Fortune 500 companies and small start-ups. A former Cornell University IT trainer and writer, John served six years as the mayor of an Upstate New York city, where he championed the use of technology to help solve problems facing municipalities.

Resources

Download PuTTY: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Webmin Downloads: http://webmin.com/download.html
and http://prdownloads.sourceforge.net/webadmin/webmin_1.791_all.deb

Setting Up a Mirror: https://www.howtoforge.com/local_debian_ubuntu_mirror_p2

Setting Up DNS (bind9): https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-14-04
and https://help.ubuntu.com/community/BIND9ServerHowto

Postfix and Dovecot Installation: https://help.ubuntu.com/community/Postfix
and https://help.ubuntu.com/community/Dovecot

Installing Roundcube: https://github.com/roundcube/roundcubemail/wiki/Installation

Install a LAMP Stack: https://help.ubuntu.com/community/ApacheMySQLPHP

TinyProxy How-To: https://tinyproxy.github.io

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

RETURN TO CONTENTS

Coroutines and Channels in C
Using libmill

Want to try a different approach to writing concurrent applications in C? This article looks at libmill, a library that brings Go-style concurrency to C.

AMIT SAHA

PREVIOUS
Feature:
The Tiny Internet Project, Part III

NEXT
Doc Searls' EOF

libmill is a C library that brings Golang-style concurrency to C. Using it, you can call a function like `f(arg1, arg2)` using `go(f(arg1, arg2))`, and the function will be executed in a separate coroutine. If you have multiple coroutines executing, libmill's scheduler takes care of scheduling the coroutines. Data will be passed to and from coroutines using channels. In this article, I introduce the key libmill features—creating coroutines and using channels to pass data to and from them. In addition, I take a look at some of libmill's other convenience functions that make it more than a coroutine and channels library.

Installing libmill

libmill currently is considered stable. Its latest release at the time of this writing is version 1.8 (released in March 2016). To install it, download the gzipped tarball from http://libmill.org/download.html, extract it and do the following steps:

```
$ ./configure
$ make
$ sudo make install
```

Note that you likely will need to run `sudo ldconfig` on Ubuntu/Debian, and manually add /usr/local/lib to a file in /etc/ld.so.conf.d/, and then run `sudo ldconfig` on Fedora to be able to link to the libmill shared library when compiling your programs.

Hello Coroutine

Now that you've installed libmill, let's write a first example of using coroutines in programs (Listing 1a).

Listing 1a. listing1.c

```
# include <stdio.h>
# include <libmill.h>

coroutine void f(int index)
{
  printf("Worker %d\n", index);
}

int main(int argc, char **argv)
{
  for(int i=1;i<=10; i++) {
    go(f(i));
  }
  return 0;
}
```

Let's compile and run this program:

```
$ gcc -o listing1 listing1.c -lmill
$ ./listing1
Worker 1
Worker 2
Worker 3
Worker 4
Worker 5
Worker 6
Worker 7
Worker 8
Worker 9
Worker 10
```

The function, `f()` in Listing 1a is defined almost like an ordinary C function. The only difference is the `coroutine` specifier at the beginning:

```
coroutine void f(int index)
```

The `coroutine` specifier tells libmill that you plan to call this function in a coroutine using the `go()` construct, as you saw in the `main()` function in Listing 1a:

```
go(f(i));
```

Here, you call the function, `f()`, in a coroutine ten times, one after the other. The reason you see the above output in the same order (as opposed to any other order) as starting the coroutines is that the program runs in a single process. Given that though, if libmill finds that a coroutine implicitly or explicitly can be scheduled out and another coroutine is ready to run, the latter will start running.

libmill provides a function called `msleep()` that can be used to tell libmill's scheduler explicitly that it wants to sleep and let other runnable processes run. This function will allow you to see libmill's scheduling in action (Listing 1b).

Listing 1b. listing1-msleep.c Coroutines with msleep()

```
# include <stdio.h>
# include <libmill.h>
# include <string.h>
# include <stdlib.h>

coroutine void f(int index)
{
  msleep(now() + rand() % 50 );
  printf("Worker %d\n", index);
}

int main(int argc, char **argv)
{
  for(int i=1;i<=10; i++) {
    go(f(i));
  }
  msleep(now() + 60);
  return 0;
}
```

When you compile and run this code, you'll see output where the coroutines are no longer executed in the order they are started:

```
$ gcc -o listing1-msleep listing1-msleep.c -lmill
$ ./listing1-msleep
Worker 1
Worker 4
Worker 10
Worker 6
Worker 3
Worker 9
Worker 8
Worker 5
Worker 7
Worker 2
```

The key statement in Listing 1b is `msleep(now() + rand() % 50 );` in the function `f()`. When you use `go(f(i))` to start the function `f()` in a coroutine, it first goes to "sleep" using the `msleep()` function. This tells the libmill scheduler that it can schedule other coroutines in the meantime.

Unlike the standard library's `sleep()` function, the argument to the `msleep()` function is a "deadline"—a libmill concept that means you give a function a deadline in time after which libmill will continue its execution. The `now()` function returns the current time in milliseconds, so `msleep(now() + rand() % 50)` sets the deadline to a random time between 0 and 50 microseconds for each coroutine. You'll also want to wait for all the coroutines to finish in the `main()` function, so insert the statement `msleep(now() + 60)` in it.

One point to mention here is if you instead use the `sleep()` function, the entire thread of execution will block, and no scheduling will be possible among the coroutines. This is the reason libmill comes with its own set of non-blocking alternatives for input/output operations, such as `tcprev()`, `mfread()` and others.

Channels are both a messaging as well as a synchronization mechanism when working with coroutines.

Using Channels for Communication

Channels are both a messaging as well as a synchronization mechanism when working with coroutines. A channel is unidirectional and typed. When creating a channel, you have to specify the type of data it will carry. Using a channel, you can send data to a coroutine, and then in the sending coroutine, wait to receive data back on a different channel.

Next, let's look at an example of using a channel to send work to a coroutine, wait for it to process the work and send the result back over another channel. The work here is simple. You pass the first command-line argument to the program and send it to the coroutine, which returns the length of the string (Listing 2).

In the `main()` function, create two channels, one for input and another for output, and call them input and output, respectively:

```
chan input = chmake(char*, 0);
chan output = chmake(int, 0);
```

`chan` is a type defined by libmill to represent channels. A channel is created using the `chmake()` function. The first argument to `chmake()` is the type of data it will carry, and the second argument is its size. The default behaviour of channels is that a sender will block until there is a receiver and vice versa. When the size of the channel is 0, it is called an unbuffered channel, and it will not allow any data to be written in either of the scenarios. When the size of the channel is a non-zero number, it allows as many messages to be sent to it before blocking.

The two channels created above are unbuffered channels. This suits the purpose here since you want the coroutine to wait until you send it

Listing 2. listing2.c Example of Using Channels

```
# include <stdio.h>
# include <libmill.h>
# include <string.h>
# include <stdlib.h>

coroutine void worker(chan input, chan output) {
  /* Receive work*/
  char *work = chr(input, char*);
  printf("Processing: %s\n", work);
  /* Send the result back by writing to the channel*/
  chs(output, int, strlen(work));
}

int main(int argc, char **argv)
{
  if (argc != 2) {
    printf("Please specify one command line argument\n");
    exit(1);
  }
   /* Create an unbuffered channel to send work on*/
  chan input = chmake(char*, 0);
  if (input == NULL) {
    printf("Failed to create channels for input\n");
    exit(1);
  }

  /* Create an unbuffered channel to receive result on*/
  chan output = chmake(int, 0);
  if (output == NULL) {
    printf("Failed to create channels for output\n");
    exit(1);
  }

  /* Create a worker coroutine */
  go(worker(input, output));
  /* Send data to worker by writing to input channel*/
  chs(input, char*, argv[1]);
  /* Receive result from the coroutine and output*/
  printf("Result: %d\n", chr(output, int));

  /* Close channels */
  chclose(input);
  chclose(output);
  return 0;
}
```

some work to do on the input channel, and then you want to wait for the coroutine to send you the result back in the output channel.

The coroutine function, `worker()`, accepts the two channels as parameters: `chan input` and `chan output`. When it starts, you use the `chr(input, char*)` function (`chr` stands for "ch"annel "r"eceive) to read a `char*` data item from the input channel. Until there is a data item available to be read, `worker()` blocks. Once there is a data item to read, it creates a copy of it in work, then uses the `chs(output, int, strlen(work))` function (`chs` stands for "ch"annel "s"end) to write the length of the string to the output channel.

Let's compile and run the above program:

```
$ gcc -o listing2 listing2.c
$ ./listing2
Please specify one command line argument
$ ./listing2 hello
Processing: hello
Result: 5
```

In the main() function, you start the coroutine using `go(worker(input, output))`. Next, you write the string `argv[1]` to the input channel using the `chs()` function. Next, you use the `chr()` function to read the data on the output channel and print it. Note, how you don't have to worry about having to wait explicitly before you read from the output channel. It is all taken care of for you automatically. Finally, you close both the channels, which frees the resources:

```
chclose(input);
chclose(output);
```

Writing a Coroutine-Based TCP Server

libmill comes with a number of helper functions that are customized for writing coroutine-based network server programs. Next, I describe how to write a simple TCP server that handles each client in a separate coroutine (Listing 3).

Listing 3. listing3.c Simple Coroutine-Based TCP Server

```
# include <libmill.h>
# include <stdio.h>
# include <stdlib.h>
# include <errno.h>
# include <unistd.h>

/* Handler coroutine */
coroutine void handler(tcpsock as) {
  printf("New connection!\n");
  tcpclose(as);
}

int main(int argc, char **argv)
{
  int port = 9090;
  ipaddr addr = iplocal(NULL, port, 0);
  tcpsock server = tcplisten(addr, 10);

  if (!server) {
    perror("Can't setup a listening server\n");
    return 1;
  } else {
    printf("Server listening on %d\n", port);
  }

  /* Server loop*/
  while(1) {
    tcpsock as = tcpaccept(server, -1);
    if (!as)
      continue;
    /* Dispatch this request */
    go(handler(as));
  }

  return 0;
}
```

The function `handler()` handles a new client connection. It accepts a parameter of type `tcpsock` as an argument and will be used to represent a new client connection. It prints a string "New connection!" and closes the client connection using the `tcpclose()` function.

Let's now look at the `main()` function that creates the server. The following two statements create a listening socket:

```
ipaddr addr = iplocal(NULL, port, 0);
tcpsock server = tcplisten(addr, 10);
```

The `iplocal()` function is used to convert a human-friendly IP address to listen on to an address of the `ipaddr` type. The first argument to the function is the IP address or network interface to listen on; specifying `NULL` will make the server listen on all local network interfaces. The second argument to the function is the port to listen on, and the third function specifies whether you want an IPv4 or an IPv6 address. Leaving it as 0 defaults to IPv4 for now.

Next, you call the `tcplisten()` function to create the listening socket. The first argument to `tcplisten()` is the address you obtained in the previous step. The second argument is the backlog—the maximum number of incoming connections that have not yet been accepted, but will not be refused. It returns a value of type `tcpsock` representing the listening socket.

If for some reason the `tcplisten()` function cannot set up the listening socket, it returns `NULL` and sets `errno` to an error code specifying the reason it failed. Hence, you check for that and use the `perror()` function to report the error when there is one.

Once you have set up the listening socket correctly, you create an infinite loop that will accept an incoming connection using `tcpaccept()`. The first argument to the `tcpaccept()` function is `server`, the listening socket you created earlier. The second argument is a deadline for the function. The deadline is a time in the future up to which `tcpaccept()` will wait for a client connection to come in before returning. A value of -1 indicates

that `tcpaccept()` will block indefinitely for an incoming connection:

```
/* Server loop*/
 while(1) {
   tcpsock as = tcpaccept(server, -1);
   if (!as)
     continue;
   /* Dispatch this request */
   go(handler(as));
 }
```

`tcpaccept()` returns `NULL` upon error, so you check for it and start waiting for the next connection, if there was one. If the connection was accepted, you call the `handler()` function with the accepted connection socket in a coroutine.

Let's compile and run the program:

```
$ gcc -o listing3 listing3.c -lmill
./listing3
Server listening on 9090
```

In a different terminal session, you can try using telnet to 127.0.0.1 9090, and you will see "New connection!" being printed for every new client connection.

The repository for this article's code (https://github.com/amitsaha/lj_libmill) contains a more elaborate example of using libmill's functions to create an HTTP server in the directory named hashid_service. It handles each incoming connection in a new coroutine.

Choosing among Multiple Channels

When you have multiple channels and you want to perform an operation when a channel is ready for reading or writing, the `choose` construct allows you to do that. Listing 4 is an fictional example of when this might be useful. It downloads data from remote URLs specified as command-line arguments, making the request to each URL in a separate coroutine.

Listing 4. listing4.c Choosing Among Multiple Channels

```
# include <libmill.h>
# include <stdio.h>
# include <stdlib.h>
# include <time.h>

struct data {
  char *url;
  char *data;
};

coroutine void f(chan work, chan result, chan error) {
  struct data d;
  char *url = chr(work, char*);
  d.url = url;
  /* Simulate 70% success and 30% failure scenario*/
  double r = (double) rand()/RAND_MAX;
  if (r < 0.71) {
    // Successful scenario
    d.data = "Data at the URL";
    chs(result, struct data, d);
  } else {
    // Unsuccessful scenario
    d.data = "Error retrieving data";
    chs(error, struct data, d);
  }
}

int main(int argc, char **argv) {
  chan work = chmake(char*, 0);
  chan result = chmake(struct data, 0);
  chan error = chmake(struct data, 0);

  for(int i=1; i<argc; i++) {
    go(f(work, result, error));
    chs(work, char*, argv[i]);
  }

  for(int i=1; i<argc; i++) {
    choose {
    in(result, struct data, value):
      printf("Processed URL: %s, Result: %s\n", value.url, value.data);
    in(error, struct data, value):
      printf("Processed URL: %s, Error: %s\n", value.url, value.data);
    end
    }
  }
  // Close all channels
  chclose(work);
  chclose(result);
  chclose(error);
  return 0;
}
```

This program uses three channels: `work`, `result` and `error`. The `work` channel is used to send work from the main program to the coroutine, and the coroutine uses the `result` and `error` channels to send back the result of a successful request and error, respectively. The channels are created in the `main()` function with the following code:

```
chan work = chmake(char*, 0);
chan result = chmake(struct data, 0);
chan error = chmake(struct data, 0);
```

The first channel, `work`, is of type `char*`, and the second and third channels are of type `struct data`, which is defined earlier in the code as:

```
struct data {
  char *url;
  char *data;
};
```

Since you can get the result of processing the URL in any order, you wrap the result of processing in the above structure.

Next, in the `main()` function, you start as many coroutines as the number of URLs and then use the `chs()` function to send it the URL to download from. In the `f()` function, you don't actually attempt to connect to a remote URL, but simulate a scenario where there is some error in downloading 30% of the time, due to a faulty network. If there was an error, you write to the `error` channel; otherwise, you write to the `result` channel.

Back in the `main()` function, you have created all the coroutines, so now you want to just wait and report back on the result of trying to download data from each of the supplied URLs. You do this using the `choose` construct as follows:

```
for(int i=1; i<argc; i++) {
    choose {
    in(result, struct data, value):
```

```
      printf("Processed URL: %s, Result: %s\n", value.url, value.data);
    in(error, struct data, value):
      printf("Processed URL: %s, Error: %s\n", value.url, value.data);
    end
    }
 }
```

The number of data items you expect to have on both channels, `result` and `error`, is equal to the number of URLs. Hence, you create a for loop that runs for as many times. In the body of the loop, you create the `choose` construct.

The first `in` clause is used to wait for data to be available for reading in the channel result of type `struct data`. When there is a data item available to be read, a single data item is read and the variable specified, the value is declared by the `in` clause itself and is used to refer to the data item read from the channel. Similarly, the second `in` clause is used to wait for data on the error channel. Every `choose` construct must have an `end` clause at the end.

When either of the above two activities happens, you use `printf()` to write the URL processed and the result of the processing. When you compile and run the program, you'll see a result similar to the following:

```
$ gcc -o listing4 listing4.c -lmill

$ ./listing4  https://raw.githubusercontent.com/amitsaha/
➥lj_libmill/master/listing3/listing3.c
 ➥https://raw.githubusercontent.com/amitsaha/lj_libmill/
➥master/listing1/listing1-msleep.c

Processed URL: https://raw.githubusercontent.com/amitsaha/
➥lj_libmill/master/listing3/listing3.c, Error: Error
 ➥retrieving data
Processed URL: https://raw.githubusercontent.com/amitsaha/
➥lj_libmill/master/listing1/listing1-msleep.c, Result:
 ➥Data at the URL
```

By default, a program written using libmill will use only one processor core, even if your computer has multiple processor cores.

In addition to the `in` clause, `choose` supports a number of other clauses. The `out` clause can be used to wait until you can write to a channel, `ch`, and has the syntax `out(ch, <data type>, data)`. The `deadline` clause allows you to set a deadline in the `choose` construct—it will fire if no other clause has fired in the time specified as deadline. The `otherwise` clause will execute if none of the other clauses match.

Using Multiple Processors

By default, a program written using libmill will use only one processor core, even if your computer has multiple processor cores. The approach to using multiple cores is to use libmill's `mfork()` function to create a new process. Each process then becomes capable of doing the same work on a different processor core. Listing 5 shows an example of using `mfork()` to create a version of the earlier TCP server that creates multiple processes listening for incoming connections. Each process continues to handle an incoming connection in a separate coroutine as earlier.

When you compile and run the above program, you will see messages like the following, which tells you that you have three processes listening on port 9090:

```
./listing5
Listening on 9090 (PID: 3760)
Listening on 9090 (PID: 3762)
Listening on 9090 (PID: 3761)
```

Listing 5. listing5.c Using mfork() to Set Up a Multiprocess Network Server

```
# include <libmill.h>
# include <stdio.h>
# include <stdlib.h>
# include <errno.h>
# include <unistd.h>

# define NUM_PROCESSES 3

/* Handler coroutine */
coroutine void handler(tcpsock as) {
  printf("PID: %d, New connection!\n", getpid());
  tcpclose(as);
}

int main(int argc, char **argv)
{
  int port = 9090;
  ipaddr addr = iplocal(NULL, port, 0);
  tcpsock server = tcplisten(addr, 10);

  if (!server) {
    perror("Can't setup a listening server\n");
    return 1;
  }
  /* Set up the server processes - the main process is
     listening as well, so we fork NUM_PROCESSES-1
     child processes
  */
  for (int i = 1; i < NUM_PROCESSES; i++ ) {
    pid_t pid = mfork();
    /* Child process?*/
    if (pid == 0)
      break;
  }

  printf("Listening on %d (PID: %d)\n", port, getpid());
  /* Server loop*/
  while(1) {
    tcpsock as = tcpaccept(server, -1);
    if (!as)
      continue;
    /* Dispatch this request */
    go(handler(as));
  }

return 0;
}
```

The process tree looks like this:

```
$ pstree 3760
-+= 03760 amit ./listing5
 |--- 03761 amit ./listing5
 \--- 03762 amit ./listing5
```

If you try connecting to port 9090 from multiple clients, you will see that the connections will be handled by different processes, which is what you want.

The key change in this listing from the previous server is the following section of code in the `main()` function:

```
/* Set up the server processes - the main process is listening
  as well, so we fork NUM_PROCESSES-1 child processes
*/
for (int i = 1; i < NUM_PROCESSES; i++ ) {
    pid_t pid = mfork();
    /* Child process?*/
    if (pid == 0)
      break;
}
```

Similar to the `fork()` function, `mfork()` creates a new child process and returns 0 in the child process and the child process ID in the parent process. Hence, if the value of the pid you get is 0, you break from the loop; otherwise, you continue until you have forked `NUM_PROCESSES-1` number of processes.

Inspecting Coroutines

libmill comes with a function called `goredump()` that you can use to dump the state of coroutines and channels. You can call it as a function from your program or from the gdb prompt. Let's look at an example of the latter.

Consider the program in Listing 4 that uses multiple channels to send work and receives the results from the coroutines. Let's say you forgot

to send any work to the coroutine, because you accidentally removed the statement: `chs(work, char*, argv[i]);` (Listing 6). When you run the program, it simply will hang, doing nothing. So, let's see what's going on. First, find the process ID of the process and start gdb:

```
(gdb) attach 2132
..
```

Now, call the `goredump()` function and print its result:

```
(gdb) p goredump()
```

```
(gdb) p goredump()

COROUTINE  state                          current           created
-------------------------------------------------------------------------
{0}        choose(<2>,<3>)                 ---              <main>
{1}        chr(<1>)                      listing6.c:13     listing6.c:34

CHANNEL  msgs/max    senders/receivers      refs  done  created
-------------------------------------------------------------------------
<1>      0/0         r:{1}                    1     no    listing6.c:29
<2>      0/0         r:{0}                    1     no    listing6.c:30
<3>      0/0         r:{0}                    1     no    listing6.c:31
```

The output of `goredump()` has two sections: one for coroutines and other for channels.

For each coroutine, it displays the current state—or what it is doing currently, where it is currently executing and where it was created. You can see that the main coroutine (0) is waiting in the `choose` construct to read from the `result` and `error` channels (channel numbers 2 and 3, respectively).

The second coroutine (that is, the worker coroutine) is waiting at the function `chr()` to read from channel 1 at line number 13. So, that tells you why the program simply hangs—it should tell you that you haven't written anything to channel 1.

Listing 6. listing6.c Buggy Version of Listing 4

```
# include <libmill.h>
# include <stdio.h>
# include <stdlib.h>
# include <time.h>

struct data {
  char *url;
  char *data;
};

coroutine void f(chan work, chan result, chan error) {
  struct data d;
  char *url = chr(work, char*);
  d.url = url;
  /* Simulate 70% success and 30% failure scenario*/
  double r = (double) rand()/RAND_MAX;
  if (r < 0.71) {
    // Successful scenario
    d.data = "Data at the URL";
    chs(result, struct data, d);
  } else {
    // Unsuccessful scenario
    d.data = "Error retrieving data";
    chs(error, struct data, d);
  }
}

int main(int argc, char **argv) {
  chan work = chmake(char*, 0);
  chan result = chmake(struct data, 0);
  chan error = chmake(struct data, 0);

  for(int i=1; i<argc; i++) {
    go(f(work, result, error));
  }

  for(int i=1; i<argc; i++) {
    choose {
    in(result, struct data, value):
      printf("Processed URL: %s, Result: %s\n", value.url, value.data);
    in(error, struct data, value):
      printf("Processed URL: %s, Error: %s\n", value.url, value.data);
    end
    }
  }
  chclose(work);
  chclose(result);
  chclose(error);
  return 0;
}
```

The section on channels displays for each channel the number of messages currently in the channel and the maximum number of messages, senders/receiver coroutines, number of references to the channel, whether the sender is done sending to the channel and where it was created. You can see that for channels 2 and 3, both the readers are in coroutine 0 along with the line numbers.

Conclusion

This article looks at some of libmill's key features, including some convenient functions it makes available. A number of other libmill features provide building blocks to write efficient coroutine-driven network programs in C. See the Resources section for some important links to learn more about libmill.■

Amit Saha is a software engineer and the author of *Doing Math with Python* (No Starch Press). He blogs at http://echorand.me, and you can send him email at amitsaha.in@gmail.com.

Resources

The Code for This Article: https://github.com/amitsaha/lj_libmill

libmill: http://libmill.org

libmill Source Code: https://github.com/sustrik/libmill

libmill Tutorial: http://libmill.org/tutorial.html

libmill Documentation: http://libmill.org/documentation.html

libmill 1.0 Release Announcement: http://www.freelists.org/post/libmill/libmill10-released

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

RETURN TO CONTENTS

The Forrester Wave™: Digital Experience Platforms, Q4 2015

The demand to be at every touchpoint in the customer lifecycle is no longer an option—it's a requirement. To manage and deliver experiences consistently across all touchpoints, organizations are looking to digital experience platforms as the foundation of their digital presence.

Get Forrester's evaluation of the best vendors, including:

- The ten providers that matter most.
- How each vendor stacks up to Forrester's criteria.
- Six needs a digital experience platform architecture must meet.

> **http://geekguide.linuxjournal.com/content/forrester-wave-digital-experience-platforms-q4-2015**

ACQUIA™ The Ultimate Guide to Drupal 8 by Acquia

With 200+ new features and improvements, Drupal 8 is the most advanced version of Drupal yet. Drupal 8 simplifies the development process, enabling you to do more, in less time, with proven technologies that make it easier to be a first time Drupal user. Read this eBook, written by Angie Byron (you may know her as "webchick"), to get up to speed on the new changes in Drupal 8. Drupal 8's improvements include:

- API-driven content approach.
- Rest-first native web services.
- Seamless integration with existing technologies.
- Multilingual features and capabilities.
- Responsive by nature and mobile-first.

> **http://geekguide.linuxjournal.com/content/ultimate-guide-drupal-8**

ACQUIA™ How to Choose a Great CMS by Acquia

Web Content Management Systems serve as the foundation of your digital experience strategy. Yet many organizations struggle with legacy proprietary products that can't keep pace with the new realities of digital marketing. To determine if you are in need of a new CMS, use our guide, which includes:

- An evaluation to see if your current CMS supports your digital business strategy.
- The top considerations when selecting a new CMS.
- A requirements checklist for your next CMS.
- Ten questions to ask CMS vendors.

> **http://geekguide.linuxjournal.com/content/how-choose-great-cms**

Fast/Flexible Linux OS Recovery

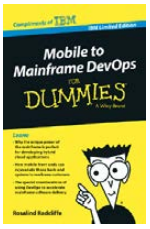
How long does it take to restore a system, whether virtual or physical, back to the exact state it was prior to a failure? Re-installing the operating system, re-applying patches, re-updating security settings takes too damn long! If this is your DR Strategy, we hope you've documented every change that's been made, on every system?!

Most companies incorporate backup procedures for critical data, which can be restored quickly if a loss occurs. However, that works only if you have an OS to restore onto and the OS supports the backup.

In this live one-hour webinar, learn how to enhance your existing backup strategies for complete disaster recovery preparedness using Storix System Backup Administrator (SBAdmin), a highly flexible full-system recovery solution for UNIX and Linux systems.

Webinar: April 26, 2016 at 1:00 PM Eastern

> **http://www.linuxjournal.com/storix-recovery**



Mobile to Mainframe DevOps for Dummies

In today's era of digital disruption empowered by cloud, mobile, and analytics, it's imperative for enterprise organizations to drive faster innovation while ensuring the stability of core business systems. While innovative systems of engagement demand speed, agility and experimentation, existing systems of record require similar attributes with additional and uncompromising requirements for governance and predictability. In this new book by Rosalind Radcliffe, IBM Distinguished Engineer, you will learn about:

- Responding to the challenges of variable speed IT.
- Why the mainframe is a unique and ideal platform for developing hybrid cloud applications.
- How mobile front ends can rejuvenate back-end systems to reach new customers.
- And, special considerations for using a DevOps approach to accelerate mainframe software delivery.

> **http://devops.linuxjournal.com/devops/mobile-mainframe-devops-dummies**

BRAND-NEW EDITION!

DevOps For Dummies – New Edition with SAFe®

In this NEW 2nd edition, learn why DevOps is essential for any business aspiring to be lean, agile, and capable of responding rapidly to changing customers and marketplace.

Download the E-book to learn about:

- The business need and value of DevOps.
- DevOps capabilities and adoption paths.
- How cloud accelerates DevOps.
- The Ten DevOps myths.
- And more.

> **http://devops.linuxjournal.com/devops/devops-dummies-new-edition-safe**

A New Project for Linux at 25

A monetary mutation to end financial serfdom.

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: Coroutines and Channels in C Using libmill

John McPhee says his books on geology could all be compressed to a single statement: *the summit of Mt. Everest is marine limestone*. We can do the same for Linux with this one: *microsoft.com is hosted on Linux*. According to Netcraft, so are dozens of other Microsoft sites and services (http://searchdns.netcraft.com/?restriction=site+contains&host=microsoft.com&lookup=wait..&position=limited).

For much of Linux's early history, Microsoft was an enemy—maybe even *the* enemy. Even as late as 2001, Steve Ballmer, Microsoft's CEO and ball-buster-in-chief, called Linux a "cancer" (https://slashdot.org/story/01/06/01/1658258/ballmer-calls-linux-a-cancer). Yet, Microsoft introduced its own Linux distro in 2003 (http://www.mslinux.org), and by 2011

(according to *Linux Weekly News*), Microsoft was the fifth largest corporate contributor to the Linux kernel (http://www.zdnet.com/article/top-five-linux-contributor-microsoft)—and Ballmer was still in charge. In November 2014, new Microsoft CEO Satya Nadella said the company "loves Linux" (http://arstechnica.com/information-technology/2014/10/microsoft-loves-linux-as-it-makes-azure-bigger-better). And why not?

As Linus put it to *IEEE Spectrum* earlier this year, "if you're creating some new Internet infrastructure or whatever, I'm almost surprised when it doesn't run Linux" (http://spectrum.ieee.org/computing/software/linux-at-25-qa-with-linus-torvalds). Microsoft makes a lot of "whatever", and it unsurprisingly described Azure Cloud Switch, introduced in September 2015, as "a cross-platform modular operating system for data center networking built on Linux" (http://blogs.microsoft.com/firehose/2015/09/18/microsoft-flicks-on-azure-cloud-switch-an-os-built-on-linux/#sm.0001dhc7pdc62ehaw801mzbdiezar). At the time, 20% of the company's virtual machines (VMs) on Azure (its cloud computing platform) already ran on Linux. I'll bet even more do now.

We see similar developments on the mobile front. For example, last year Microsoft invested in the Android developer Cyanogen (https://www.cyngn.com), the corporate spin-off of open-source (and Android/Linux-based) CyanogenMod (http://www.cyanogenmod.org) and maker of the commercial CyanogenOS (https://cyngn.com/cyanogen-os). As I write this, Microsoft is laying off Windows Phone workers by the thousands (http://www.theinquirer.net/inquirer/news/2459340/microsoft-axes-1-850-more-staff-from-mobile-biz-as-windows-phone-goes-into-freefall), while using its Cyanogen partnership to bring Microsoft apps and services into the massive Android world of mobile everything (http://www.thecountrycaller.com/29464-microsoft-corporations-msft-strategic-cyanogen-partnership-manifests-into-platform-integration).

As it happens, this month (August 2016) is the 25th anniversary of Linus' Usenet post announcing the birth of Linux, which at the time had no name and Linus called "just a hobby" (https://groups.google.com/forum/#!msg/comp.os.minix/dlNtH7RRrGA/SwRavCzVE7gJ). He closed that post saying his new OS "probably never will support anything other than AT-harddisks, as that's all I have :-("

Now all ten hosts on Netcraft's list of the world's Most Reliable

What most impressed me about Kevin was that he had, to my knowledge, the only digital identity business that gave full respect to the autonomous and independent natures of individual human beings and each connection those human beings might have to any other party, without relying on a central authority.

Hosting Company Sites (http://news.netcraft.com/archives/2016/06/03/most-reliable-hosting-company-sites-in-may-2016.html) run on Linux :-)

The problem with that smiley is that it says Happily Ever After, which is what you reach at the end of a story, not what you have at the beginning or the middle of one. As I said here in June 2016, all stories are about conflict, and Linux seems done with those.

To stay interested (and interesting, which is our job as a magazine), we need some new fights. In that same article, I listed five possibilities, all (at least hopefully) dear to the hearts and minds of *Linux Journal* readers. In this article, I'll dive into one of those: *Decentralization and Distributed Everything*. And, I'll do it by focusing on the mission of one guy: Kevin Cox (https://kevinrosscox.me), of Canberra, Australia.

I first met Kevin a few years ago at a conference in Munich. What most impressed me about Kevin was that he had, to my knowledge, the only digital identity business that gave full respect to the autonomous and independent natures of individual human beings and each connection those human beings might have to any other party, without relying on a central authority. In this way, his system embodied a quality Brian Behlendorf (http://brian.behlendorf.com) calls *minimum viable centralization*.

Not long after that conference, Kevin sold his company (which has since disappeared into its acquirer) and started thinking and working more deeply on liberating each of us from the clutches of centralization and making *distributed everything* not only work, but

Figure 1. John William Waterhouse's "Ulysses and the Sirens" (1891)

out-scale anything centralized.

When I asked Kevin to help me grok where he's headed these days, he told me to start with Cory Doctorow's talk at the Decentralized Web Summit in early June 2016 (http://boingboing.net/2016/06/09/internet-greybeards-and-upstar.html). Cory's one slide in the talk is of "Ulysses and the Sirens", John William Waterhouse's 1891 painting based on Homer's *Odyssey* (Figure 1).

In the painting, we see Ulysses tied to the mast of his ship at his own request so he won't be tempted toward rocky shores by the beautiful Sirens. (Those are the birds with female faces. They don't look tempting to me, but I suppose they were porn for Waterhouse.) Cory says this illustrates "'Ulysses pacts': bargains you make with yourself when your willpower is strong to prevent giving into temptation later when you are tired or demoralized, and how these have benefited the web to date, and how new, better ones can protect the decentralized web of the future" (http://boingboing.net/2016/06/09/how-will-we-keep-the-decentral.html).

The temptation Kevin would save us from is dependence on third parties of any kind, even things like OpenID (https://en.wikipedia.org/wiki/OpenID) and blockchains (https://en.wikipedia.org/wiki/Blockchain_(database)). He says "these take things away from your autonomy. You cannot suddenly

decide that you no longer want to be identified with a blockchain identity. You cannot ditch your OpenID any more than you can easily change your email address."

Next Kevin points me to promise theory (https://kevinrosscox.me/2016/06/02/promise-theory-and-financial-systems), which Wikipedia defines as "a model of voluntary cooperation between individual, autonomous actors or agents who publish their intentions to one another in the form of promises" (https://en.wikipedia.org/w/index.php?title=Promise_theory&oldid=724357922). In other words, promises work without a central authority or any kind of command and control system other than the individual's own.

This is a conceptual stretch for most of us computer types, because command and control is embodied in computer architecture, operations and programming. But they are not embodied in free societies. Holding those together are a matrix of promises, also known as agreements and understandings. In *Angels and Ages*, Adam Gopnik's excellent short book on Lincoln and Darwin (who were born the same day in history), he writes, "Law is the practice of rules in the context of deals, and Lincoln believed in both." Law is command and control. Deals are promises.

Most deals are not formal, but even the formal ones we call contracts simply certify agreements between two consenting parties (though technically, more can be involved). They don't require a third party. Government gets involved only when enforcement is at issue.

Mark Burgess (http://markburgess.org) first shared promise theory in a chapter of *Ambient Networks* (Springer, 2004) titled "An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation" (http://link.springer.com/chapter/10.1007%2F11568285_9). Ten years later, he wrote about it in *Linux Journal* (http://www.linuxjournal.com/content/promise-theory—what-it):

> In a promise-based design, each part behaves only according to the promises it makes to others. Instead of instructions from without, we have behavior promised from within. Since the promises are made by "self" (human self or machine self), it means that the decision is always made with knowledge of the same circumstances under which implementation will take place....

> A promise-oriented view is somewhat like a service view. Instead of trying to remote-control things with strings and levers, one makes use of an ecosystem of promised services that advertise intent and offer a basic level of certainty about how they will behave. Promises are about expectation management, and knowing the services and their properties that will help us to compose a working system.

Promises, he says, scale:

> ...biology has selected redundant services as its model for scaling tissue-based organisms. This offers a strong clue that we are on the right track. Avoiding strong dependencies is a way to avoid bottlenecks, so this shows the route to scalability.

Yet, he also warns:

> Autonomy and standalone thinking seem to fly in the face of what we normally learn about programming—that is, to share resources, but this is not necessarily true. Security and scalability both thrive under autonomy, and complexity melts away when the dependencies between parts are removed and all control is from within.

Within the promise, that is. (You can read more about promise theory in Mark's book, *Thinking in Promises*, published by O'Reilly in 2015.)

According to mathematical sociology (https://en.wikipedia.org/wiki/Mathematical_sociology), promises work across weak interpersonal ties (https://en.wikipedia.org/wiki/Interpersonal_ties#Weak_tie_hypothesis). Says Wikipedia, "Weak social ties, it is argued, are responsible for the majority of the embeddedness and structure of social networks in society as well as the transmission of information through these networks."

The other two kinds of ties are *strong* and *absent*. What we have in computing are mostly strong ties. Those are what enable us to issue commands and to build centralized products and services. Looking at Facebook through the framework of mathematical sociology, we see a gigantic service with strong ties to billions of human dependents who do their best to operate within the company's centralized system, paid for

by a B2B advertising business. In spite of being called a "social network", Facebook cannot scale human promises in truly social ways, because it doesn't support the weak ties that comprise genuine social networking—nor can other centralized organizations, such as banks, governments and large retailers.

Yet the internet can do that, since its own geology is a set of simple protocols, which are nothing more than agreements between computing entities at end points. Its protocols say information packets should be sent between end points by whatever routes look best along the way, and retransmitted when packets are dropped. These protocols are not encumbered by billing, or any business model, and there is no central entity controlling them. The internet's success is a demonstration of promise theory at work on a worldwide scale. Those of us on it comprise a rising tide of weak ties that lifts everything that floats on it, including giant corporate pyramids.

The net is also an ideal environment for scaling up countless promises between autonomous entities, unencumbered by the need for anything underneath to create scarcities or to impose operational or regulatory burdens.

The new promises Kevin wants to start with are hacks on the economy. Specifically, he wants us all to "take control of our own money" by "removing half the cost of financing the production of goods and services". That cost is financial institutions. He says we don't need them any more than the internet needs a central operator. He also says we can scale up far more economic activity and wealth generation with an economy that emerges from our autonomous activity, based on promises between entities that don't need third parties.

For example, instead of going to banks or other third parties for loans or savings, we can go to each other. Kevin calls this *envesting*, which he describes in an email as:

> ...a beneficial mutation of a bank loan, which is nothing more than a transfer of value over time.
>
> Envesting creates more value for the same amount of money as a bank loan. Bank loans create debt, which is expensive money that

we must repay.

Bank loans themselves can be thought of as a mutation to the exchange of value. The mechanism for creating bank loans is that the bank creates new money, then lends it. When the loan is repaid with money, the bank destroys the money.

Banks do not have to do it that way. They can lend money they have in their bank without creating more; and lots of financial institutions do. It is the creation and destruction of money that leads to inefficiencies.

The creation of money increases cost the more we create. Cost increases exponentially with the number of loans. This makes loans expensive.

Remove the need to create money and you have zero cost money. In fact, you already have it, in the deposits of savers. There is no cost of creation by using those funds to lend.

The loan is still the same. We exchange value and if there is a long transfer time, we return a little bit more value.

Envesting does not create extra money, like traditional bank lending. Instead it creates a form of money we call a voucher, or a prepayment, or a reward, or a coupon. These tokens are simple because you do not need a financial system to control them, and because they represent real goods (e.g. your house) and services. And, since we do not need the financial system to manage these instruments, we do not have to pay for an extraneous system that doesn't create any value for us.

In his book, *Killing the Host: How Financial Parasites and Debt Bondage Destroy the Global Economy* (http://michael-hudson.com/2015/09/killing-the-host-the-book), Michael Hudson explains how the financial sector has taken over the real economy. With envesting, we have a mutation that can immunize the real economy from that parasite. It can also be easily copied and applied in many different ways for many different loans.

> When we use fungible money to repay loans we have strong connections between loans. When we repay loans with product or percentage of ownership, we have weak connections, which is good. This change to the way we repay loans will reduce the cost of making loans. The total savings is likely to equal the full value of the loan.
>
> Using this approach for buying a house with rent rather than with repaying a money loan results in cost savings to both renter and saver. For example, here is the cost to buy a house with rent compared to a money loan: https://kevinrosscox.me/2016/06/03/existing-borrower. And here is the income to a saver who lends money to a renter, compared to lending money to a bank: https://kevinrosscox.me/2016/06/03/comparison-between-annuity-and-rent-and-buy-loans.

Kevin calls these arrangements "rent and buy loans". These are bankless loans involving just two parties. The original owner continues to own the property, like the bank holds collateral, until the loan is paid off, and ownership is transferred. The payments in the meantime are like rent. And the cost for the buyer is a lot lower. He also explains (https://kevinrosscox.me/2016/06/03/why-rent-and-buy-loans-are-great-value):

> Money created with rent and buy loans is zero cost, but it has the value of the property to which it refers.
>
> Money created with regular loans has a value *independent* of the asset to which it refers. That is, the money itself has a value. If we remove the cost of money, then we reduce the cost of creating and administering loans. These reductions in cost are passed on to renter/buyers and saver/lenders.
>
> There are many costs incurred making sure people do not create money without the backing of an asset. Unfortunately this has become difficult with the invention of derivatives and other financial instruments.
>
> The world is now awash in money because too much has been created. This happened because we can back the creation of money with other

money. If the underlying asset of this pyramid fails, then everything above fails. The underlying assets are affected because they are tightly connected to the failed loans.

Rent and buy loans are not connected tightly to any pyramid. This means if the money pyramid fails it does not affect rent and buy loans. Rent and buy loans use distributed systems to connect renters (buyers) directly to savers (lenders) without an intermediary (banks) who create another layer of cost in order to benefit from the previously difficult task of aggregating funds from savers and packaging those funds in order to re-sell them to borrowers and taking a fat cut along the way. Think of rent and buy loan platforms as a method for neutral connection of autonomous buyer/renters and lender/savers without any data extraction.

Personal data in Kevin's system isn't an issue because:

Applications don't "own the data". Uber could be an application built on top of this. But anyone could also write another Uber because they would have access to any data individuals allow. Ownership of the data always remains with the entities, including individuals. Entities give permissions to applications. The applications access the data. This is much cheaper than applications restricting access to data.

No, I don't fully understand it either. But I also didn't fully understand

free software (https://www.gnu.org/philosophy/free-sw.en.html) and the GPL (https://www.gnu.org/licenses/gpl-3.0.en.html) when I first encountered them, even though I knew in my gut that the world needed them.

I didn't understand them because there were too few examples of them at the time (the late 1980s and early 1990s), and because I was busy doing stuff like helping Sun Microsystems succeed with SPARC and trying to make network parts builders work together in compatible ways while the internet was still busy forbidding commercial activity (which didn't end until 1995).

After Linux took off (as GPL'd free software), I could see clearly how freedom worked because the means were there—not just for demonstrating it to everybody, but for developing more and more with it. I suspect the same could be true for promise-based financial dealings such as rent and buy.

So my request here is to help Kevin debug the case he makes for his ideas, while putting them to work.

It helps also to remember the introduction of Linux as a mutation that not only proved free software could work in the world, but utterly changed the norms of software development, liberating vast amounts of development labor from the feudal castles of corporations and governments, while creating far more development opportunity along the way—so much that today there's a worldwide shortage of programmers.

It also helps to have a big hairy enemy. The world's broken financial system is an ideal candidate. So let's do for the world's economy what we did for its software.■

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

RETURN TO CONTENTS