RUN A FULL VERSION OF R ON ANDROID

LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

WATCH: ISSUE OVERVIEW

JUNE 2016 | ISSUE 266
http://www.linuxjournal.com

Automate Certificate Maintenance with LET'S ENCRYPT

How to Organize Your Qubes VMs

+

BUILD a Raspberry Pi Camera

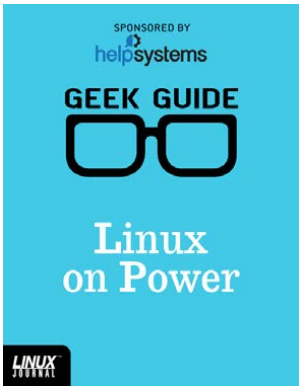
GETTING STARTED with nginx

NEW!

Ceph: Open-Source SDS

Author: Ted Schmidt

Sponsor: SUSE



Linux on Power

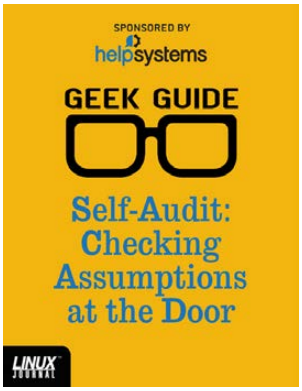
Author: Ted Schmidt

Sponsor: HelpSystems

SSH: a Modern Lock for Your Server?

Author: Federico Kereki

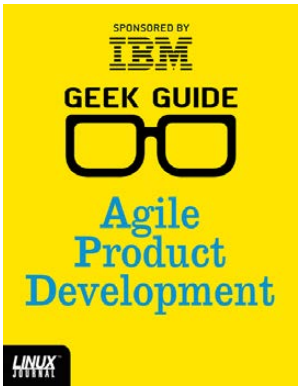
Sponsor: Fox Technologies



Self-Audit: Checking Assumptions at the Door

Author: Greg Bledsoe

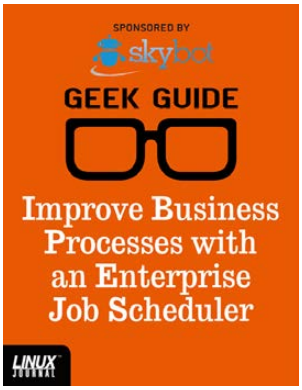
Sponsor: HelpSystems



Agile Product Development

Author: Ted Schmidt

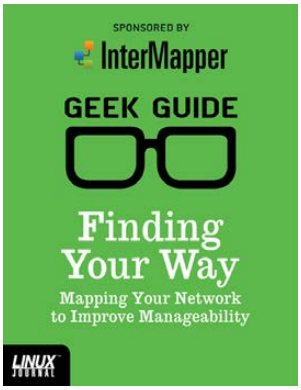
Sponsor: IBM



Improve Business Processes with an Enterprise Job Scheduler

Author: Mike Diehl

Sponsor: Skybot



Finding Your Way: Mapping Your Network to Improve Manageability

Author: Bill Childers

Sponsor: InterMapper

DIY Commerce Site

Author: Reuven M. Lerner

Sponsor: GeoTrust

CONTENTS

JUNE 2016
ISSUE 266

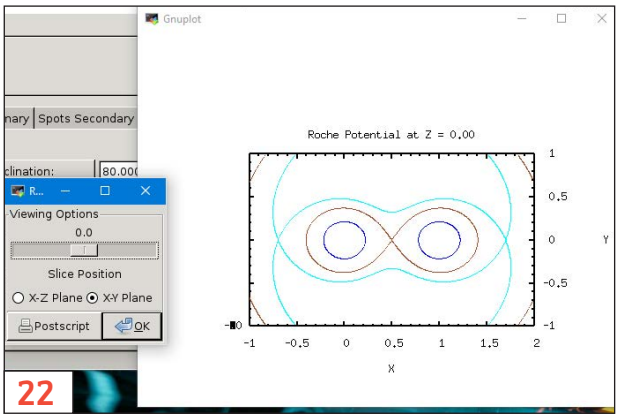
FEATURES

ON THE COVER

- Run a Full Version of R on Android, p. 90
- Automate Certificate Maintenance with Let's Encrypt, p. 78
- How to Organize Your Qubes VMs, p. 50
- Build a Raspberry Pi Camera, p. 58
- Getting Started with nginx, p. 32

COLUMNS

IN EVERY ISSUE



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



Sharpen your Android skills at

AnDevCon

The Android Developer Conference

World's Largest

BOSTON

August 1-4, 2016

Sheraton Boston

"Simply the best Android developer conference out there! A must-go if you do Android development."

—Florian Krauthan, Software Developer, Hyperwallet

Get the best Android developer training anywhere!

- Choose from more than 75 classes and in-depth tutorials
- Meet Google and Google Development Experts
- Network with speakers and other Android developers
- Check out more than 50 third-party vendors
- Women in Android Luncheon
- Panels and keynotes
- Receptions, ice cream, prizes and more!

www.AnDevCon.com

A BZ Media Event

Things to Do in the Shade

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 e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

It's summer time here in the northern hemisphere (although it did snow at my house on May 15, 2016—hopefully Mother Nature is done with that nonsense). When the sun is shining, there's nothing I like to do more than sit outside—under an umbrella, with SPF 50+ sunscreen and, of course, a book. These days, those books usually are digital, and since it's June 1st, that means a new issue of *Linux Journal* is ready to read.

This month, Reuven M. Lerner switches things up a bit and describes how easy it has become to use nginx as a Web server for your applications. It's been a viable and efficient option for several years now, but Reuven shows just how polished and easy it is to install and configure. Next, Dave Taylor does some polishing this month as well, as he puts the final touches on his wegrep series. I'm ashamed to admit that my scripting often never gets that last bit of polish, and it's unfortunate, because a good script should be flexible and portable. Dave shows how to take that last important step.

Kyle Rankin's latest series on the Qubes distribution has been incredible, and if you've been a little overwhelmed, no worries. In this issue, Kyle describes some of his real-life examples on how to use Qubes on a daily basis. The notion of virtualizing applications isn't new, but the way

VIDEO: Shawn Powers runs through the latest issue.

Qubes compartmentalizes everything you do is unique, and sometimes examples help to clarify things.

I make some clarifications this month as well, but in my case, I'm clarifying photos of birds. Specifically, I demonstrate my new homemade IP cameras built using Raspberry Pi devices. The camera modules for the RPis are cheap and incredibly high quality. It only makes sense to take advantage of that for BirdCam!

Andrei Lukovenko has an incredible article this month on HTTPS certificates. First off, he introduces the Let's Encrypt certificate authority (which I'd never heard of), and then goes on to walk through automating the certificate renewal process! I've been using StartSSL for years to get free SSL certificates, and even though the one-year expiration is a long time, it seems like every year, I have to relearn how to install certificates. Andrei explains how to make your computers do all the work, while getting more frequent SSL cert updates as well! It's an article you don't want to miss.

This month, you also will learn how to install the full R statistical software on a mobile Android device. Mobile devices are powerful enough that installing such packages makes sense, and with the help of Marius Hofert and Kurt Hornik, you'll learn how to do so using Linux in a chroot environment on Android. Even if you're not interested in installing R, the process for installing Linux inside Android is fascinating and fun! Marius and Kurt go through the entire process, including information on unlocking bootloaders and rooting devices.

This month's issue of *Linux Journal* is perfect for reading in the shade with a glass of iced tea and a hammock. Or, if you happen to be in the southern hemisphere, perhaps curled in a blanket with a cup of hot tea. Either way, I love issues where you learn to do cool things, and this one is full of ideas and projects. We also include product announcements, cool apps, tech tips and countless other nerdy tidbits that will help make your summer a bit cooler. We hope you enjoy this issue as much as we enjoyed putting it together!■

RETURN TO CONTENTS

LETTERS

PREVIOUS
Current_Issue.tar.gz

NEXT
UpFront

"The Power of Tiny initrd" Is Awesome

Eduardo Arcusa Les' article "The Power of Tiny initrd" in the March 2016 issue was an eye-opener for me. Everything is in the article, with real-world examples, line-by-line configurations and beautiful graphs. We need more articles like this. I was really entertained to read it as someone hoping to be a Linux sysadmin in future.

—Zongren

Eduardo Arcusa Les replies: *I really appreciate your words. This was my first article, and it was done with much affection and effort. That was exactly what I wanted, real examples to show people how powerful Linux is and what may be done with it. If you have had so much fun reading it as I have enjoyed writing it, you surely will be a good sysadmin in the future. And things will be even better if you have people around you that encourage and inspire you, which is how it was for me.*

Request for More Details on "The Power of Tiny initrd"

Eduardo Arcusa Les' featured article titled "The Power of Tiny initrd" in the March 2016 issue was one heck of an article! He just did what I've been searching for the past several years! I mean, a server running from a RAM—that's fudging shirt!

Although he has explained his adventure in some detail, I (and I suppose many others too!) would like to know more in-depth details,

such as how did he create an initrd and how to put those "changing" files outside initrd, but link them to initrd? Perhaps the author can write a blog post with the details?

Thanks for the article!

—Ron

Eduardo Arcusa Les replies: *Thanks, Ron, for your words. The truth is that it's amazing to see a server running completely in RAM, and I hope that my article has helped you in your progress to achieve it.*

I didn't explain how create an initrd, but there are a lot of how-tos on the Internet to accomplish that. All you need is to create a very simple initrd to start a server with PXE and save it as base-initrd. When you need to create a new server, make a copy of the base-initrd and put only files that almost never need to be changed (for example, binaries of services or libraries that those services need).

How to put those "changing" files outside the initrd but link them to initrd is explained in the article. Those files, like configuration files of the services, need to be on another server to be edited easily without changing the whole initrd. Then when the server boots, it executes /etc/rc to copy those files with sshfs before the services start. The /etc/rc script is within the initrd; you need to edit it to copy those files that are prone to change.

I hope this helps.

More Powers to Ya: Redneck Utilities Better Than Expected, Sort of

I suppose it is fun for everyone to complain about utilities, and I am no different. Last winter, I discovered that about half of my trailer's outlets stopped working one brutally cold 50°F morning. (This is Florida, so we are all wimps about cold weather and have some combination of poor insulation and shitty electric heat, even the fools who bought new houses in the real-estate boom.)

The previous summer, I'd had a few brown-outages that magically seemed to heal themselves by August, so I'd forgotten all about it.

Having run extension cords to all the working outlets to keep my servers and Raspberry Pi appliances running, I called the local electricity co-op to ask for some help. Before I could finish a damn good cup of coffee, there was a lineman out there poking around at the pole. He said one of the "hot" (120V AC) wire connections was corroded and had fallen off and then arc-welded itself back on a couple times. (I think there are two hots with a 180-degree phase difference so you can easily make "big appliance" 240V AC voltage by connecting across both hots.) I had a sinking feeling he was telling me this because he was not going to fix the problem, and I would be stuck dealing with a handyman—or worse, a licensed contractor. However, he got to the point, and I was pleasantly surprised: even though the utility's policy was not to work on anything on the client-side of the meter, it was such an easy fix that he would do it on the spot. After about 15 minutes of polishing with a wire brush, he put some conducting goop on the wire, tightened a nut and reconnected at the pole. For the record, he recommended I upgrade the meter, but said that there was probably no rush. That was two years ago. I won't go into specifics, but I would like to emphasize that this was *not* Duke Energy.

—Mike "Mighty Bush" Grossman

Shawn Powers replies: *Mike, I think maybe you should contact the company and tell them you'll sell the secret to "self-welding" electrical panels. Once they pay you, just spritz some salt water on the connections and tell them all they have to do is wait! (Only kidding of course, I'm glad the guy fixed it for you. I think sometimes professionals take for granted how "simple" something is for them compared to the rest of us!)*

New Format

I just want to drop you a quick note about the new format. I've been a longtime *Linux Journal* reader, and I have to say that's the greatest thing since the days when you printed the magazine on paper. Reads *so* much better on a tablet.

—David

Shawn Powers replies: *David, that's great to hear! I like it better too. I think we all liked the paper magazine so much, it was hard to think about what would be better than looking "how it always looked before". It's great to hear that the new format looks better for more than just me.*

Qubes, Hurrah!

I just read Kyle Rankin's article on Qubes in April's *Linux Journal*. I am happy to see this system get "airtime" and some explanation. I know I needed it.

When I first began to read about Qubes, I admit I was a little overwhelmed. But seeing this now, and with expectation awaiting the next articles in the series, I will probably install it on something in the near future.

So, thank you Kyle and editors.

—Jesse

The cal Command

In a letter published in the April 2016 issue, Wally Olson observed that the `cal` command prints some "space" "underbar" "backspace" sequences around the current date. (It actually prints terminal-specific escape sequences if its output goes to a terminal; the backspace sequence is used when printing to a file or pipe.)

The point of this is to cause the current date to be highlighted.

With the Debian/Ubuntu version of the `cal` command (provided by the bsdmainutils package), you can turn this off with `cal -h`.

The version used on Red Hat behaves differently. When output is sent to a file or pipe, it doesn't do any highlighting at all.

Incidentally, there's also a difference in trailing blanks, which you can see by typing `cal | cat -A`.

—Keith Thompson

strncpy

In a letter published in the March 2016 issue, Mischa Salle wrote:

> strcat, strcpy, sprintf and the like are dangerous and should be avoided unless in completely straightforward cases. The standard replacements for these are strncat, strncpy and snprintf, which are all three POSIX.

A minor detail: all those functions are defined by the ISO C standard, not just by POSIX.

The strncpy function is not a "safer" version of strcpy. If the destination array is too small to hold the source string, the target is not null-terminated, meaning that any subsequent attempt to treat it as a string will cause undefined behavior.

There are rare cases where strncpy is the right solution. Replacing strcpy

is not one of those cases.

I've written about strncpy here: http://the-flat-trantor-society.blogspot.com/2012/03/no-strncpy-is-not-safer-strcpy.html.

Furthermore, replacing the relatively unsafe unbounded string functions by bounded versions without very careful thought doesn't really solve anything. They can avoid overflowing the target array, but only by truncating the data. A contrived example: if the string `sudo rm -rf /tmp/unimportant_directory` is quietly truncated to, say, 13 characters, the result is `sudo rm -rf /`, and the consequences are likely to be worse than any buffer overflow.

You need to test whether the target array is big enough to hold the data you want to store in it and then decide specifically what to do if it isn't. Simple truncation is rarely the best response.

—Keith Thompson

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

PREVIOUS
Letters

NEXT
Editors' Choice

diff -u

What's New in Kernel Development

An effort to speed up **futex locks** for real-time software has hit a snag, though possibly not a permanent one. **Thomas Gleixner** posted some patches to eliminate collisions between the hash values used to track futex state changes. These collisions didn't break the locks; they just slowed things down. Thomas' code would speed up the threads that used his code, but only if most other threads allowed the collisions to take place. His reasoning was that real-time applications could call his routines, while normal code could do things the old way.

There were various objections. One was that, as **Linus Torvalds** put it, "the *last* thing you want is programmers saying 'I'm so important that I want the special futex'. Because every single programmer thinks they are special and that *their* code is special. I know—because I'm special."

But his main objection, first voiced by **Ingo Molnar**, was that Thomas' code introduced an **ABI** (application binary interface) change that was highly specialized, but that would have to be carried into the future and supported in the kernel on a permanent basis, long after any value it once possessed had disappeared.

The solution, proposed by Ingo, was to enable Thomas' code for all futex use by default. This would have the annoying quality of making

the code less useful for real-time applications, because all other applications would see the same benefit. But, it had the benefit of actually being an improvement even if everyone used it equally, and also of avoiding the ABI change.

Binary interface changes are killers. I think Linus would staple his arm to the table before he'd accept a patch that causes existing compiled software to break. Application programmer interfaces (APIs) are different. Those can be deprecated and replaced gradually, as users update their source code to use the new procedures. It's a hassle and takes a long time, but it's doable. ABI changes? Not so much.

The nightmare holy grail of **cgroup** implementation continues at a breakneck pace. Trying to make a single piece of hardware look like it's actually multiple independent systems, and having that be secure, while letting all software run natively on the original hardware, turns out to be one of the more insane of the many world-changing things that have found Linux at their absolute center.

Adding features to cgroups is like trying to climb up a sheer cliff using only your teeth and the tips of your toes. The cadre of the insane includes **Bandan Das**, who recently submitted some patches to get **workqueues** running under cgroups.

Workqueues are primarily a coding construct that allows user code to set certain low-priority tasks in motion, memory allocations and whatnot, without having to wait for them to finish. Workqueues typically are handled by dedicated worker threads that plow through them as system load permits.

One of the features of Bandan's patches was to allow a given workqueue to be handled by a worker thread that was dedicated to the particular virtual server that was associated with that workqueue. This way resource accounting wouldn't break, and the underlying system could track properly which virtual server should be given how much RAM, CPU time and so on.

Bandan's patch didn't encounter the kind of byzantine security objections that often hit cgroup features, but it did exhibit some unexplained slowdowns. Once those are resolved and Bandan submits the code for actual inclusion in the kernel, the security hawks undoubtedly will descend from their watchful perches for feeding time.

Shuah Khan has been working on stopping media devices from periodically hanging the system under certain circumstances. Apparently, some of the existing media device code would allow users to release a device while it was still in use, after which bad things would happen, up to and including a full-on system crash. She posted some patches to implement a proper sequence of events of media device shutdown that also would account for the fact that any given media device may have multiple users and various pieces of software waiting to grab it.

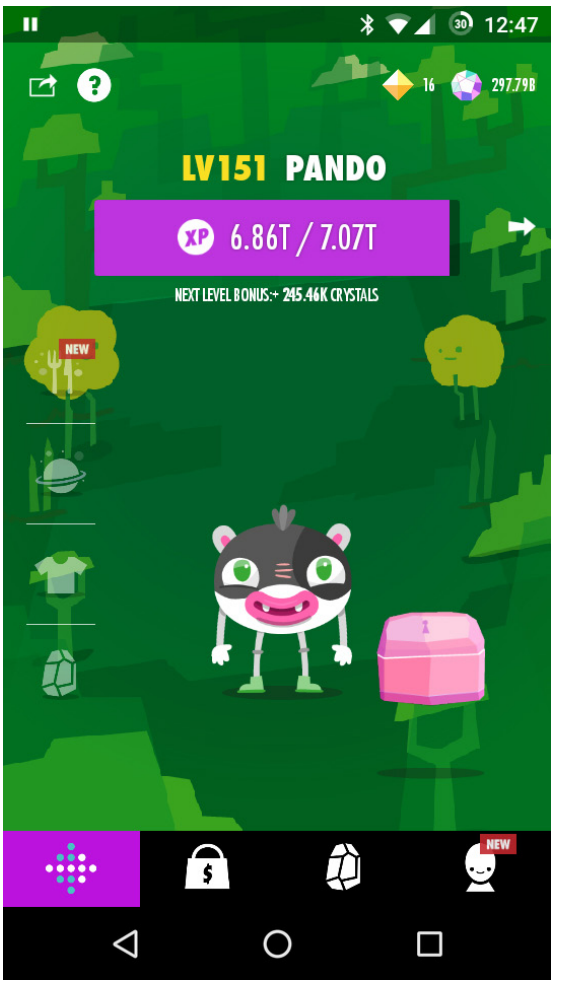
During the course of discussion, Shuah, **Takashi Iwai** and **Mauro Carvalho Chehab** discussed how to organize the work so that it could be developed in an ongoing way and possibly made useful for more than just media devices. They planned to nail down the API so user code could start using it, then to begin to flesh out to fixes and features, and finally to put the whole thing into git, rather than have it be just a set of patches coming in through e-mail.

Jens Axboe hurled his keyboard across the room, ran to the window and threw it open and shouted into the dark of night, "Since the dawn of time, our **background buffered writeback** has sucked!!!!!!!" Enough was enough. He was sick of waiting for background writes to finish while managers and users screamed at him that their production systems had locked up. Not only that, but his **Chrome** was slower too.

He posted some patches to relegate background buffered writeback to its proper place in the universe—a background process that doesn't interfere with user activity.

Holger Hoffstätte loved the code and backported it to Linux 4.4.x, reporting wonderful smoothness. **Dave Chinner**, on the other hand, concocted a test suite that was able to expose some performance problems with Jens' code that made the system a bit worse than it had been before. He and Jens worked on reproducing and fixing that issue together.—Zack Brown

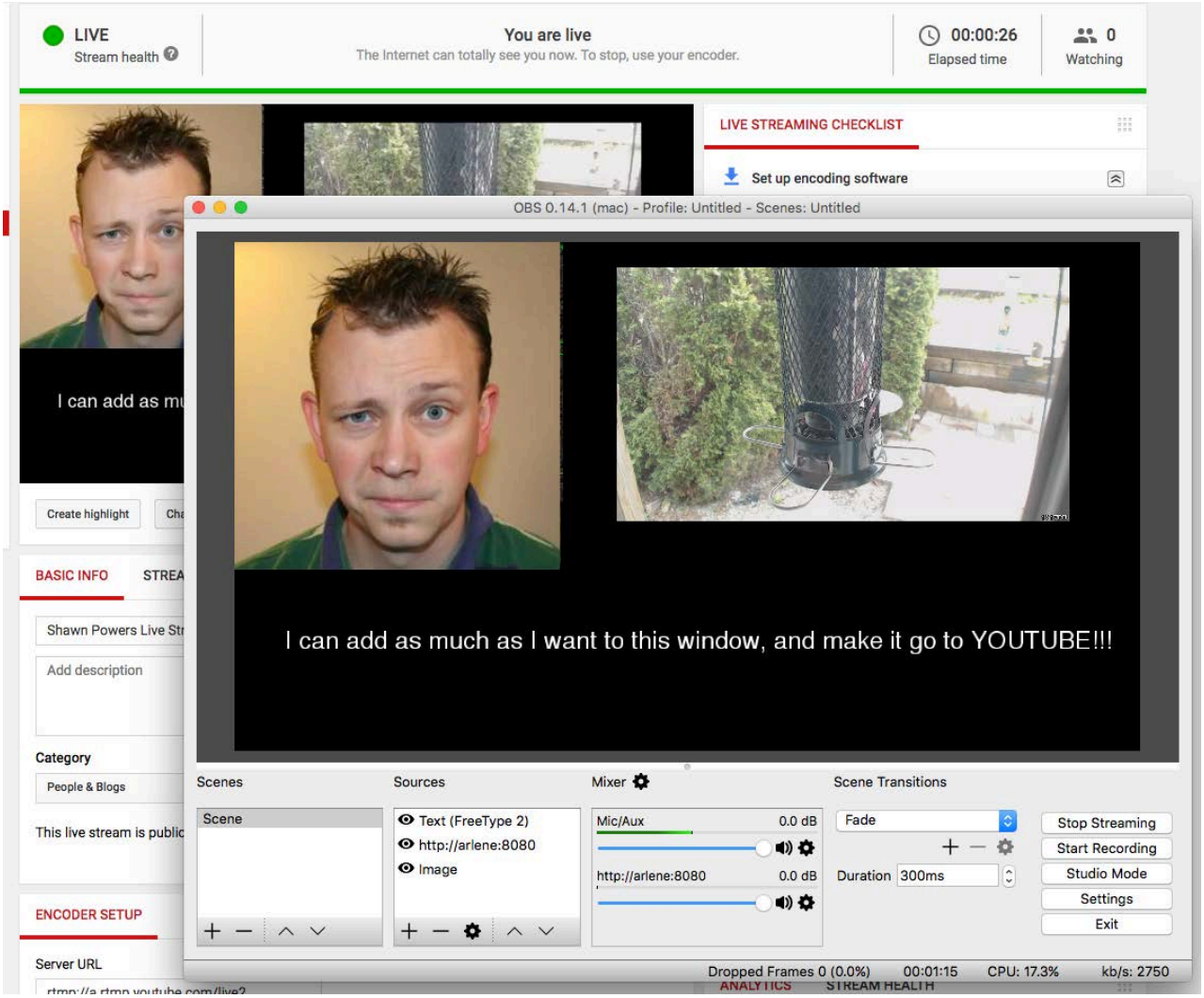
Android Candy: More Life Gamification



You might remember a couple months ago my mention of Habitica, which is a gamification of your daily to-do list. One of my friends on Twitter mentioned an app he uses on Android called Wokamon, which ties in with your FitBit (or any of several other "step-counter" devices). Based on how many steps you take in a day, your Wokamon grows and evolves into a larger, more powerful...well...Wokamon.

It's absurd, but it's still incredibly fun to see your steps make a little Tamagachi-like digital critter grow. The other aspect of the game is that tapping on the screen will earn you crystals, which can be used to purchase upgrades. Those upgrades make your Wokamon grow faster. You also can watch short ads to get a treasure chest full of crystals. That allows you to buy even more things and also gives the game's developers a little ad revenue for their efforts.

I have no idea whether Wokamon is something I'll keep playing with, because there doesn't seem to be much in the way of competition. Maybe that's just because I'm the only person I know in real life who uses it. Still, it's fun for now, and it has forced me to consider another project for the future—namely, how to create an automated "screen tapper" so I can earn crystals all night long while I sleep. I figure anything that helps me think of crazy future projects is worth my time. If it sounds interesting to you, just search for "Wokamon" in the Google Play store. It's free and oddly entertaining.—Shawn Powers



When Birdcam Goes Mainstream

If you read my articles on when I originally set up BirdCam a few years ago, you'll remember I did it with compatibility in mind. At the time of this writing, BirdCam (http://birds.brainofshawn.com) is simply an HTML page with the JavaScript language to refresh the images constantly, in order to create a low-fps video stream of sorts. One of the frustrations is that if I want to rearrange camera images or add a new camera (check out my Open-Source Classroom

column in this issue), it means a lot of complex HTML coding. It also means the "stream" is less and less reliable, because it depends on multiple images refreshing several times a second. I want to do something more powerful.

Enter: YouTube.

My goal is to get BirdCam to stream a live video to YouTube Live, so it can be embedded on Web sites, viewed on mobile devices and even "tuned in" via Roku or Fire TV. I tried in vain to get ffmpeg and/or VLC to stream video to YouTube from the command line, but I couldn't ever get it working reliably. I haven't given up hope, but until then, I'm planning to play with Open Broadcaster Studio.

It's an open-source, cross-platform studio package that allows you to put multiple video streams, text boxes and still images into a single window that is encoded and streamed to YouTube. I couldn't get the OpenGL version required to work on my Ubuntu laptop, but the OS X version worked flawlessly in my test. With a little bit of tweaking, BirdCam 3.0 might be a legitimate 24/7 YouTube stream. If you want to play around with streaming live to YouTube, check out the awesome open-source studio software at http://obsproject.com.

—Shawn Powers

THEY SAID IT

Broadly speaking, the short words are the best, and the old words best of all.

—Sir Winston Churchill

Brains, like hearts, go where they are appreciated.

—Robert S. McNamara

There are no secrets to success. It is the result of preparation, hard work, and learning from failure.

—Colin Powell

You cannot be mad at somebody who makes you laugh—it's as simple as that.

—Jay Leno

It is better to look ahead and prepare than to look back and regret.

—Jackie Joyner-Kersee

Nightfall on Linux

In my last few articles, I've looked at general astronomy programs that are helpful for many tasks you might need to do in your stargazing career. But, several specific jobs are more complicated and require specialized software to make relevant calculations, so in this article, let's look at Nightfall (http://www.hs.uni-hamburg.de/DE/Ins/Per/Wichmann/Nightfall.html).

Nightfall is a program that can handle calculations involving binary star systems. It can animate binary star systems, taking into account not only orbital speeds but also rotational motion and the changing shape of stars due to their close positions. You can model what it would look like and what kind of light curves you would register when observing a binary system. You even can take a set of actual observational data and find a best-fit model for the system you are studying.

Most distributions don't include a package for Nightfall, so you need to build it from source. There are several dependencies, so the instructions following assume that you are using a Debian-based distribution. If you are using something else, you should be able to find the comparable packages for your distribution of choice. To install the dependencies, run the command:

```
sudo apt-get install libgtk2.0-0 libgtk2.0-dev gnuplot
```

Nightfall also includes the ability to use OpenGL to handle 3D rendering of animations of the binary systems you want to model. If you want to use OpenGL, you also need to install:

```
sudo apt-get install libgl1-mesa-dev freeglut3-dev
 ➥libgtkgl2.0-dev libjpeg62-dev
```

Once all of the requirements are installed, you can install Nightfall itself. You should change directory into a temporary or source directory where you can do the unpacking and build the code. Once you are there, download the latest version of Nightfall with:

```
wget http://www.la-samhna.de/nightfall/nightfall-1.88.tar.gz
```

Then, unpack it with:

```
tar xvzf nightfall-1.88.tar.gz
```

When you go to configure Nightfall, you probably will want to include the `openmp` option. This allows Nightfall to use the multiple CPUs you probably have in your machine to speed up the calculations involved. You can build and install Nightfall with:

```
./configure --enable-openmp make sudo make install
```

This installs Nightfall under the /usr/local directory. You then can start Nightfall with:

```
nightfall -U
```

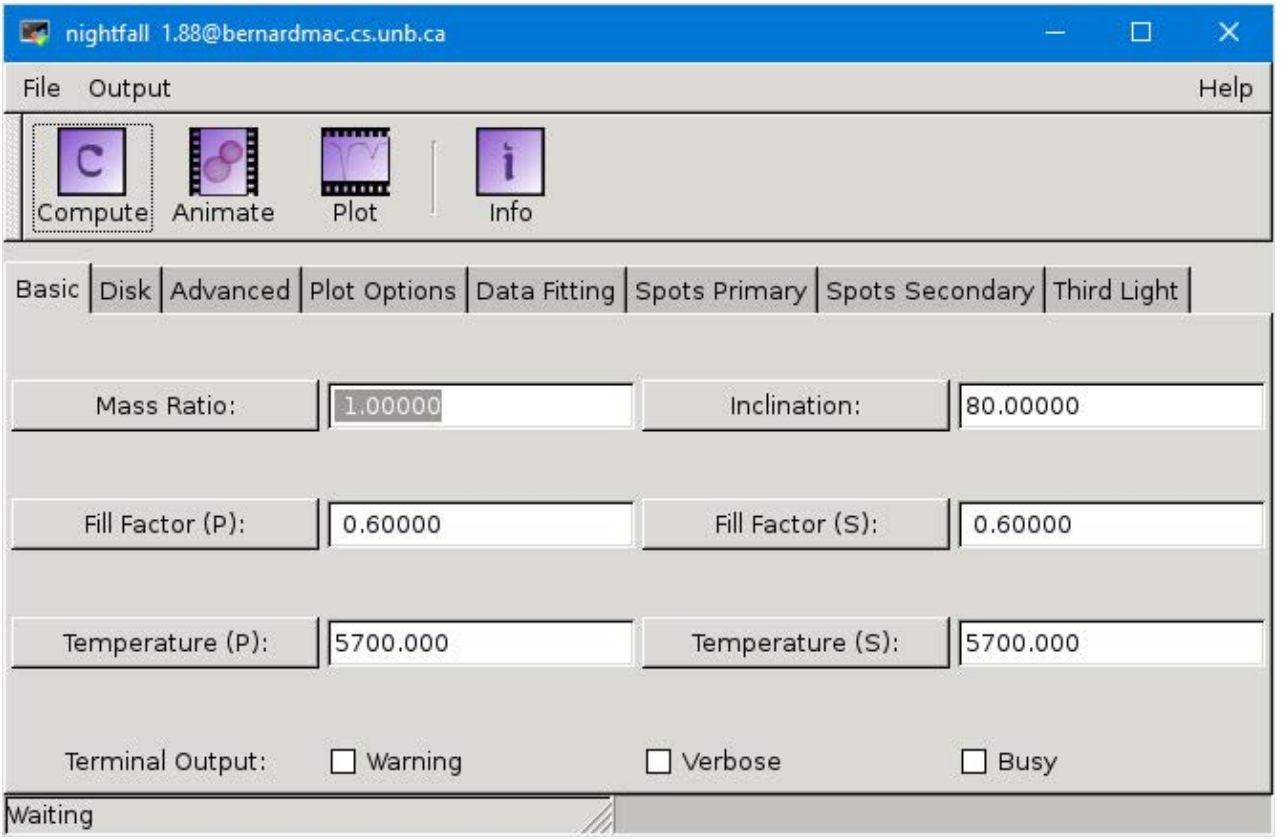


Figure 1. The GUI lets you configure all kinds of options within Nightfall to set up your model.

The `-U` option is necessary to force the GUI to be used interactively.

Now that Nightfall is up and running, you will start to see just how much control you have over the model that is being simulated. The first tab is where you can set up the core parameters for your binary system model. You can set the mass ratio and the inclination of the two stars. You also can set the surface temperatures and the Roche lobe filling factors for each of the stars. The temperature helps define their luminosities, and the Roche lobe filling factors define the distortion of the stars.

Depending on the exact conditions, you may have an accretion disk of material around the central star. The second tab lets you set

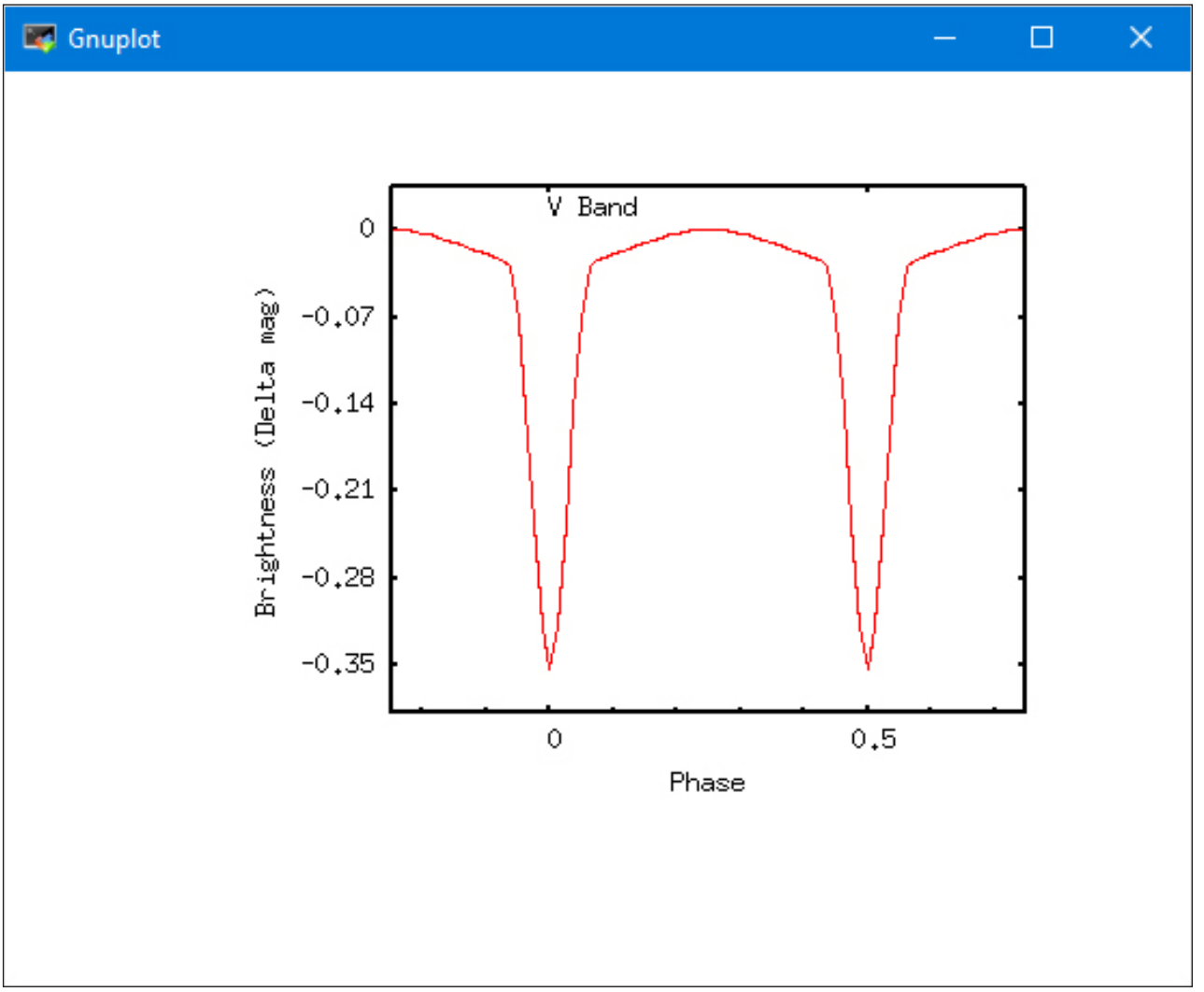


Figure 2. Once the calculation is done, you can plot the observed light curve.

the type of accretion disk (that is, how it interacts with the central star), along with more physical parameters like the inner and outer diameters and the temperature.

The advanced tab allows you to set some less obvious parameters for your model, such as the eccentricity for the orbiting star and whether you also need to model the atmosphere.

Now you can click on the Compute button at the top of the window, and on most modern machines, it goes pretty quickly. Once the calculations are done, you can plot the output from the system you just modeled.

The Output menu item on the menu bar at the top gives you

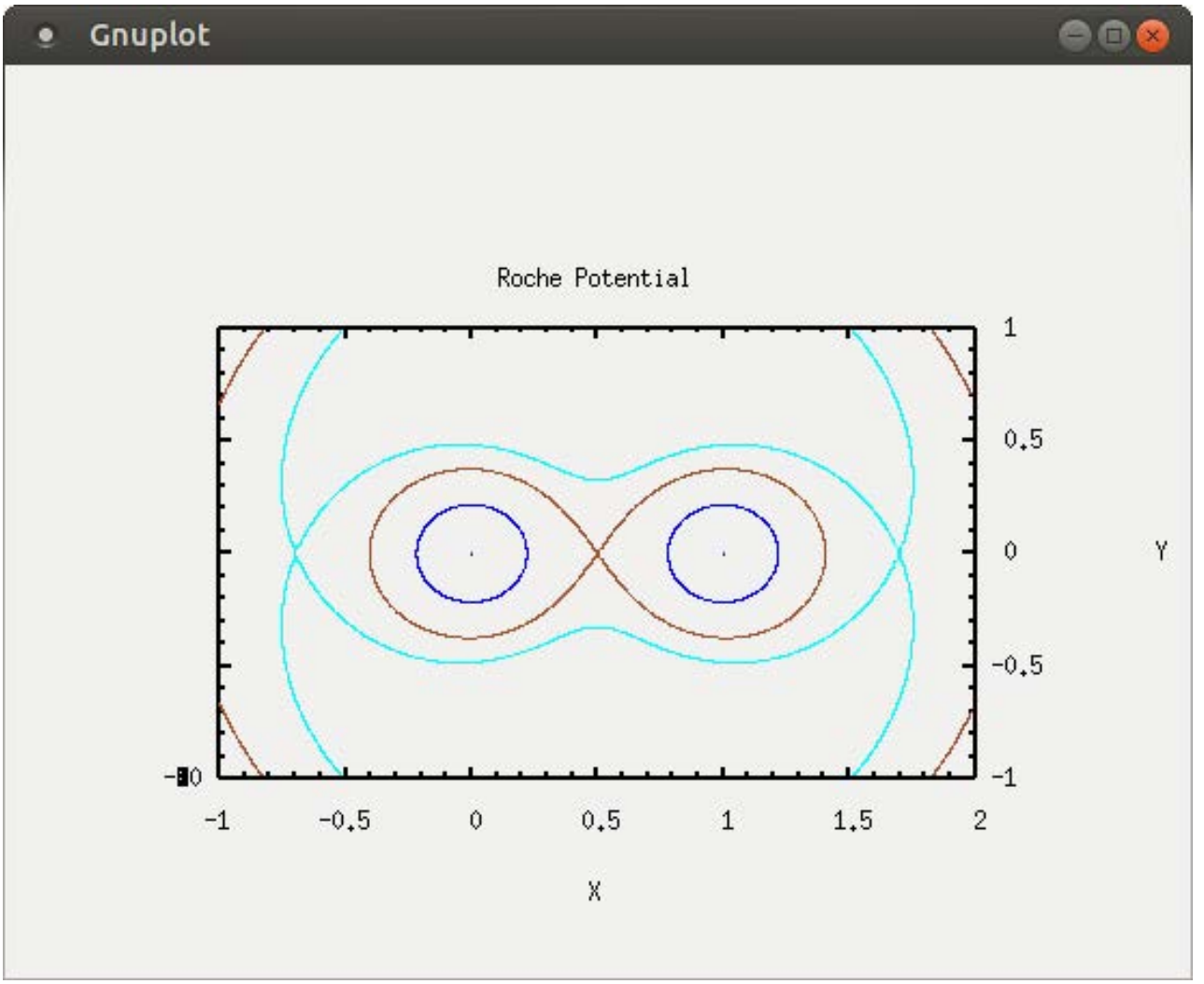


Figure 3. You can plot the geometry of the stars within the binary system.

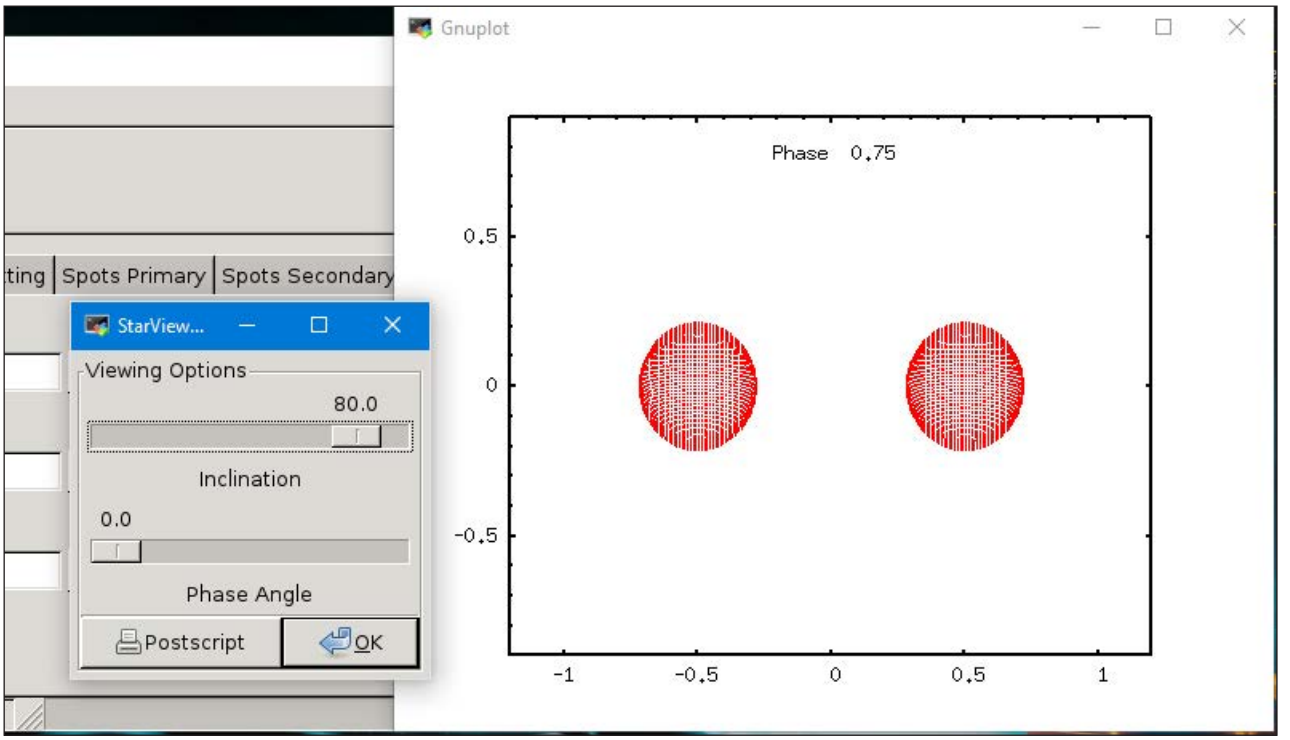


Figure 4. Selecting StarView lets you see the binary system from some distance away.

several options on how to display the calculated results. The first entry is PlotCurve, which draws the light curve as seen from a distant observer. Gnuplot is used to draw the actual plot of the visible amount of light that is seen.

The second output option is to select ViewGeometry. This plot shows you how the stars themselves are distorted within their orbits.

The StarView menu item presents a view of how the binary system would look from your observation point. A control box also pops up that allows you to change the inclination of the orbital plane and the phase of the orbit.

The RocheSlicer menu item provides a way to visualize just how distorted the stars become. A control box also pops up that lets you select various slices through the star system.

The last menu item is the DataSheet option. This option pops up a new window with a text description of the results of all of the calculations that were made.

One of the keys in science is being able to reproduce your results, both experimental and computational. With this in mind, you can

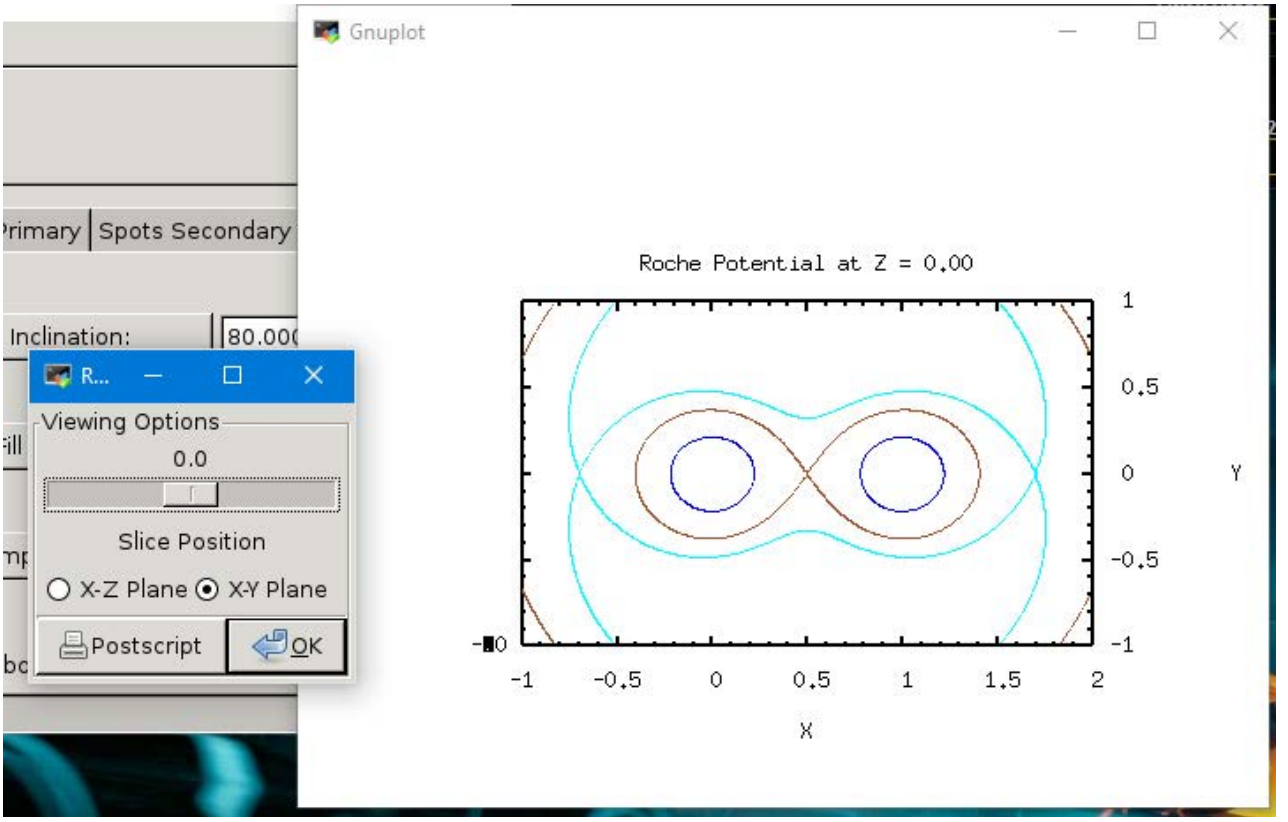


Figure 5. The RocheSlicer lets you investigate the distortions in the stars of your binary system.

save the model you just ran and load it again later. These models are saved as configurations by clicking the menu item File→Save Configuration. You can load previous models by clicking the menu item File→Open config file. If you compiled and installed Nightfall using the instructions above, you will have a number of example configurations available that you can play with as well.

Nightfall is not only useful in determining what a given binary star system would look like, but you also can feed in your own observational data and calculate a best-fit model to those observations. You can load your observational data by clicking the menu item File→Open data file. Again, if you compiled and installed Nightfall following the instructions above, you will have several sample data files that you can load. You probably should click on the menu item File→Clear memory first. Once the data is loaded, select the Data Fitting tab in the main window. You then need to select the Mass and/or Separation buttons as parameters for the fitting. Then,

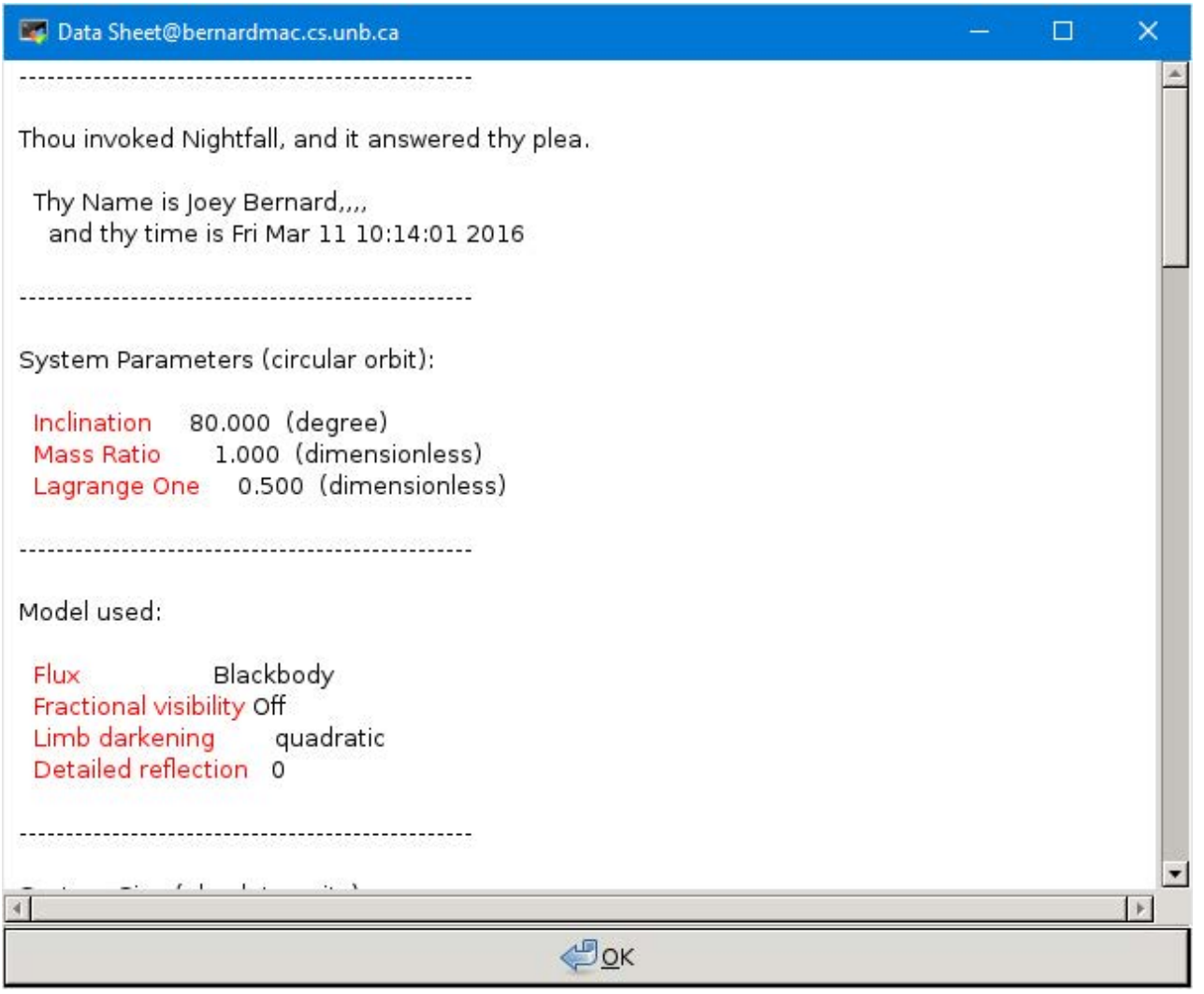


Figure 6. The DataSheet gives you the detailed numerical results of your simulation.

select the FIT with tolerance button, using the tolerance you enter within the text box. You also can choose whether to use simulated annealing or Monte Carlo methods. You may want to do some research to see how active astronomers use these parameters to find best-fit models.

The last thing to notice is that almost everything you can do with the GUI is also doable with command-line options. This means you can generate many different models with different sets of parameters and generate the relevant plots and output data automatically. This way, you can farm out the work to some cluster of machines (but that's moving into the realm of "professional" astronomy and beyond the scope of this article).—Joey Bernard

Non-Linux FOSS: Screenshotting for Fun and Profit!

I do a lot of my day-job work on a Windows computer. Part of this involves taking screenshots for training purposes. For years, I've used the built-in "Snipping Tool" that comes with Windows, but I've always hated it.

The other day, I happened across an open-source tool for Windows that launches with a tap of the "Print Screen" key, and it offers a flexible, easy to use screenshot process. You can save the screenshot, but my favorite part is that it can be dumped directly into a program like Paint.NET (my favorite Windows graphics editor, and it's also open source).

If you are on Windows and wish you had a simple way to take a screenshot, check out Greenshot today. It's open source and works amazingly well: http://getgreenshot.org.—Shawn Powers

PREVIOUS
UpFront

NEXT
Reuven M. Lerner's
At the Forge

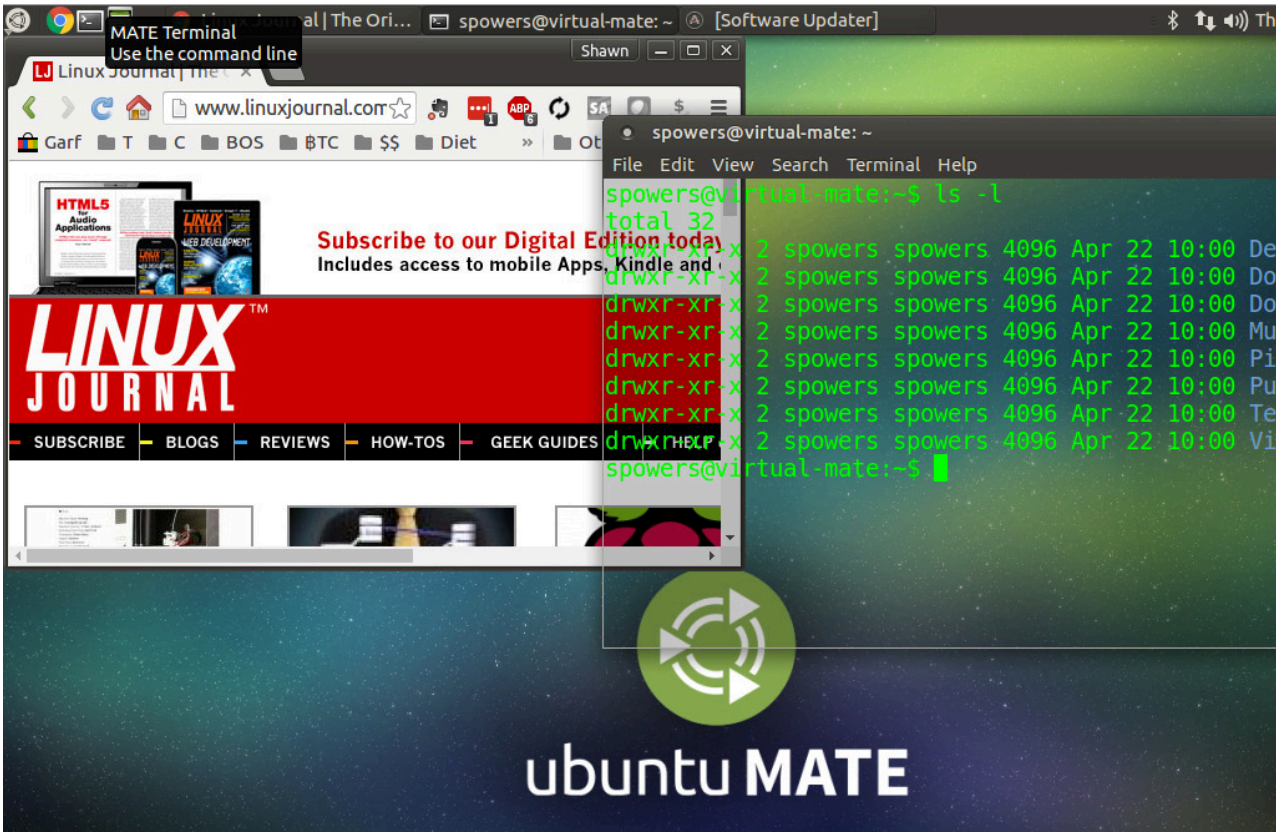
Ubuntu MATE, Not Just a Whim

I've stated for years how much I dislike Ubuntu's Unity interface. Yes, it's become more polished through the years, but it's just not an interface that thinks the same way I do. That's likely because I'm old and inflexible, but nevertheless, I've done everything I could to avoid using Unity, which usually means switching to Xubuntu. I actually really like Xubuntu, and the Xfce interface is close enough to the GNOME 2 look, that I hardly miss the way my laptop used to look before Unity.

I wasn't alone in my disdain for Ubuntu's flagship desktop manager switch, and many folks either switched to Xubuntu or moved to another Debian/Ubuntu-based distro like Linux Mint. The MATE desktop started as a hack, in fact, because GNOME 3 and Unity were such drastic changes. I never really got into MATE, however, because I thought it was going to be nothing more than a hack and eventually would be unusable due to old GNOME 2 libraries phasing out and so forth.

I was wrong.

I'm very happy I was wrong, and with the advent of Ubuntu 16.04 LTS, I decided Ubuntu MATE (now officially in the Ubuntu family) was here to stay. It was the first version of 16.04 that I installed, and I've



never looked back. Running Ubuntu MATE is like living in an alternate reality where GNOME 3 and Unity were never invented. Imagine if GNOME 2 was still king, and you could tweak your desktop to look exactly like it always looked when using GNOME. It's glorious.

Xubuntu, I still love you. You got me through a rough patch, and I'll forever be grateful. But I'm now an Ubuntu MATE man, and I have no plans of changing. And, if MATE 3.0 suddenly changes everything good about desktop computing again, I'll dust of my Xubuntu installer, and apologize for ever leaving!

Due to its bringing back of the GNOME 2 interface in a stable, updated way, Ubuntu MATE 16.04 gets this month's Editors' Choice award (and also my sincere thanks). You MATE folks are awesome.

—Shawn Powers

RETURN TO CONTENTS

nginx

You've probably heard about the nginx HTTP server, but have you tried it? It's easier than you think, and worth a look.

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

ENGINEERS LOVE TO THINK THAT THEY MAKE DECISIONS BASED ON PURE LOGIC AND MERIT. But of course, everyone has biases in terms of programming languages, editors and other technologies—biases that probably can be defended in technical terms, but that often come down to an emotional argument as much as a technical one. (Except in the case of Emacs, of course, which is *clearly* the best editor by all objective standards.) The problem with such biases is that they can cause people to make choices and decisions that feel comfortable, but aren't necessarily right.

Case in point: I've been using the Apache HTTP server for many years now. Indeed, you could say that I've been using Apache since before it was even called "Apache"—what started as the original NCSA HTTP server, and then the patched server that some enterprising open-source developers distributed, and finally the Apache Foundation-backed open-source colossus that everyone recognizes, and even relies on,

today—doing much more than just producing HTTP servers.

Apache's genius was its modularity. You could, with minimal effort, configure Apache to use a custom configuration of modules. If you wanted to have a full-featured server with tons of debugging and diagnostics, you could do that. If you wanted to have high-level languages, such as Perl and Tcl, embedded inside your server for high-speed Web applications, you could do that. If you needed the ability to match, analyze and rewrite every part of an HTTP transaction, you could do that, with `mod_rewrite`. And of course, there were third-party modules as well.

Things got even better through the years as the Web got larger, and Web sites were expected to do more and more. Scalability became an important issue, and Apache handled it with (not surprisingly) a variety of modules that implemented different back-end schemes. You could have the traditional mix of processes, or use threads, or combinations of the two.

Beyond the flexibility, it was clear that Apache httpd was well maintained, well documented and stable. Installation was easy, upgrades were easy—really, everything was easy.

So, it's no surprise that Apache always has been my first choice when it comes to HTTP servers. And yet, I always knew in the back of my mind that I really should spend more time checking out other options. In particular, one alternative stood out—nginx.

Whereas Apache was primarily designed to be modular, nginx was designed to be fast—really fast. Moreover, it was designed to be fast when dealing with large numbers of simultaneous requests. This is thanks to its approach to networking, which is diametrically opposite to Apache's. Apache httpd allocates one new process per incoming HTTP connection. Thus, if there currently are 1,000 simultaneous connections to your Web site, there will be 1,000 Apache processes running on your computer. If you're using multiple threads, you can expect to have 1,000 separate threads servicing those 1,000 requests.

nginx takes the opposite approach, using a single process and no threads. This means that in nginx, those 1,000 simultaneous connections would be handled by one process, rotating through each of those connections to see if there is data to be sent or received. This "reactor" pattern of designing network software has become popular lately, with node.js and event-driven additions to Python 3.5 demonstrating the

interest in this way of writing code.

So yes, nginx is fast. And it's even modular, although the modules cannot be added dynamically, as in the case of Apache. Rather, they must be compiled into nginx in order to use them. For this reason, adding and removing features from nginx, although certainly possible, is less flexible than is the case with Apache, which doesn't require recompilation.

In this article, I go through the basic installation and configuration of nginx to get a simple Web application running. In so doing, you'll see how the configuration differs from Apache, both in style and in execution, and how you need to think if you're going to use nginx.

Installation

Years ago, if you wanted to install nearly any open-source software, you needed to download a .tar.gz file, open it, modify the configuration, compile it and install it. Today, of course, you can install things on a Linux box running Debian or Ubuntu with a simple `apt-get` command. For example, I can install nginx as follows:

```
apt-get install nginx
```

But, wait a second. If nginx cannot be modified after I compile it, perhaps I should check to see how I can modify the configuration I'll get from the default installation. And of course, while you can change the server configuration, you cannot change the modules that are compiled into the server. So making sure that the right modules are compiled into nginx is pretty important before installing it.

On the Ubuntu 14.04 server I used for testing, running `apt-cache search nginx` revealed the following options:

- nginx-extras
- nginx-full
- nginx-light

Which one is appropriate for you, or should you try something else? The

answer, of course, depends on what you want to do.

If you want to serve static files, any of these will do just fine. Even nginx-light, the smallest of the bunch, has features like SSL, gzip and rewriting built in to it. Indeed, nginx-light even includes fastcgi, the module you'll need if you want to run a program like WordPress.

But, let's say you want to deploy Ruby on Rails applications, using the Phusion Passenger add-on. Which version of nginx should you install to run that? The answer, quite simply, is "none of them". nginx will need to be recompiled in order to install Passenger. This is, oddly enough, not as painful as you might expect. However, it does mean that before you even can decide how to install nginx, you need to consider what you want to do with it.

Static Pages

Let's start exploring nginx by installing the nginx-lite package under Ubuntu, then looking at the configuration and how you can get a basic static site running.

First, I'm going to install the nginx-core package:

```
$ sudo apt-get install nginx-core
```

I then can start the server with the fairly standard shell command:

```
$ sudo service nginx start
```

After a few moments, nginx will have started, as I can tell either by typing this:

```
$ sudo serviced nginx status
```

to which I get the response:

```
nginx is running
```

And if I go to the home page on my current server, I'm greeted by, "Welcome to nginx!"

But of course, I'd really like to have my own content there. Let's take

a look at the configuration file, which is in /etc/nginx/nginx.conf on my system, and see how it's formatted and how to change it to make some custom static content.

Now, if you're used to Apache configuration files, the style of nginx's file is going to take some getting used to. Like Apache, each line contains a configuration setting in a name-value style. Unlike Apache, the sections are delimited using curly braces ({ }), and each line must end with a semicolon (;). For example, the first line in my installed, default nginx configuration file is:

```
user www-data;
```

This means nginx will run as the www-data user, which is pretty standard in the world of Ubuntu (and Debian). Next comes the configuration parameter:

```
worker_processes 4;
```

This describes how many processes nginx should launch when running. But, it would seem to contradict what I wrote above, namely that nginx uses only a single process (and no threads within that process) for extra speed, no? Well, yes and no—the idea is that you'll probably want to have one nginx worker process per CPU core on your server. On this server, I have four cores, each of which can (and should) have an nginx worker process. You can think of this as a one-computer version of a load balancer, distributing the load across the available CPUs. Each worker process can and will handle a large number of network connections.

If your server will be running more than just nginx—for example, if you are running a database server on the same machine—you likely will want to reduce this number, so that at least one core is always available for those other processes.

The default configuration file then contains an "events" section:

```
events {
        worker_connections 768;
        # multi_accept on;
}
```

In this, I set `worker_connections`—meaning, how many network connections can each worker process handle simultaneously? In this case, it's set to 768; I'm not sure where this number comes from, but it means that if my site becomes popular, I might find that I run out of network connections. You might well want to raise this number.

The `multi_accept` directive, which is commented out by default, is also set to "on" by default—meaning that nginx is willing to accept new connections as they arrive, handling more than one at a time. I can't think of a good reason to turn this off.

Next is an "http" section, which you won't be surprised to hear has to do with HTTP connections made to the system.

Most of these configuration directives aren't going to be of interest right away; as you can see, nginx's logging directives are similar to those in Apache and other servers:

```
access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;
```

Where is the location of the site defined? In the case of nginx, it's not directly within the "http" block. Rather, it's inside another configuration file—or more accurately, a set of configuration files for the sites configured on the server:

```
include /etc/nginx/sites-enabled/*;
```

Because I'm using a fresh installation of nginx on a computer that hasn't been used for other things yet, there is only a single server configured. You easily can imagine a situation in which a single computer is configured to work with dozens, or even hundreds, of different sites, each of which will have its own configuration file. In this case, however, I'll just work with the "default" server, defined here:

```
/etc/nginx/sites-enabled/default
```

This file starts with a "server" section, describing a single port on which nginx should be listening. This means if you want to listen on multiple

ports—for example, on port 80 for HTTP and port 443 for HTTPS—you'll need to configure those in separate blocks. This "server" block opens with the following:

```
listen 80 default_server;
```

This means that it's going to be listening to port 80, and that this is the default server for the system. Consider a computer on which nginx is running, which is hosting several dozen sites using virtual hosts. Using `default_server`, you can tell nginx which site will accept requests for names that aren't otherwise claimed by another virtual host.

Finally, here are the two lines that tell nginx where to look for my files:

```
root /usr/share/nginx/html;
index index.html index.htm;
```

The `root` directive tells nginx in which directory to look. And the `index` directive indicates that if someone asks for the directory—in this case, the simple URL "/"—which file should be served.

So, I know that to modify my (current, default) static Web site, I need to edit the file /usr/share/nginx/html/index.html. And sure enough, if I look in that location on my server's filesystem, I see the "Welcome to nginx" file. By changing that file, I can change what my site looks like.

Using PHP

However, if I want to use a server-side language, I'm out of luck. As currently configured, nginx won't let me use PHP or anything else. If I simply rename the file to index.php and add a line of PHP inside of it:

```
<?php echo '<p>Hello World</p>'; ?>
```

then at best, I'll get the source file downloaded to my browser, without any execution of the PHP code. At worst, things will just fail.

So, let's figure this out a bit. First, if I'm going to use PHP, I'll need

to install the language on my server. Note that installing the entire php5 package in Ubuntu then tries to install Apache as well, which is clearly not the goal here! Thus, I'll just install a few selected packages:

```
$ sudo apt-get install php5-cli php5-fpm
```

What's php5-fpm? That's for "FastCGI", a standard that was established many years ago in order to cut down on the overhead of CGI (that is, external) programs that Web servers would run in order to create customized, dynamic pages. Rather than starting the external program once for each HTTP request, I'll start it only once, executing the already-started program each time an HTTP request comes in. I'll thus need to set up PHP to work with the FastCGI protocol.

This is done using a server, which you'll need to install and configure. The idea is that nginx will receive a request for a file containing PHP; it'll invoke PHP using FastCGI and then will return the program's output to the user's browser.

There are several ways to set up the FastCGI server. I used UNIX sockets, which allow two programs to communicate if they're both on the same server. You could instead use network sockets, in which case the FastCGI server could exist on a different computer from the nginx server, but for the example here, that's overkill.

In order for this to work, I'll need to modify the configuration for PHP's FastCGI implementation. The change that I made was in the file /etc/php5/fpm/pool.d/www.conf, which came with my PHP configuration. In this file, there is a (commented-out) line with the `listen` value. I set it to use a UNIX socket, as follows:

```
listen = /var/run/php5-fpm.sock
```

Once I had done that, I restarted the FastCGI server for PHP:

```
sudo service php5-fpm restart
```

That restarted PHP's FastCGI-compliant server, making it possible for nginx to talk to the server.

Connecting nginx to PHP

With that in place, I just need to tell nginx when to invoke the FastCGI server and how it can contact that server.

First, I changed the `index` line to look for the file index.php, by replacing the previous `index` line:

```
location / {
      index index.php;
  }
```

Now, when an HTTP request comes in for a directory, it'll serve up index.php.

Next, I needed to tell nginx that when it sees a file ending with a ".php" suffix to use FastCGI:

```
location ~ \.php$ {
    try_files $uri =404;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass   unix:/var/run/php5-fpm.sock;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME
 ➥/usr/share/nginx/html$fastcgi_script_name;
    }
```

The two most important lines here are `fastcgi_pass`, which must point to the socket file I've created, and `fastcgi_param`, which indicates where the FastCGI programs are to be located. In the above `fastcgi_param` directive, I'm indicating that files with a ".php" suffix in /usr/share/nginx/html will be executed in the right place.

Notice also the `include` line, which imports a huge number of directives having to do with FastCGI into the system. You can take a look at it, if you want, but I've been using FastCGI for many years and tend to treat many of the configuration options as something approaching black magic.

What's Next?

Now that you've seen that you can configure nginx with PHP, you

can go in any of several directions. First, you could use PHP not only to create simple "hello, world" programs, but also to run real applications, such as those based on WordPress (which is written in PHP). Next month, I'll describe how you can connect nginx to WordPress for a robust and high-speed solution.

But, nginx can be used with languages other than PHP as well. Phusion Passenger, which I have discussed in the past, works not only with Apache, but also with nginx. The only issue is that because nginx must be recompiled when you add or remove (or update) a module, the installation can be a bit tricky.

The bottom line is that nginx, although it takes some getting used to for an old Apache user like me, turns out to be flexible, well documented and (of course) extremely efficient at handling Web traffic. If you're setting up a new Web server and think you might need to squeeze some more "oomph" out of your system, it's definitely worth looking into nginx.■

RESOURCES

nginx is a popular server, and as such, there are lots of sources for information about it. One of the best such sources is http://nginx.com, the official site of nginx run by the company that has been founded to develop and support it. From that site, you can read a great deal of high-quality documentation, including a Wiki (https://www.nginx.com/resources/wiki/start) with many user-submitted suggestions.

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

RETURN TO CONTENTS

Polishing the wegrep Wrapper Script

Dave finishes his "grep" wrapper, adding all sorts of useful functionality to this improved `grep` command-line tool.

DAVE TAYLOR

Dave Taylor has been hacking shell scripts since the dawn of the computer era. Well, not really, but still, 30 years is a long time! He's the author of the popular *Wicked Cool Shell Scripts* and *Teach Yourself Unix in 24 Hours* (new edition just released!). He can be found on Twitter as @DaveTaylor and at his tech site: http://www.AskDaveTaylor.com.

PREVIOUS
Reuven M. Lerner's
At the Forge

NEXT
Kyle Rankin's
Hack and /

WHEN LAST I DISCUSSED SHELL SCRIPTS, I was presenting a shell script that offered an alternative to the -C context flag in GNU `grep`. Although most modern Linux systems have the more capable `grep` command, older systems likely don't have this particular feature, and it's also a good excuse to dig into working with wrapper scripts too.

"Wait. What's a wrapper script?" I can hear you ask, and some of you also are now trying to think of a famous rapper whose name you can reference for a punny response. I've already beat you there: "Can't touch that!"

A wrapper is a script that replaces a command on

the Linux system but secretly calls the command, just offering more and better capabilities and features. When you have an alias set up so that every invocation of `ls` is really `ls -F`, that's the same basic idea.

Linux and its grizzled father UNIX are really powerful because they offer these sorts of capabilities; it's hard to write a wrapper for Microsoft Excel on a Windows 10 system, by contrast.

A command with multiple versions in the wild is a perfect example of where a wrapper can be so beneficial too. Imagine you're deploying a few hundred servers and want to run a bare-bones Linux on them to maximize available cycles. Problem is, your admin scripts rely on the very latest-and-greatest versions of `sed`, `grep` and `find`. Solution? Point the scripts at your wrapper versions of those commands, and make sure every flag you need is implemented, either in the base command (as would be the case on the newer systems) or through the wrapper code itself.

So, back to wegrep. When last I left this script, it offered up the base -C functionality of giving one or more lines of context before and after each match to a `grep` search. Left on the to-do list were to make it smarter about when to add the "- - - - - -"divider line, to add line numbers and to highlight the actual match.

Let's start with making the script smarter with the divider line, because that's by far the easiest. Like any script that tries to separate multiple blocks of output neatly, the key is really to count how many times the output has been sent. Here's the solution:

```
if [ $matches -eq 0 ] ; then
  echo "-----"
fi
matches=$(( $matches + 1 ))
```

This appears prior to each block of output. The very first time it produces the top divider line, and otherwise it's skipped. After the matching line or lines, however, there's another divider line that is included each and every time.

Adding line numbers can be accomplished a number of ways, but I'm going to exploit an interesting capability of the `sed` command itself,

the "=" expression. Let me demonstrate with the wonderland.txt data file that contains the first couple paragraphs of *Alice in Wonderland*:

```
$ head -5 wonderland.txt | sed =
1
------------------------------------------------------
2

3
ALICE'S ADVENTURES IN WONDERLAND
4

5
Lewis Carroll
```

You can see what it does, I hope? It adds line numbers, but by having the number actually show up on a line prior to the actual matching line. It's a bit funky, but a second `sed` invocation fixes the problem and gives output that makes a lot more sense:

```
$ head -5 wonderland.txt | sed = | sed 'N;s/\n/:   /'
1:   ------------------------------------------------
2:
3:   ALICE'S ADVENTURES IN WONDERLAND
4:
5:   Lewis Carroll
```

In the above, the replacement sequence is a colon followed by the Tab character itself, which can be entered by typing Ctrl-V followed by the Tab itself—easily done in scripts.

So, that's two down: a smarter divider line and the ability to number the output lines. Let's see how that works:

```
$ sh wegrep.sh '^Alice' wonderland.txt
-----
12:
```

```
13:  ^Alice was beginning to get very tired of sitting by
14:  her sister on the bank, and of having nothing to do:
-----
27:  There was nothing so very remarkable in that; nor did
28:  ^Alice think it so very much out of the way to hear the
29:  Rabbit say to itself, 'Oh dear! Oh dear! I shall be
-----
```

The dividers work perfectly, showing up the minimum amount needed to denote each matching block of lines clearly, and the line numbers are neat and helpful.

The trickier part is still left to tackle. How do you actually highlight the match in each section?

ANSI Color Sequences

You may not realize it, but odds are incredibly high that your Terminal or xterm window, whether you're directly in a Linux system or connecting via a Windows or Mac computer, is emulating what's known as an ANSI terminal.

ANSI is the American National Standards Institute, but don't be misled; this is a global standard, particularly when it comes to colors, bold and other visual aspects to the terminal.

The problem is, the sequences to turn on and turn off bold or specific colors has to be fairly obscure to ensure that users don't accidentally end up invoking it. So "color:" would be a fail, as would "<color>". Instead, it's done through an escape sequence: Escape + [+ 3 + 2 + m causes all subsequent text to be rendered as green, for example.

The Escape + [sequence prefix has a name of its own. It's a Control Sequence Introducer, although you probably don't need to know that! You can find a full table of ANSI color sequences on-line, of course: https://en.wikipedia.org/wiki/ANSI_escape_code.

Once you're done with the highlighted text, you'll need to change the display back to regular text, and that's done with the sequence Escape + [+ 0 + m.

Add them all up, and here's what you use to highlight whatever value is

stored as $1 in a string:

```
\033[32m$1\033[0m
```

The `\033` is a shorthand for Escape. Rather than make this an echo statement, it's a good use of `printf`, so here's the sequence:

```
sed ''/$1/s//`printf "\033[32m$1\033[0m"`/'' "$2"
```

This basically replaces every occurrence of $1 with itself, prefixed with the ANSI green sequence and suffixed with the sequence to return subsequent text to its normal display characteristics.

I'm being a bit lazy here by exploiting how the script works too. If it can show matching lines from a file, it also can show matching lines that have had the ANSI sequences slipped in. So here's the new flow, and it's a bit more complicated than my original stab at this script:

```
sed ''/$1/s//`printf "\033[32m$1\033[0m"`/'' "$2" | \
sed = | sed 'N;s/\n/:  /' | \
sed -n "${before},${after}p"
```

Four invocations of sed in a row—ah, I love Linux!

In the above, the first sed invocation adds the ANSI sequences, the second and third work together to add the line number prefixes, and the fourth shows the lines in the stream from the range `$before` to `$after`.

To see how those are calculated, here's the full script:

```
#!/bin/sh
# wegrep - grep with context and regular expressions
grep=/usr/bin/grep
sed=/usr/bin/sed
context=1
matches=0
if [ $# -ne 2 ] ; then
  echo "Usage: wegrep [pattern] filename" ; exit 1
fi
```

```
for match in $($grep -n -E "$1" "$2" | cut -d: -f1)
do
  before=$(( $match - $context ))
   after=$(( $match + $context ))
  if [ $matches -eq 0 ] ; then
    echo "-----"
  fi
  sed ''/$1/s//`printf "\033[32m$1\033[0m"`/'' "$2" | \
    sed = | sed 'N;s/\n/:       /' | \
    sed -n "${before},${after}p"
  echo "-----"
  matches=$(( $matches + 1 ))
done
exit 0
```

It's surprisingly short given how useful this wrapper script is and how

many new features have been added to an older, crude `grep` program.

And, here it is in use:

```
$ sh wegrep.sh 'Alice' wonderland.txt
-----
12:
13:  Alice was beginning to get very tired of sitting by her
14:  sister on the bank, and of having nothing to do: once
-----
16:  reading, but it had no pictures or conversations in it,
17:  'and what is the use of a book,' thought Alice 'without
18:  pictures or conversation?'
-----
27:  There was nothing so very remarkable in that; nor did
28:  Alice think it so very much out of the way to hear the
29:  Rabbit say to itself, 'Oh dear! Oh dear! I shall be
-----
```

There's still a hiccup in the script, however. Because of the ANSI sequence sed invocation, the proper functionality of regular expressions is lost (try it, you'll see what I mean). Is it a huge problem? Maybe not, but I'm going to leave solving it as an exercise for you, the reader.

Next month, I'll dive into something new. If you have suggestions, let me know via e-mail: dave@linuxjournal.com.■

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

RETURN TO CONTENTS

Secure Desktops with Qubes: Compartments

Figuring out how to compartmentalize your desktop across VMs can be daunting, so I've provided an example of how I do it to help you get started.

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 IS THE THIRD ARTICLES IN MY SERIES ABOUT QUBES. In the first two articles, I gave an overview about what Qubes is and described how to install it. One of the defining security features of Qubes is how it lets you compartmentalize your different desktop activities into separate VMs. The idea behind security by compartmentalization is that if one of your VMs is compromised, the damage is limited to just that VM.

When you first start using Qubes, you may not be quite sure how best to divide up all of your files and activities into separate VMs. I know when I first started using it, I found inspiration in Joanna Rutkowska's (Qubes' creator) paper on how she used Qubes (http://invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf). In this article, I describe how I organize my activities into VMs on my personal computer. Although I'm not saying my approach is perfect, and I certainly could secure things even further than I do, I at least will provide you one example you can use to get started.

Summary of Qubes Concepts

In my previous article, I elaborated on overall Qubes concepts like the different VM types, trust levels and other features, but since I refer to those concepts in this article as well, here's a brief summary. (If you want to know more, read my column in the April and May 2016 issues.)

The first concept to understand with Qubes is that it groups VMs into different categories based on their use. Here are the main categories of VMs I refer to in the rest of the article:

- Disposable VM: these also are referred to as dispVMs and are designed for one-time use. All data in them is erased when the application is closed.

- Domain VM: these also often are referred to as appVMs. They are the VMs where most applications are run and where users spend most of their time.

- Service VM: service VMs are split into subcategories of netVMs and proxyVMs. These VMs typically run in the background and provide your appVMs with services (usually network access).

- Template VM: other VMs get their root filesystem template from a Template VM, and once you shut the appVM off, any changes you may have made to that root filesystem are erased (only changes in /rw, /usr/local and /home persist). Generally, Template VMs are left powered off unless you are installing or updating software.

When you create new VMs of any type, you can assign them a color based on your level of trust on a continuum from red (untrusted) to orange and yellow to green (somewhat more trusted) to blue and purple and grey (even more trusted) to black (ultimately trusted). The window borders and icons for a particular VM are colorized based on their trust level, so you get visual cues that help prevent you from, for instance, pasting trusted passwords into an untrusted VM.

Although by default all new Qubes VMs you create have unlimited network access, Qubes allows you to create firewall rules to restrict what a VM can do. If your VM doesn't need network access (such as for the highly-trusted vault VM you can use to store GPG keys and password vaults), you even can remove the network device completely.

In a default install, Qubes provides a few appVMs to help you get started:

- untrusted appVM: red.
- personal appVM: yellow.
- work appVM: green.
- vault appVM: black.

The idea is for you to perform any general-purpose untrusted activities (like general Web browsing) in the untrusted VM and not store any personal files there. Then you can perform more trusted activities like checking your e-mail or any Web browsing that requires personal credentials in the personal VM. You can check your work e-mail and store your work documents in the work VM. Finally, you can store your GPG keys and password manager files in the vault (which has no network at all). Although this is nice for getting started, as you can see, you may want to isolate your activities and files even further.

The installer also creates a sys-net, sys-firewall and sys-whonix service VM to provide you with network access, a firewall for appVMs and a Tor gateway, respectively. You also optionally can enable a sys-usb service VM that is assigned all of your USB controllers to protect the rest of the

system from USB-based attacks.

My Personal Computer

My personal computer is a Purism Librem 13, and my general desktop use is pretty basic. Here's my normal list of activities in order of risk:

- Web browsing.
- Checking e-mail.
- Chatting on IRC.
- Using my 3D printer.
- Writing articles.

Generally speaking, Web browsing and e-mail are the riskiest activities I perform on my computer each day, as they can expose me to malicious file attachments and other compromises. On the other end, all I need to write articles is a text editor with no network access, so that's a pretty safe activity. Below I list the different appVMs I've created based on this type of use, ordered from least-trusted to most-trusted. I also show what color I assigned the VM and describe how I use each appVM.

dispVM—red: I use disposable VMs whenever I'm doing something particularly risky, such as when I want to view a sketchy-looking URL. For instance, my mail client is configured to open all attachments automatically in a disposable VM (based on the official Qubes mutt guide: https://www.qubes-os.org/doc/mutt). That way, even if someone were to send me a malicious Word document or PDF, I can read it in the disposable VM, and the attack is isolated inside that VM. When I close the document, any malicious program it is running goes away and in the meantime, the attacker had no access to any of my personal files.

untrusted—red: My untrusted appVM is where I perform all of my general-purpose Web browsing but not any Web sites that require a user name and password. It has unrestricted access to the Internet. I've set up some other more-trusted VMs (such as the one where I chat in IRC)

In this way, any tracking cookies are limited to the Web browser inside this appVM; Tor prevents any other servers apart from Facebook from knowing I'm using Facebook, and even Facebook itself doesn't know my IP.

to open up URLs in this VM automatically (by setting the default Web browser in that appVM to be the qvm-open-in-vm command-line tool). I don't store any personal files in my untrusted VM, so if I feel like a URL I opened looks particularly sketchy, I can just delete the VM and re-create it, and in less than a minute, I'm back with a clean untrusted VM.

Since I browse random Web sites with this VM and might open obscured URL-shortened URLs in it, it's one of the VMs most likely to be compromised. That said, because I don't store any personal files in the VM, and I don't browse to any Web sites that require a user name and password, the most an attacker could do besides just use that VM for its network and CPU resources is view my general browsing habits.

fb—orange: It may surprise some readers to know that I have a Facebook account. I personally don't post to my account all that much (and when I do, I post only things I'm fine with the whole world seeing), but like many of you, I have friends that I don't see often who post about what's going on with their lives only on Facebook. I'm concerned about the privacy issues surrounding Facebook tracking my every move on the Web, but I still want to be able to view my friends' posts, which I can't do without logging in.

My compromise has been to create a special appVM just for Facebook and nothing else. This appVM is configured to use the sys-whonix proxyVM for network access, so all of its traffic goes over Tor, and I use Facebook's Tor hidden service at https://facebookcorewwwi.onion to access the site. In this way, any tracking cookies are limited to the Web browser inside this appVM; Tor prevents any other servers apart from Facebook from knowing I'm using Facebook, and even Facebook itself doesn't know my IP.

personal-web—yellow: Because Web browsing is one of the riskier activities one can perform, I've decided to separate my authenticated Web browsing not only from my general Web browsing, but also from the rest of my personal files. Since sites that provide a login also usually let you log in over HTTPS, I restrict this VM's network access so it can connect only to port 443 on the Internet. The personal-web appVM is set aside for any site that needs a user name and password (outside of banking). So, for instance, when I'm shopping on-line, I might use my untrusted VM, but when I'm ready to log in and buy something, I use my personal-web appVM. I've set up my password vault to open URLs in this appVM automatically instead of the untrusted one.

The idea here is to prevent an attacker who has compromised my untrusted appVM through a malicious Web site from being able to grab any of my Web credentials. Although it's true that attackers who compromise one of the many Web sites I log in to through personal-web would be able to get credentials for other sites, they still wouldn't be able to access any of my personal files (like documents or GPG or SSH keys). They also wouldn't be able to access my banking, because I separate my banking into its own VM. Some Qubes users who are concerned about this sort of thing end up launching disposable VMs for any authenticated sessions.

personal—yellow: My personal appVM is the closest to a traditional user's home directory, and it contains the bulk of my personal files, such as my SSH keys. That said, I don't do any Web browsing from this appVM and use either the untrusted, personal-web or finance VMs for that. I mostly use this VM to `ssh` to other servers I manage, check e-mail, connect to a remote screen session that I use for IRC and manage a few personal GitHub projects. Because of this, I can restrict the Qubes firewall so it allows only outbound SSH to any remote IP, and otherwise to open only the handful of ports IMAP and SMTP need explicitly to my mail server.

Because this VM contains the bulk of my personal files and my (password-protected) SSH key, I'm more careful about what I do in this VM than in some of the others. That's a big reason why I don't browse the Web from this VM and why although I check e-mail from this VM, I automatically open all attachments in a disposable VM.

Qubes provides a service known as split-GPG that acts like a GPG wrapper program you can use in other appVMs whenever you want to access a GPG key inside a vault.

printrbot—green: I decided to set aside a special appVM just for interacting with my printrbot, because I have a shared internal network volume I mount that stores all of my 3D models. This VM controls my 3D printer and also launches the Cura program I use to modify and slice 3D models. By splitting this off to its own appVM, I can use the Qubes firewall to restrict access to just my local Octoprint server and my network storage, since this VM has no need to use the Internet at large.

finance—green: Because of how sensitive a person's financial accounts are compared to other accounts, I've decided to create a special appVM that's normally off, that I use only for banking. That way, my banking credentials won't leak to other appVMs. This VM is locked down so that it can connect only to port 443 on the Internet at large, and if I wanted to lock it down further, I could restrict that to only the hostnames belonging to my banking sites.

writing—blue: This VM is the VM I'm using right now, because it is where I store all of my personal documents and organize my writing. I decided to isolate this activity from others partially because I don't want to risk leaking books I'm working on that are unreleased, but also because I realized this VM really needs no network apart from access to a local git repository I use to organize a few writing projects, so I can create really restrictive firewall rules.

vault—black: The vault is the most sensitive and most trusted appVM in my environment. Instead of just creating restrictive firewall rules for this host, to make it as secure as possible, it has no network device at all. I use this VM to store my GPG key and my KeepassX password vault. Qubes provides a service known as split-GPG that acts like a GPG wrapper program you can use in other appVMs whenever

you want to access a GPG key inside a vault. Basically, an appVM that wants access to the key sends its encryption or decryption payload to the vault VM using the Qubes wrapper script. You get a colorized prompt on your desktop asking whether you want to allow the appVM to have access to the vault's GPG key for a set period of time. If you accept, the payload goes to the vault, the vault encrypts or decrypts it, and then the output goes back to the appVM. In that way, your appVM never sees the GPG private key in the vault, and it behaves kind of like a poor-man's Hardware Security Module.

I hope that seeing how I organize my Qubes desktop will help you figure out how best to organize your own VMs. Generally speaking, it comes down to separating different types of activities and files from each other based on risk. The guiding principle is to assume it's possible to break into a particular VM (in particular, less-trusted VMs) and to try to limit the files and access any certain VM has. If you want some more specific guides on Qubes best practices, the first place to start is Qubes' own documentation page (https://www.qubes-os.org/doc) as it provides a number of useful guides for common activities (like setting up split-GPG).■

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

RETURN TO CONTENTS

Build Your Own Raspberry Pi Camera

A high-resolution wireless IP camera for less than $100? You bet!

PREVIOUS
Kyle Rankin's Hack and /

NEXT
New Products

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 e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

DURING THE PAST FEW YEARS, my BirdCam setup has evolved significantly. As I mention in the UpFront section of this issue, I hope to get the stream transferred to a YouTube Live stream at some point, so I can watch the feathery show on my television. And although watching the birds is the end goal, I'm constantly on a mission to improve the quality and flexibility of my setup.

Right now, the "best" quality video comes from Logitech 720p cameras that connect to my motion

server via USB. (You can check out the setup in previous articles—just search for "birdcam" on http://www.linuxjournal.com, and you'll find lots of information.) I've purchased several IP cameras, but each one has its limitations and frustrations. For one, the quality of an IP video camera is just not as sharp as a photo from a local device—at least not without spending literally thousands of dollars for a high-quality IP camera from Axis. I figured there must be an easier way, and with the dawning of the Raspberry Pi 3 era, I think I found it.

Why Pi?

The RPi3 has built-in Wi-Fi. That means I can access it without worrying about Ethernet cabling. And, that means I can potentially put it outside without trying to weatherproof a network cable or figure out how to poke a hole in my house to get the Ethernet inside!

The RPi3 has an incredible camera module. Literally the day after I purchased my 5 megapixel camera module, RPi released an 8 megapixel camera, which means if you buy now, you'll get even better results.

The RPi3 is small. That's important, because I plan to put everything inside a weatherproof project case and mount it outside next to a bird feeder that isn't right outside my window (they all are now by necessity).

The RPi3 is fairly inexpensive, especially when compared to an IP camera with similar image quality. The RPi3 itself is about $35, the camera is $25, and I have a pile of MicroSD cards and MicroUSB chargers lying around. Even with the cost of the project box (around $15), it's all less than $80.

The Goal

Since the Raspberry Pi device I'll be setting up is a full-blown Linux computer, the configuration options are endless. It's possible to install Motion on the little beastie and handle motion detection fully on the RPi. I already have Motion installed on my Birdcam server though, so what I want is for the Raspberry Pi simply to serve out a stream that my existing server can use to capture movement like it does with the USB cams locally connected.

My first attempt at creating the perfect RPi IP camera (RPIPCam?) included compiling mjpg_streamer and creating an MJPEG streaming

camera that could be added to the Motion setup on my BirdCam server. I realized after the fact that Motion would be just as happy with a simple Web server on the RPi serving up a still image, then constantly re-downloading that image. It means my Raspbian Linux image doesn't have to have any custom software installed at all, which is ideal.

The Process

The first step is to install Raspbian. This is done easily by getting the NOOBS zip file from https://www.raspberrypi.org/downloads/noobs and unzipping it onto your freshly formatted MicroSD card. Don't let the insulting name fool you; NOOBS is incredible. It allows you to install a variety of distributions, and it does all the heavy lifting. I can't recommend it enough.

Once you get the file unzipped onto your MicroSD card, connect the RPi Camera Module via ribbon cable, an HDMI monitor, USB mouse and USB keyboard. It's possible to install without all that, but it's much nicer to use a keyboard/mouse/monitor while installing. You won't need them later, but at first, save yourself a lot of hassle and set it up like you would a desktop.

During the setup process, you'll be asked what OS you want to install. Don't bother setting up the network yet, because although a working network will allow you to install other operating systems (like the awesome OpenELEC), it won't save your Wi-Fi settings, and you'll have to set it up again anyway. Raspbian will be the only option, which is what you want.

Once Raspbian is installed, you'll reboot the system, and it should come up into a GUI desktop. Thankfully, if you set up Wi-Fi now with the GUI tool, it will save the configuration for you and automatically connect even after you turn off the GUI. While you're still in the GUI, open a terminal window and figure out your MAC address so you can configure DHCP to give the RPi the same address every time. At the very least, type `ifconfig` and see what your IP address is so you can connect to the device over the network later.

Once you know your network information, type `sudo raspi-config` to start the Raspbian configuration tool. Inside the text menu system, you

want to do four things:

1. Enable the camera. It's a menu option. You'll simply select "enable" in the menu, and that should be it.

2. Change the "pi" user's password.

3. In the "boot options" menu, choose the console option requiring login.

4. In "advanced options", make sure the SSH server is enabled.

When you're finished, type `sudo reboot` and wait for the RPi to boot back up. Make sure the GUI doesn't start, and make sure you can `ssh` in to the RPi from another computer. Once you're sure it's working, you can disconnect the monitor, keyboard and mouse.

Post-Install Setup

The only software I installed on my RPi, in addition to what came by default, is the lighttpd package. It's a very fast, simple Web server. Since all I need to do is serve out an image via HTTP, it's perfect. So, if you're following along, `ssh` in to the Raspberry Pi and type:

```
sudo apt-get update
sudo apt-get install lighttpd
```

The next step is to start taking photos and serving them out, but before doing that, check to make sure the camera module is working. On the terminal, type:

```
vcgencmd get_camera
```

You should see something like this:

```
supported=1 detected=1
```

If not, run `raspi-config` again and make sure the camera

module is enabled. You may need to reboot. It should work out of the box without any additional software. I spent a *long* time trying to troubleshoot a non-working camera, and it turned out to be a bad camera. With a working unit installed properly, with the ribbon cable in tight, the camera worked right away. Double-check the connections, but if it appears to be connected and just won't work, perhaps you have a bad camera. I doubt bad cameras are common, but since I purchased two cameras and one was broken, my track record shows they have a 50% failure rate. (Never do statistics with small sample sizes!)

Once the camera reports that it is working, you can start taking photos. The built-in programs are really the best (possibly only) way to do this. Run the `raspistill` command and have it take a new photo every 100ms, overwriting the previous photo each time. Here's the command:

```
raspistill --nopreview -w 1280 -h 720 -q 80 -o /dev/shm/pic.jpg
 ➥-tl 100 -t 0 -th none
```

And, this is what the flags do:

- `--nopreview`: You aren't using a GUI, so there's no point in trying to generate a preview image to see on the monitor.
- `-w`: width of the snapshot, in pixels.
- `-h`: height of the snapshot, in pixels (the camera can do much more than 720p, but that's the size I want for my BirdCam).
- `-q`: quality 0–100; the bigger the number, the higher the quality, but also the bigger the filesize. Experiment and find your happy compromise. Keep in mind the limitation of Wi-Fi speeds.
- `-o`: Where to save the file. Since you want it to overwrite, this is a static location. I always put it in /dev/shm/, because that's the ramdisk on the machine, and it won't wear out the SD card with constant writing.

- `-tl`: time between snapshots. I set this to 100ms so the "video" results in about 10fps. You might need to go slower depending on resolution and quality. Just watch the output for dropped frames.

- `-t`: how long to keep capturing before stopping. This defaults to 5 seconds, which does no good. Setting it to 0 seems to make it never timeout.

- `-th`: the size and quality settings of the thumbnail image. You don't want a thumbnail here, so "none" simply makes sure none are created.

The only other configuration to make is to link the image file so that it can be seen remotely via HTTP. Since you installed lighttpd earlier, just type:

```
sudo ln -s /dev/shm/pic.jpg /var/www/html/pic.jpg
```

Then see if it's working by opening a browser window and heading over to http://raspberry.pi.ip.address/pic.jpg, and you should see a still image. If you refresh the browser, you should see a new image. Move the camera a bit to make sure when you click refresh, you're seeing a new image. Note that the image won't refresh automatically in your browser yet, but if you manually refresh, you should see the updated image.

If it's working as expected, go back to the SSH terminal and press ^C to stop the `raspistill` process. If you want it to start automatically, I recommend typing `crontab -e` and then making an entry like this in the crontab file:

```
@reboot raspistill --nopreview -w 1280 -h 720 -q 80 -o
 ➥/dev/shm/pic.jpg -tl 100 -t 0 -th none > /dev/null 2>&1
```

That will start the process on boot and send all the output to /dev/null so you don't get constant e-mail messages from the cron dæmon.

The Mockup

I love mock-ups. In fact, on about half my projects, I never really go beyond the mockup stage. Quite frankly, I'm just impatient and want to make something work quickly! So my first implementation of the RPi IP Camera looks like Figure 1. I propped the box against my office window and was able to get a pretty decent image (Figure 2). Keep in mind that the image isn't even close to the maximum image quality the RPi camera can manage, and it's really quite awesome. One frustration I have is that the RPi camera module has "infinite focus", so it can't be adjusted for macro shots. If you look at Figure 3, you'll see the difference. That's with my Logitech USB camera, which supports manual focus. I like the sharp image with the fuzzy background a bit more. Still, the images from both are very high quality. And, since 1280x720 is fairly low resolution for the RPi camera, I'm very happy.

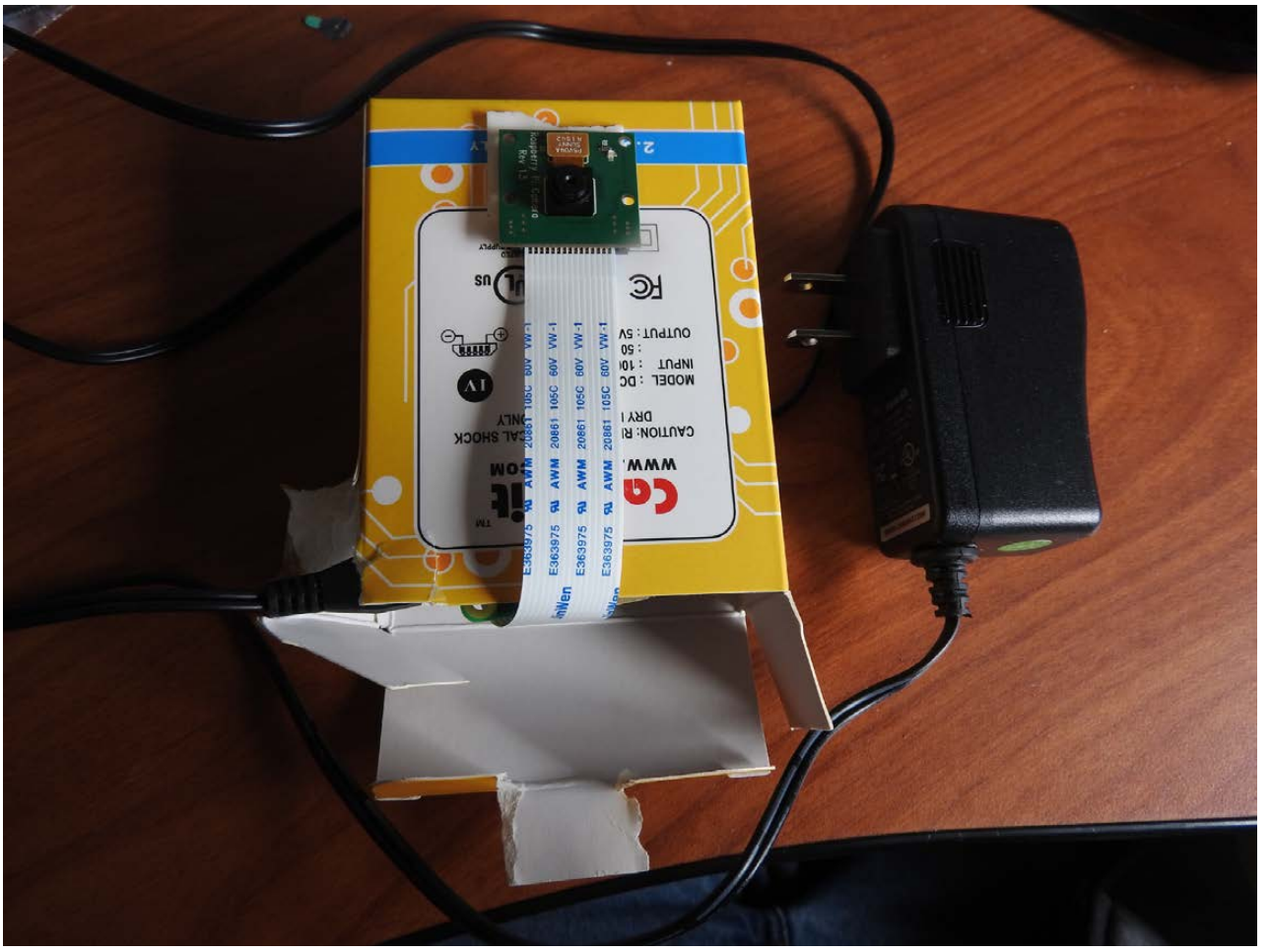


Figure 1. My "mock-ups" usually deserve to be mocked.

Figure 2. This is the hummingbird feeder as recorded via the Raspberry Pi camera.

Figure 3. This is the hummingbird feeder as recorded via the Logitech USB Webcam.

Integration with BirdCam

Remember, since the Raspberry Pi is a complete computer, you don't need to implement Motion on another system. In my case, the Motion install is

on another computer, but yours doesn't have to be.

I won't go into the setup process for Motion itself, because that's covered in depth in my past BirdCam articles. What I will show you, however, is how simple it is to configure a new thread.conf file for the new camera you just created. In /etc/motion/, I created a new file, thread4.conf, which I included in the main /etc/motion/motion.conf file. Here's what thread4.conf looks like:

```
framerate 10
output_normal on
quality 80

webcam_port 8084
webcam_quality 75
webcam_maxrate 10
webcam_localhost off
webcam_maxrate
webcam_limit 0

netcam_url http://rpi.ip.address/pic.jpg
netcam_tolerant_check on
```

There are lots of other things you can configure with Motion, like capturing motion, movies and so on. This simple configuration file, however, turns the simple .jpg file served by the RPi into an MJPEG stream, which can be viewed on port 8084. So if you load up http://birdcam.ip.address:8084, you'll see a full-motion MJPEG video stream!

Final Touches

My next step is to put the RPi IP Camera into my weatherproof project box. I have the box (Figure 4) and plan to do the following for my final product:

1. Cut a hole in the plastic box for the camera to see through, then cover the hole with a piece of glass from an old photo frame. I'll use silicone sealant to make sure the hole is waterproof, and I'll mount the camera with double-sided tape or possibly hot glue. I'll also cover the bright-red

LED on the camera to avoid glare.

2. Experiment with mounting lenses from cheap reading glasses over the camera. I'd like to mount the RPi IP Camera very close to a bird feeder, so I might need to use the reading lens to make a low-tech macro lens for the camera.

3. Secure the MicroUSB transformer inside the project box so that during the winter, the transformer's warmth will keep the RPi from freezing. I'm not sure of the operating temperature for the camera unit, but the transformer should keep the box warm enough.

4. Drill and seal a spot for an extension cord to come out of the project box. I want it to be airtight, so more silicone sealant will be involved.

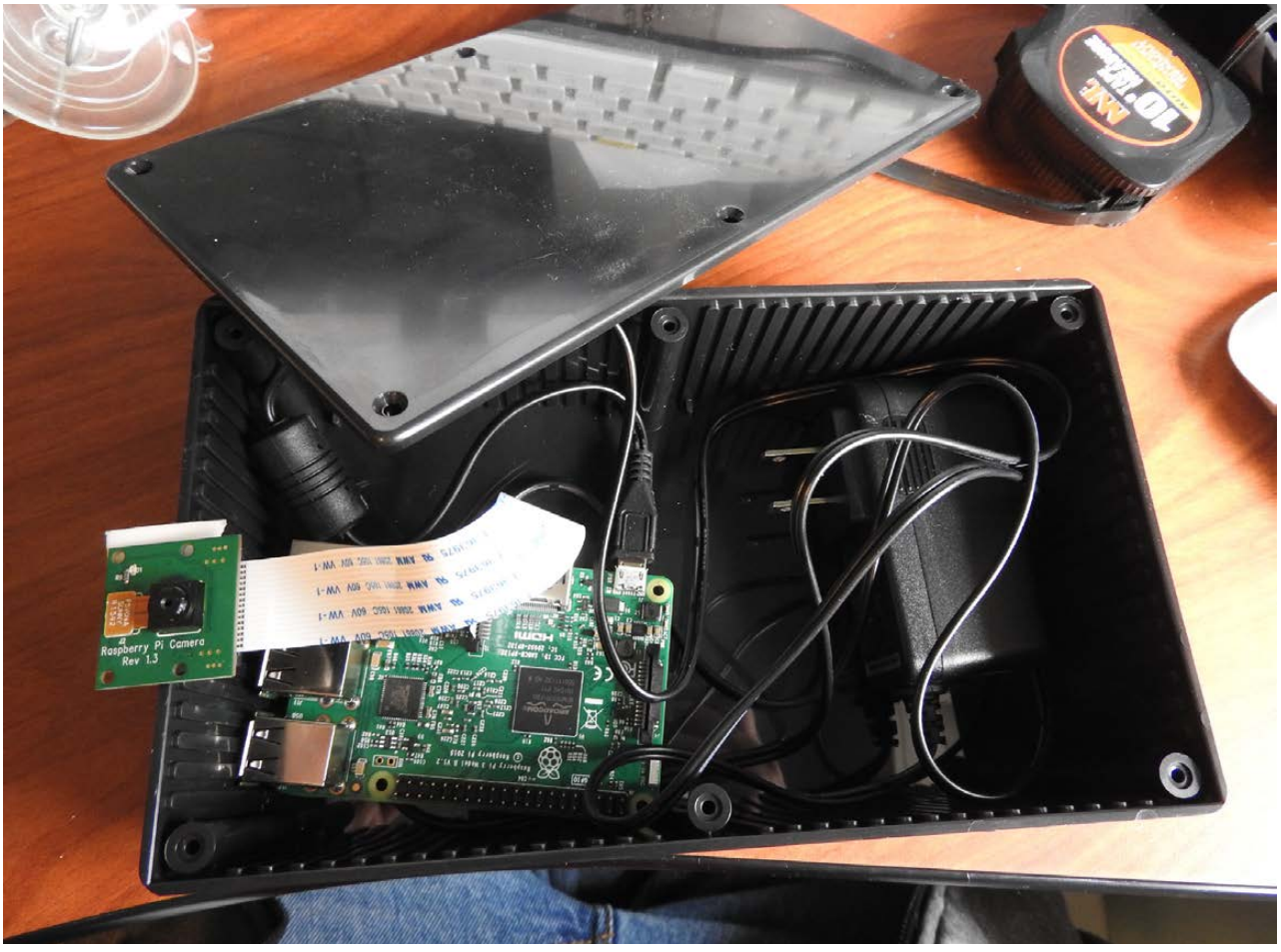


Figure 4. The project box was $15 on Amazon and will have plenty of room for everything.

The Future?

Honestly, I really want to make the camera solar-powered. I could mount a solar panel on the roof of a bird feeder, and then use that to power a lithium-ion battery to run the RPi. The biggest problem is that Raspberry Pi computers tend to be very sensitive to voltage changes and reboot easily. My concern is that the charging/powering circuitry is beyond my ken right now. I'd ideally like to get a solar cell powerful enough to charge a battery that will keep the RPi running all night. But, that's a project for another day!

If you do any interesting projects inspired by or similar to my BirdCam, I'd love to hear about them. In the meantime, keep an eye on http://birds.brainofshawn.com, because I plan to make lots of enhancements this summer!■

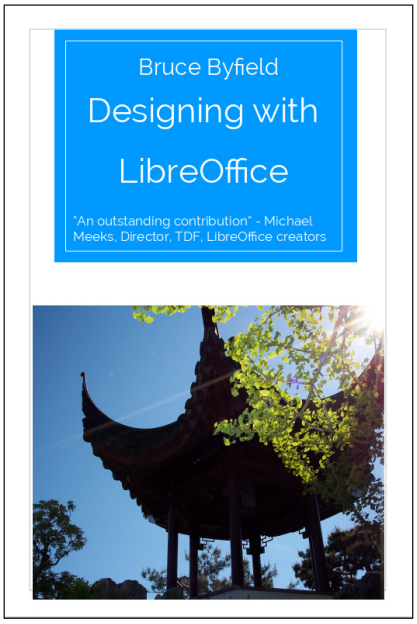
Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

RETURN TO CONTENTS

NEW PRODUCTS

PREVIOUS
Shawn Powers' The Open-Source Classroom

NEXT
Feature: Let's Automate Let's Encrypt



Bruce Byfield's *Designing with LibreOffice*

Tech author and *Linux Journal* writer Bruce Byfield's new book *Designing with LibreOffice* is not the usual death march through the menu and standard tasks. Rather, Byfield's book takes two fresh approaches to the world's most popular free office suite. First, Byfield outlines the importance of using styles and templates in order to utilize LibreOffice with the most convenience and least effort. This approach lets users concentrate on self-expression, rather than formatting. Second, Byfield explains the basics of modern design and how to apply them in LibreOffice, illuminating the open secret that LibreOffice is as much a desktop publishing tool as an office suite. Byfield explains and illustrates the range of design choices as well as the pros, cons and considerations behind each choice. *Designing with LibreOffice* was released under a Creative Commons Attribution-ShareAlike license and can be downloaded electronically from the book's Web site or ordered on paper at http://www.lulu.com.
http://designingwithlibreoffice.com

Contrast Security's Contrast Enterprise

With more and more businesses running on the Node.js server-side JavaScript runtime environment, application vulnerabilities are a growing threat to entire organizations. The antidote to this problem, says Contrast Security, is the new Contrast Enterprise, which is marketed as the only application security product that enables the discovery and remediation of Node.js security vulnerabilities in real time. Contrast Enterprise achieves this without disrupting software development processes or involving application security experts. Product features include high levels of accuracy so that developers don't burn cycles chasing false alarms; continuous operation throughout the Agile development process—that is, no security scans or waiting for results; and deep security instrumentation for identifying vulnerabilities across Node.js deployments, such as APIs, microservices, containers and libraries.
http://www.contrastsecurity.com

SoftMaker FreeOffice

The bottom line on SoftMaker FreeOffice 2016—the updated, free, full-featured Office alternative to the expensive Microsoft Office suite—is this: no other free office suite offers as high a level of file compatibility with Word, Excel and PowerPoint. This maxim applies to both Windows and Linux operating systems, says the suite's maker, SoftMaker Software GmbH. SoftMaker asserts that the myriad competing free alternatives often harbor problems opening the Excel, Word and PowerPoint file formats loss-free. Sometimes the layout and formatting get lost, and on other occasions, files cannot even be opened. SoftMaker sees itself as the positive exception to this rule, especially with the newly overhauled FreeOffice 2016. Benefiting greatly from SoftMaker's commercial offering, SoftMaker Office 2016, FreeOffice 2016 adds features such as improved graphics rendering, compatibility with all current Linux distributions and Windows flavors (XP to Windows 10), new EPUB export and improved PDF export and many other MS-Office interoperability enhancements.

http://freeoffice.com

Microstar Laboratories, Inc.'s Accel64 for Linux

Microstar Laboratories, Inc., develops Data Acquisition Processor (DAP) systems for PC-based high-performance multichannel measurement applications. Microstar observes that GNU/Linux distributions generally presume that if you have 64-bit hardware and a 64-bit operating system, the applications will use compatible 64-bit development tools and drivers, leaving support for 32-bit applications incomplete. This can present problems for vetted 32-bit applications, particularly those dependent on kernel extensions. To deal with the problem, Microstar's latest innovation in the DAP space is Accel64 for Linux software, version 1.00. Accel64 allows advanced DAP applications to be supported as 32-bit applications on 32-bit or 64-bit hardware platforms, or as 64-bit applications on 64-bit hardware platforms. Delegation of complex real-time details to the DAP/DAPL systems means that data acquisition applications can use generic kernels and graphical desktop environments, even on lightweight platforms without 64-bit support. This new software is available for free download.
http://mstarlabs.com

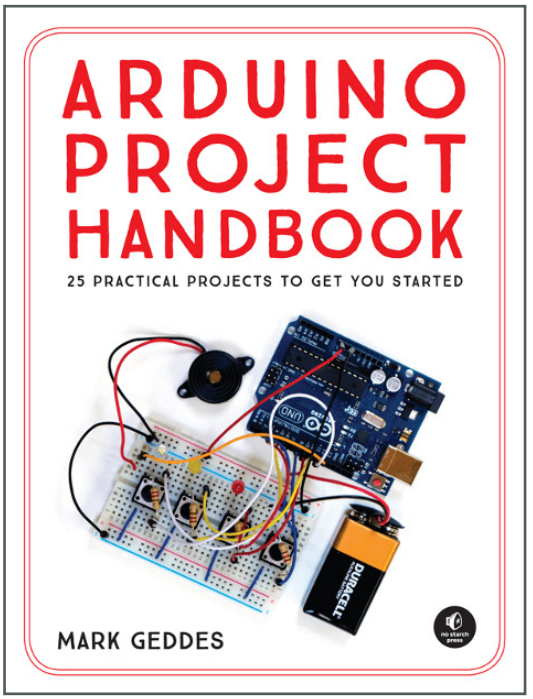
Apricorn's Aegis Secure Key 3.0 USB Drives

Packing a mighty punch in a tiny package is the Apricorn's Aegis Secure Key 3.0 line of software-free, hardware-encrypted USB drives, which recently added a 480GB version. Apricorn claims that the new Flash key is "roughly four times the competition's max size of 120GB" and "the ideal tool for corporate data security deployment". Apricorn boasts that Aegis Secure Keys are completely software-free, cross-platform-compatible with any OS and have embedded authentication, meaning that no security parameters ever are shared with the host. They further carry the highest portable device security validation that the NIST grants: FIPS 140-2 level 3, and feature separate admin and user modes, two read-only modes, forced enrollment, programmable brute-force defense, a lock-override mode and the ability to adhere to most security policies. Apricorn's Aegis Secure Key 3.0 is available in 8GB, 16GB, 30GB, 60GB, 120GB, 240GB and 480GB capacities.
http://apricorn.com

Susan Lauber's *Linux Command Line Complete Video Course* (Prentice Hall)

Users, developers and administrators can all find much to love in Linux's powerful command line. Those seeking to go deeper into the Linux command-line interface—from new users of the Linux command line to system administrators with limited command-line experience or developers more comfortable with an IDE—will find just what they need in technical instructor and trainer Susan Lauber's new Prentice Hall LiveLessons *Linux Command Line Complete Video Course*. The 6+ hours of video training introduce common utilities used at the Linux command line. While learning commands for specific tasks, users will obtain greater confidence navigating the Linux filesystem, understand how to locate and edit files, use Bash shell features for efficiency and automation and access built-in help for further exploration.
http://informit.com

Mark Geddes' *Arduino Project Handbook* (No Starch Press)



Lifelong tinkerer and gadget enthusiast Mark Geddes was so frustrated with the lack of practical, visual Arduino guides for teaching his ten-year-old that he wrote his own book on the topic. Titled *Arduino Project Handbook: 25 Practical Projects to Get You Started*, Geddes' book is a beginner-friendly collection of 25 fun and interactive projects to build with the low-cost Arduino microcontroller. Projects range from disco strobe lights and joystick lasers to rocket launchers and laser tripwires. This is a step-by-step project book, suitable for total beginners just starting out as well as for more experienced makers looking for inspiration. Readers will get set up with introductions on the hardware and software along with advice on tools, components and workspaces. Then it's time to choose a project to build from scratch, using straight-forward instructions, color illustrations, simple circuit diagrams and the complete code to program the build. *The Arduino Project Handbook* is a fast and fun way to get started.

http://nostarch.com

Dynamsoft's Barcode Reader SDK

What's slick about Dynamsoft's Barcode Reader SDK is that just a few lines of code from scratch are required instead of potentially hundreds of them, which could save months of development time. The updated Barcode Reader SDK 4.2, which adds Linux PHP support, allows application developers to embed functionality for decoding linear and 2D barcodes into Web or desktop applications almost instantly, significantly reducing development costs and costs of long-term application support. Barcode recognition is enabled from image files, scanned images and from images captured on Webcams or smartphones. The new PHP barcode reader toolkit for Linux supports PHP x64 versions 5.3–5.6. Both Thread Safe and Non Thread Safe options are provided. The Dynamsoft toolkit works with Debian, Ubuntu and CentOS. Support for Web applications in ASP.NET, in C# or VB.NET, and PHP on Windows also is included, just in case.
http://dynamsoft.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

Let's Automate Let's Encrypt

No more reasons to handle certificates manually—make your server do your work.

ANDREI LUKOVENKO

PREVIOUS
New Products

NEXT
Feature:
How We R on Android

HTTPS is a small island of security in this insecure world, and in this day and age, there is absolutely no reason not to have it on every Web site you host. Up until last year, there was just a single last excuse: purchasing certificates was kind of pricey. That probably was not a big deal for enterprises; however, if you routinely host a dozen Web sites, each with multiple subdomains, and have to pay for each certificate out of your own dear pocket—well, that quickly could become a burden.

Now you have no more excuses. Enter Let's Encrypt (https://letsencrypt.org), a free Certificate Authority that officially left Beta status in April 2016.

Aside from being totally free, there is another special thing about Let's Encrypt certificates: they don't last long. Currently all certificates issued by Let's Encrypt are valid for only 90 days, and

If, like me, you prefer nginx and want to have zero-downtime automatic certificate updates with industrial-grade encryption, keep reading.

you should expect that someday this term will become even shorter (https://letsencrypt.org/2015/11/09/why-90-days.html). Although this short lifespan definitely creates a much higher level of security, many people consider it as an inconvenience, and I've seen people going back from using Let's Encrypt to buying certificates from commercial certificate authorities for this very reason.

Of course, if you are running multiple Web sites, having to renew several certificates manually every three months quickly could become annoying to say the least. Some day you even may forget (and you will regret that forgetfulness). Let's leave routines to computers, right?

If you are using Apache under a Debian-based distribution, Let's Encrypt already has you covered with the libaugeas0 package, and it is capable of both issuing and renewing certificates. If, like me, you prefer nginx and want to have zero-downtime automatic certificate updates with industrial-grade encryption, keep reading. I'm going to show you how to get there.

First things first—some assumptions and requirements:

1. You are running the nginx (https://www.nginx.com) Web server/load balancer, and you are going to use it for TLS termination (that's a fancy, but technically correct way of saying "nginx will handle all this HTTPS stuff").

2. nginx serves several Web sites, and you want HTTPS on all of them, and you are not going to pay a single dime.

3. You also want to get the highest grade on the industry standard for SSL tests—SSL Lab's SSL server test (https://www.ssllabs.com/ssltest).

4. You do not enjoy the idea of running some not-so-well-sandboxed third-party code on your server, and you would rather have this code in a Docker container.

5. Naturally, you are lazy (or experienced) enough, so you want to write some scripts that will re-issue all certificates way before they expire.

6. I tested this code on Debian Jessie running nginx 1.6.2 and Docker 1.9.1; it also should work on all other flavors. If you do not have docker-engine installed, follow the instructions here: https://docs.docker.com/engine/installation.

Now, check whether your nginx supports TLS:

```
sudo nginx -V
```

Usually it is supported by default and should yield the following:

```
TLS SNI support enabled
```

You also need a place to store certificates:

```
sudo mkdir -m 755 /etc/letsencrypt
```

Don't sweat the permissions for this directory; the certificates themselves will not be publicly accessible. Now you need to make a small change in your nginx configuration. Create a new file /etc/nginx/letsencrypt.inc with the following contents:

```
location ^~ /.well-known/acme-challenge/ {
    root /tmp/letsencrypt/www;
    break;
}
```

Then find your "server" section in the nginx configuration, and add the following line to each Web site you host:

```
include /etc/nginx/letsencrypt.inc;
```

So the final result will look like this:

```
server {
    listen 80;
    server_name example.com www.example.com;
    ...
        include /etc/nginx/letsencrypt.inc;
        ...
}
```

After saving both files, ask nginx to reload the configuration:

```
sudo /usr/sbin/nginx -t && sudo service nginx reload
```

Notice that you are only reloading the nginx configuration—and nginx knows very well how to do it without dropping connections.

Now, let's go get some certificates! Needless to say, all domain names for which you are going to issue certificates should resolve to your server IP address; otherwise, it would be possible to issue certificates for somebody else's domain and use those certificates for man-in-the-middle attacks.

The following will pull and start a new Docker image with the official Let's Encrypt client:

```
mkdir -p /tmp/letsencrypt/www

# make sure you have the latest version of this image,
# and not some pre-beta - those used to be notoriously buggy
docker pull quay.io/letsencrypt/letsencrypt:latest

docker run --rm -it --name letsencrypt \
-v /etc/letsencrypt:/etc/letsencrypt \
-v /tmp/letsencrypt/www:/var/www \
    quay.io/letsencrypt/letsencrypt:latest \
    auth --authenticator webroot \
    --webroot-path /var/www \
    --domain=example.com --domain=www.example.com \
    --email=admin@example.com
```

As you can see, you share two data volumes between the host and the container:

- /etc/letsencrypt for storing Let's Encrypt configuration, all certificates and chains.

- /tmp/letsencrypt/www for communication between your server with Let's Encrypt servers.

The webroot plugin that runs inside the container will create a temporary challenge file for each of your domains, then Let's Encrypt validation servers will send an HTTP request to ensure that you are really controlling this domain and this server. These files are temporary and needed only during issuing or renewing a certificate.

You will need to agree on TOS by pressing a button, and after several seconds, your certificate is ready. If you have several subdomains, as in this example, you can enumerate all of them, which will result in one shared certificate issued for all of these

subdomains. However, if you have several domains, it would be much more convenient to have a separate certificate for each of them—just repeat this last `docker run ...` command for each domain you have (and thank me later if someday you decide to move one of your domains to a different server).

As you can see, the procedure for obtaining certificates is painless and safe. Almost all the heavy work is done for you behind the scenes, and if you've ever had to deal with certificates using some other traditional certification authority, you will know exactly what I mean. Whatever runs inside the container can access only two directories on the server, and only while it runs.

After you get all the certificates, it's safe to remove the

As you can see, the procedure for obtaining certificates is painless and safe. Almost all the heavy work is done for you behind the scenes, and if you've ever had to deal with certificates using some other traditional certification authority, you will know exactly what I mean.

temporary directory:

```
rm -rf /tmp/letsencrypt
```

Let's go back to the nginx configuration. Getting an A+ grade from SSLLabs requires some additional effort. Create a new Ephemeral Diffie-Hellman prime (if this is the first time you've see this term, see https://wiki.openssl.org/index.php/Diffie_Hellman for more information):

```
sudo openssl dhparam -out /etc/pki/tls/private/dhparam.pem 4096
```

Caution: if you absolutely need to support ancient versions of client software, for example, Java 6 clients, you need to skip this step and

comment the `ssl_dhparam` line in the following step. These old clients do not support Diffie-Hellman parameters longer than 1024 bytes, so you need to make a choice between supporting those clients and security.

Now, have a hot beverage; it will take some time to generate. Add these lines to the "http" section of /etc/nginx/nginx.conf:

```
http {
        ...
        ssl_dhparam /etc/pki/tls/private/dhparam.pem;
        ssl_session_cache shared:SSL:10m;
        ssl_session_timeout 60m;
        ...
}
```

Create a new file /etc/nginx/ssl_options.inc:

```
ssl on;
ssl_prefer_server_ciphers on;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers "ECDH+AESGCM DH+AESGCM ECDH+AES256 DH+AES256
     ➥ECDH+AES128 DH+AES ECDH+3DES DH+3DES RSA+AESGCM
     ➥RSA+AES RSA+3DES !aNULL !MD5 !DSS";
# Enable HSTS (HTTP Strict Transport Security) for half a year
add_header Strict-Transport-Security
 ➥"max-age=15768000;includeSubDomains";
```

And create a new "server" section:

```
server {
    listen 443;
    server_name example.com www.example.com;

    include /etc/nginx/letsencrypt.inc;
include /etc/nginx/ssl_options.inc;
```

```
ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

        # enable OCSP stapling to speed up first connect
    ssl_stapling on;
    ssl_stapling_verify on;
    ssl_trusted_certificate
     ➥/etc/letsencrypt/live/example.com/chain.pem;

...
}
```

Warning: the Strict-Transport-Security header will tell each visitor that you promise always to use HTTPS in the future. It's a one-way street, and once you set it, there is no way back—your visitor's browser will remember your promise and insist on having HTTPS. Also note: for more information on OCSP stapling, see https://en.wikipedia.org/wiki/OCSP_stapling.

After making all of these changes, reload the nginx configuration again:

```
sudo /usr/sbin/nginx -t && sudo service nginx reload
```

At this point, your Web site should have HTTPS up and running. Try to open https://www.example.com/ in a browser and enjoy the green lock sign in the address line. To verify the quality of encryption, go to https://www.ssllabs.com/ssltest, and submit your hostname for a check (usually it takes several minutes).

So, now that you have HTTPS, how about disabling HTTP? Go back to the HTTP "server" section and make the following improvement:

```
server {
    listen 80;
    server_name example.com www.example.com;
        include /etc/nginx/letsencrypt.inc;
        ...
if ($scheme = "http") {
```

```
        rewrite ^/(.*)$ https://$host/$1 permanent;
}
        ...
}
```

This will redirect all traffic from HTTP to HTTPS, automatically bringing all clients to the secure version of your Web site. Reload the nginx configuration to activate the changes.

Now it's time to automate certificate renewals. Let's Encrypt's current policy allows you to request five certificate renewals for a domain within seven days. That means it wouldn't be wise (and wouldn't make much sense either) to try to renew certificates every day. On the other hand, leaving it for the last moment before expiration also is quite dangerous. Luckily, there

To me, 30 days sounds just right. That means my certificates will be reissued every 60 days on average, and if something fails afterward, I will have a whole month to fix whatever is broken.

is an easy way to renew these certificates only when they have less than 30 days before expiration. To me, 30 days sounds just right. That means my certificates will be reissued every 60 days on average, and if something fails afterward, I will have a whole month to fix whatever is broken.

Create a script for renewal (I placed it in/root/update_keys.sh) with these contents:

```
#!/bin/bash

mkdir -p /tmp/letsencrypt/www

ADMIN_EMAIL=admin@example.com
HOSTNAME=$(hostname)
```

```
OUTPUT="$((docker run --rm -i --name letsencrypt \
    -v /etc/letsencrypt:/etc/letsencrypt \
    -v /tmp/letsencrypt/www:/var/www \
    quay.io/letsencrypt/letsencrypt:latest renew) 2>&1)"

if [[ $? -eq 0 ]]; then
    echo "${OUTPUT}" | grep -q "No renewals were attempted"
    if [[ $? -eq 0 ]]; then
        # all certificates have more than 30 days left -
        # nothing to do
        exit 0
    fi
    echo "${OUTPUT}" | tr -Cd '[:print:]\n' \
        | mail -s "${HOSTNAME}: Let's Encrypt keys renewal -
         ➥success" "${ADMIN_EMAIL}"
    else
        echo "${OUTPUT}" | tr -Cd '[:print:]\n' \
            | mail -s "${HOSTNAME}: Let's Encrypt keys renewal -
             ➥failed, exit code $?!" "${ADMIN_EMAIL}"
        exit 1
    fi

    # test config, reload if successful
    /usr/sbin/nginx -t &> /dev/null
    if [[ $? -ne 0 ]]; then
        echo 'please fix configfile problem' \
            | mail -s "${HOSTNAME}: nginx unable to reload"
             ➥"${ADMIN_EMAIL}"
        logger "nginx has errors - not reloaded"
    else
        service nginx reload
        logger "nginx reloaded"
    fi

    rm -rf /tmp/letsencrypt
```

Remember to assign proper access rights:

```
sudo chmod u+x /root/update_keys.sh
```

And create a crontab entry:

```
sudo crontab -e
```

with a line like this:

```
17 2 * * * /root/update_keys.sh
```

That will trigger execution of this update script at 2:17 every day. The update script will check whether your certificates have more than 30 days left, and if they don't, it will attempt to renew all expiring certificates. Are you wondering why I used 2:17 am? Well, there is a simple explanation for that: almost everybody else did not. Most people, when creating cron jobs, use some simple value like 1:00 am, 2:00 am, 3:30 am, 4:15 pm and so on, and that is a really, really bad choice if your cron job is supposed to talk to an external service, because that means the service will experience maximum loads every once in a while. It is bad for the service, and it is not good for you; the chance of getting a timeout is significantly higher if you send a request during these peak loads.

So, for this job, please, please do not use an even value, and do not use my value; use some random value instead, and everything will be fine.

As you can see, Let's Encrypt managed to make the full automation of certificate maintenance possible. If you are using it right, it just works—and it's free.■

Andrei Lukovenko is a longtime Linux user, command-line fanboy, automation aficionado.

How We R on Android

You can run a full version of the R statistical software environment on mobile devices running Android with the help of a Linux operating system installed in a change-root environment.

MARIUS HOFERT and KURT HORNIK

PREVIOUS
Feature:
Let's Automate
Let's Encrypt

NEXT
Doc Searls' EOF

The year 2014 frequently was termed the "year of wearable devices", but besides being able to access e-mail, play games or listen to music, we have yet to see real productivity (including IDEs like RStudio, editors like Emacs or tools like LaTeX and Sweave) on wearable devices like smartphones or even tablets. Although we could not agree more with Bjarne Stroustrup (see the first minute of his YouTube video listed in the Resources for this article), early attempts to bring more productivity to wearables, such as the Ubuntu Edge campaign, so far have failed. We have no doubt that the development of productivity tools for wearable devices will come, but the hype hasn't provided a working solution so far. With that in mind, this article addresses the following question: "How can one run R on Android devices in a productive way?"

For our tests we used the Google Nexus 5 smartphone (released November 2013) with these specs: quad-core 2.3GHz CPU (Snapdragon 800, 32-bit, ARM-based); 32GB memory; 2GB RAM; and Android 5.1.1 (Lollipop). During our testing phase, we also considered Android 4.4 (Kitkat) and Android 5.0 (Lollipop). Note that in practice, people usually would rather work with R on a tablet (or at least a phablet) than on a smartphone. Our choice of such a small device (with a 5" display) is purely to demonstrate the concept and its potential (and because we already owned this device).

The choice of Android as an operating system is natural for these three reasons:

- A majority of smartphones and tablets run Android.
- Android is a mobile operating system based on the (free and open-source) Linux kernel.
- As it turns out, it is feasible to use R on Android (whereas we are not sure how this could be done on iOS, for example).

Other approaches for running R on Android (not discussed in this article) are used by the R Console Free and GNURoot apps (available at the Google

Play Store). The former allows only limited R functionality under Android; the latter follows a similar paradigm as presented here, but provides only console access to R (one of the more obvious limitations). In particular, both methods do not provide GUI access to R (so no graphics).

Our understanding of productivity goes beyond the functionality of these apps. We aim not only at running a full version of R on Android (including graphics or multicore computing), but also at tools that are naturally required to be productive with R—for example, a full version of an editor like Emacs (including ESS), LaTeX and Sweave (for writing articles or working on presentations during flights). For RStudio users, see the Tips and Tricks section at the end of this article.

We also want to consider some selected auxiliary tools from the Linux world. Note that for some of these tools (like Emacs), specific Android apps are available (such as the Emacs app), but they often are in early stages of development and have frequent crashes (not offering the functionality we need), or they are not well maintained, which leads to bugs we don't want to cope with (we've pointed out several to the respective app developers with minor success in solving them).

Also, none of these apps work or communicate with each other (as they are mainly black boxes), and for a fully functioning "R app", Android is missing compilers, a shell and an X Window System (among other tools).

Our approach may be a bit more daunting to install, rather than using a simple thin client and connecting to an R server to address the above questions, but this is what we consider to be part of productivity (one advantage is not being limited to Internet access).

The Main Idea

The goal is to install a Linux operating system on Android and then install the required productivity tools (including R) under this Linux OS. We then can run an SSH or VNC server on it and connect to it via a terminal or graphically, respectively. Here are the steps:

1) Prepare the Android device. This might be the technically most difficult step, and it is partly device-dependent. In particular, it involves unlocking the device's bootloader (required for rooting the device—that is, gaining root/superuser access to the device) and rooting the device (which is required by the Linux Deploy app to install a Linux OS under Android).

2) Installing and accessing the Linux OS. The main idea here is that the Linux OS will be installed in a so-called change-root (chroot) environment on Android 5.1.1—that is, in a directory tree in which a running program cannot name or access files outside the tree. The Linux OS installed this way allows for root access (and therefore to install the required productivity tools), but sees only this local environment and does not interfere with the Android OS.

3) Installing and using R and related productivity tools. Here we show how you can install R and tools like Emacs (version 24) with Emacs Speaks Statistics (ESS) and the LaTeX system TeX Live (2015) with AUCTeX. We also show how you even can do multicore computations, and we provide some tips and tricks addressing productivity when working with Android.

Preparing the Android Device

For this article, we assume you have a laptop computer running an Android Software Development Kit (SDK). We used a MacBook Pro (2015) running OS X Yosemite 10.10.5 and Android Studio as the SDK. During our testing phase with Android 4.4 and 5.0, we used a Lenovo X1 Carbon (2013), which ran Ubuntu 14.10 (the corresponding Android SDK was installed via `sudo apt-get install android-tools-adb android-tools-fastboot`). Installing these tools is straightforward. We also wanted to use the `adb` and `fastboot` tools from the OS X terminal. An easy way to accomplish this is via `bash <(curl https://raw.githubusercontent.com/corbindavenport/nexus-tools/master/install.sh)`.

Unlocking the Bootloader and Rooting (Android 5.1.1)

In short, rooting describes the process of gaining root or superuser access to an Android device, and it's required in order to interact with Android more deeply. Specifically, it's necessary for installing a Linux OS on the device using an app like Linux Deploy (described below). Because rooting typically completely wipes the device, do a complete backup first (rooting also may brick the device, rendering it useless, although this risk nowadays is considered comparably small for newer devices). As a last resort if the rooting procedure fails, you typically can try to flash a factory image (see Resources). A factory image for the Google Nexus 5 is available at https://developers.google.com/android/nexus/images#hammerhead.

The rooting procedure depends on the Android device under consideration (there are many good on-line resources for this). It also may depend on the version number of Android. For example, for a Google Nexus 5 running Android 4.4 and 5.0, we followed the instructions given in the Unlocking the Bootloader and Rooting sidebar. However, Android 5.1.1 requires a different approach (see Resources).

1) Make sure that your device's battery status is at least 80% and that developer mode is enabled (go to Settings→About phone and then tap the build number until developer mode is enabled).

2) Connect the Android device to the laptop via a (micro) USB cable.

3) Download UPDATE-SuperSU-v2.46.zip from http://www.devfiles.co/download/dr7ZBy0w/UPDATE-SuperSU-v2.46.zip.

4) Use the Android File Transfer app to copy the .zip file to the phone's root directory (the folder shown by default when Android File Transfer is opened; in the Android filesystem, this corresponds to /sdcard or /storage/emulated/0).

5) Download http://www.devfiles.co/download/iLJJIo60/recovery-clockwork-touch-6.0.4.5-hammerhead.img and put this file in the ~/Library/Android/sdk/platform-tools folder on the laptop.

6) Boot into fastboot mode—typically by shutting down the phone, then pressing and holding both volume keys and the power key.

7) Execute in a shell on the laptop:

```
cd ~/Library/Android/sdk/platform-tools
fastboot flash recovery recovery-clockwork-touch-6.0.4.5-hammerhead.img
```

8) Use the volume keys to select "Recovery Mode" on the Google Nexus 5, then press the Power key to enter this mode.

9) From the menu that appears, select "install zip" (via the volume keys), and press the Power key to do so. Select "choose zip from /sdcard".

10) Use the volume keys to select the .zip file. Choose "yes" to install it, then wait until you see "Install from sdcard complete".

11) Choose "Go back" from the menu and "reboot system now", then "Yes - Fix root (/system/xbin/su)". After this process finishes, you can unplug the phone.

12) Install the Super Su app and reboot the device. Then, install the Root Checker app, open it and grant superuser access. The app then

indicates whether rooting the device was successful.

Once the device has been rooted successfully, you can continue installing auxiliary apps and setting up the device.

Auxiliary Apps

At this point, you can set up the Android device as desired. For the purposes of this article, we recommend installing the following apps from the Google Play Store:

- Hacker's Keyboard: a full-size keyboard for Android, including meta keys like Tab/Ctrl/Esc. You can select the Hacker's Keyboard as the default keyboard via Settings→Language input→Current Keyboard.

- Linux Deploy: an app for installing a Linux OS on a rooted Android device (note that this app requires the device to be rooted). An alternative may be the Complete Linux Installer app, but it had various rather strange bugs (for example, "Error: Unable to attach image to loop device!", so every launch required rebooting the device) and did not provide support for Android 5.1.1 at the time of this writing.

- JuiceSSH: an easy-to-use SSH client for Android. (You'll use this later to connect to the SSH server running on the Linux OS you install under Android and provide access to R via the terminal.)

- VNC Viewer: a VNC client for Android you'll use later to connect to the VNC server running on the Linux OS and provide access to graphics (the R GUI).

Installing and Accessing Ubuntu in a Change-Root Environment

The goal now is to install a Linux OS (in this case Ubuntu 15.04; during our testing phase, we also considered Debian Testing) in a chroot environment on Android 5.1.1, together with various productivity tools, such as Emacs (version 24) with ESS and TeX Live with AUCTeX. The Linux Deploy app provides a relatively straightforward approach for installing Linux on Android (described below), and updates to Android

5.0 and 5.1.1 also were provided in a reasonable amount of time (and the developer replied to questions).

1) Start Linux Deploy and press the arrow button pointing downward to specify the properties the Linux OS should have.

2) Choose the following:

- Distribution: Ubuntu (uses the lightweight X11 desktop environment LXDE by default).
- Distribution suite: "vivid" (this is Ubuntu version 15.04).
- Installation path: the path where you want the Linux image to reside (for example, /sdcard/linux.img).
- Image size (MB): the size of the Linux image. Due to the many pieces of software you will install for this example, choose 16000MB here.
- Filesystem: ext4.
- User name: an appropriate user name for the Linux OS (in our case, "mhofert").
- Select components: in addition to the components already checked, also check "X server".
- Custom mounts: check it. On startup of the Linux OS, this allows you to access files located in /storage/emulated/0 on the Android device via /mnt/0 on the Linux OS (which is handy when working on the same files from Android and within the Linux OS); see the beginning of the following section for more on this.

Now choose "Install" to install the Linux OS (hereby grant root/superuser rights to Linux Deploy via the Super Su app when prompted).

3) Choose "START" from the main menu of Linux Deploy to start the Linux OS (Figure 1). There is also an "Autostart" option available from the settings so that the Linux OS starts once Android is booted.

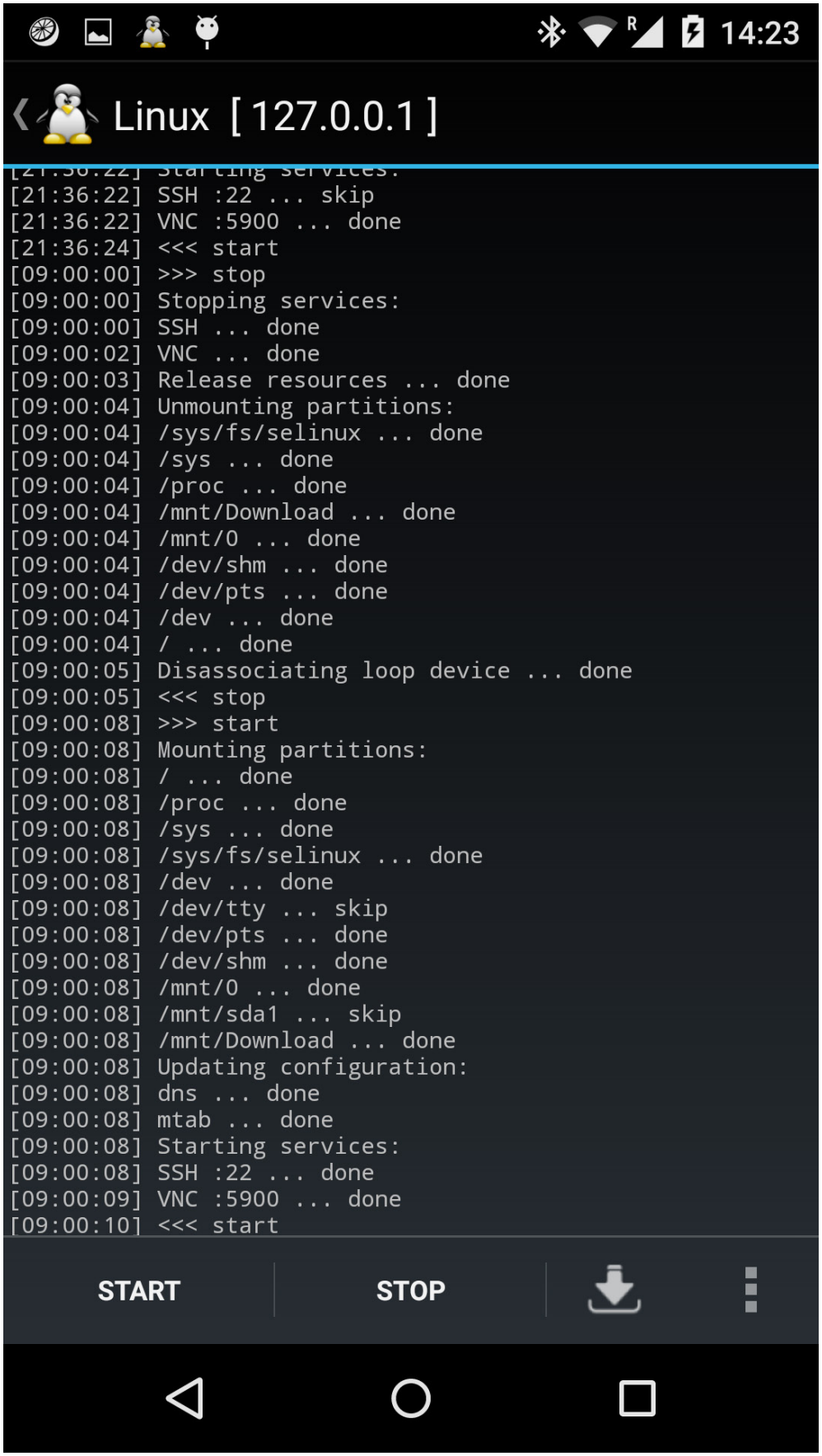


Figure 1. Running Ubuntu 15.04 via Linux Deploy on a Google Nexus 5 Android Smartphone

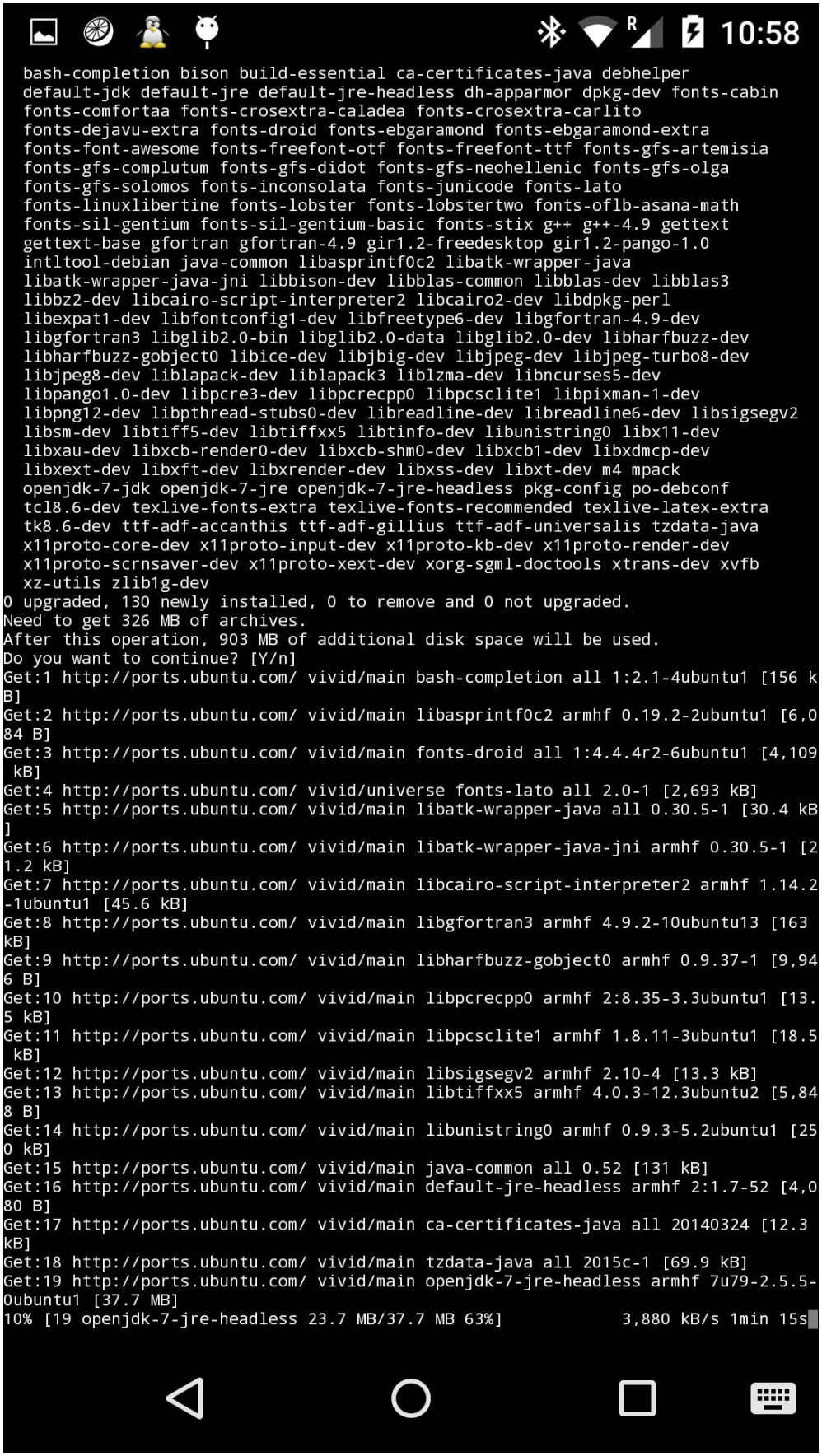


Figure 2. Installing Productivity and Auxiliary Tools via `apt-get` over SSH with JuiceSSH

Starting the Linux OS starts an SSH server (port 22) and a VNC server (port 5900), which allow you to access the Linux OS from Android via the terminal and a GUI, respectively. The following steps detail how to set up an SSH connection via JuiceSSH and then how to set up a VNC connection via VNC Viewer:

1) Open JuiceSSH, navigate to Settings, and set "Popup keyboard position" to "Disabled".

2) Go back to the main menu, then to "Connections", and add a new connection. Choose:

- Nickname: Ubuntu.
- Address: this is the IP address of the SSH server. You can find it on the top bar in Linux Deploy when the Linux OS starts (Figure 1). This may depend on your device's network (in our case, home or work) and whether the IP addresses are allocated dynamically in this network.
- Identity: Choose "New..." and the user name as selected for the Linux OS.

3) Go to the "Connections" main menu and choose "Ubuntu" to start the SSH connection to the running Linux OS. When asked for a password, use "changeme", which is the default for Linux Deploy. You should now be connected to the Linux OS via SSH. Use the Linux `passwd` command to change the login password.

As mentioned earlier, to get GUI access, including graphics for R, you need to establish a VNC connection. Using VNC Viewer, follow these steps:

1) Open VNC Viewer, press the + and enter:

- Address: <IP address as displayed by Linux Deploy>
- Name: Ubuntu

Then press the check mark to save the connection details.

2) In the main menu, press "Ubuntu" and put in the password "changeme".

Installing and Using R and Related Productivity Tools

Installing Emacs, ESS, LaTeX and R: Now you can interact with the Linux OS installed under Android and install software on it. First copy the setup files and folders (for example, .emacs, .emacs.d, .gitconfig, .Renviron, .Rprofile and .ssh) from the laptop to the Linux OS as follows. (See also the Tips and Tricks section. Additionally, you can append personal settings from .bashrc to the .bashrc already available in ~ in the Linux OS.)

1) Open "Android File Transfer" and copy the files to the phone's root directory (the folder shown by default when Android File Transfer is opened). They then appear in /storage/emulated/0 on the Android device and under /mnt/0 from within the Linux OS.

2) From the latter folder, copy (via `sudo cp -r`) all the files to the home directory ~.

Next, install some basic Linux tools in the terminal via JuiceSSH (note: this may take several hours; see Figure 2 for a screenshot during the installation):

```
sudo apt-get update
sudo apt-get install a2ps auctex chromium-browser cmake curl
 ➥emacs24 git htop
sudo apt-get install make okular pdftk preview-latex subversion
 ➥texinfo unzip wget
```

Now, you can install ESS, LaTeX and base R via `sudo apt-get install ess latex r-base` if available on ARM architecture. The drawback is that software like LaTeX typically comes in older versions, so we recommend installing the latest versions from source (see the detailed instructions in the sidebar).

After this installation, restart Linux Deploy (otherwise R is not found from within JuiceSSH or VNC Viewer). Figure 3 shows a screenshot taken during the R installation process (during `make`), and Figure 4 shows R running over SSH via JuiceSSH.

Figure 3. Installation of R during `make`

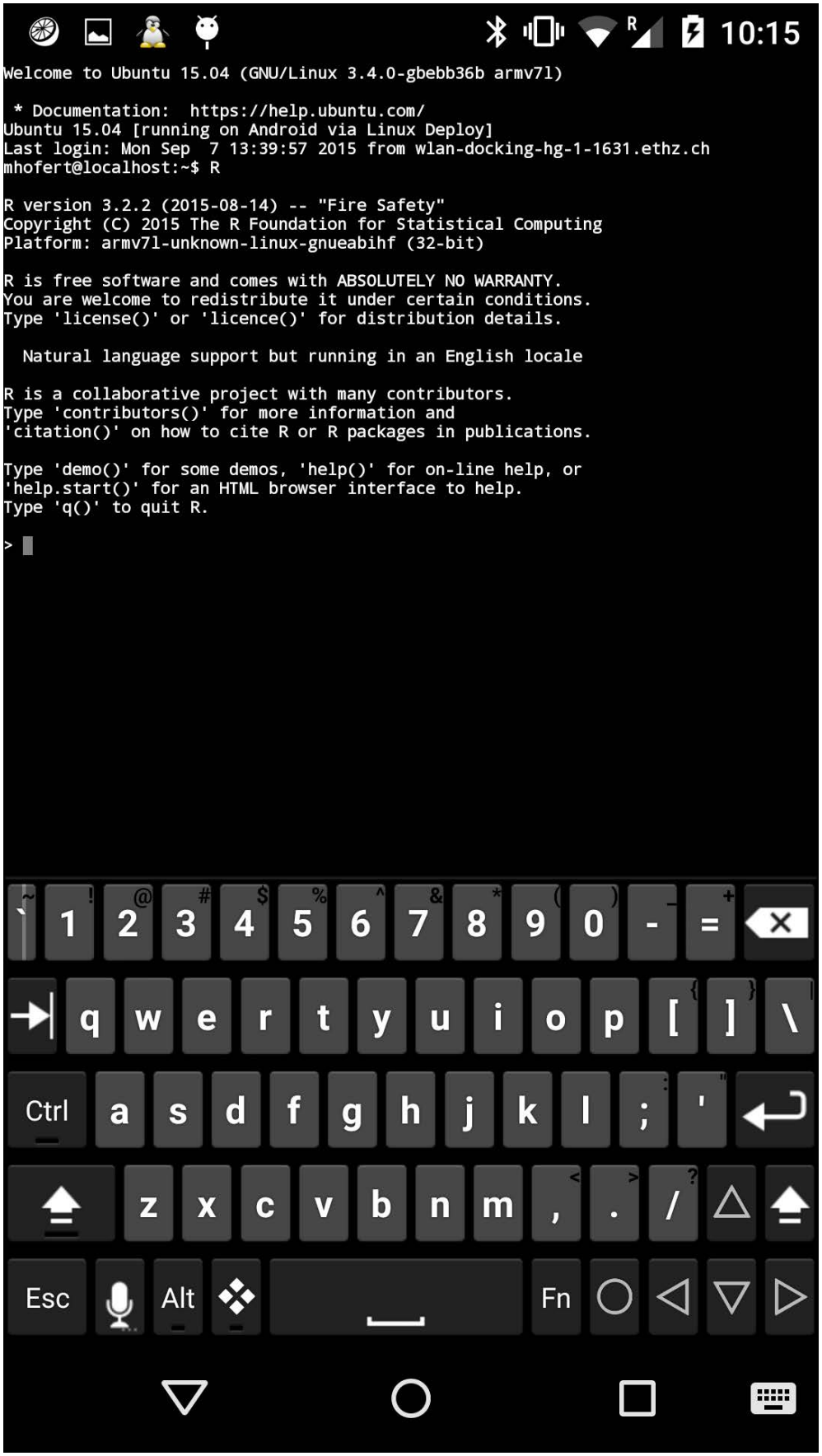


Figure 4. R Running over SSH via JuiceSSH

For installing R packages in a version-independent library, put `R_LIBS_SITE=/usr/local/R/library` in .Renviron, and make sure the /usr/local/R/library folder exists. You then simply can install

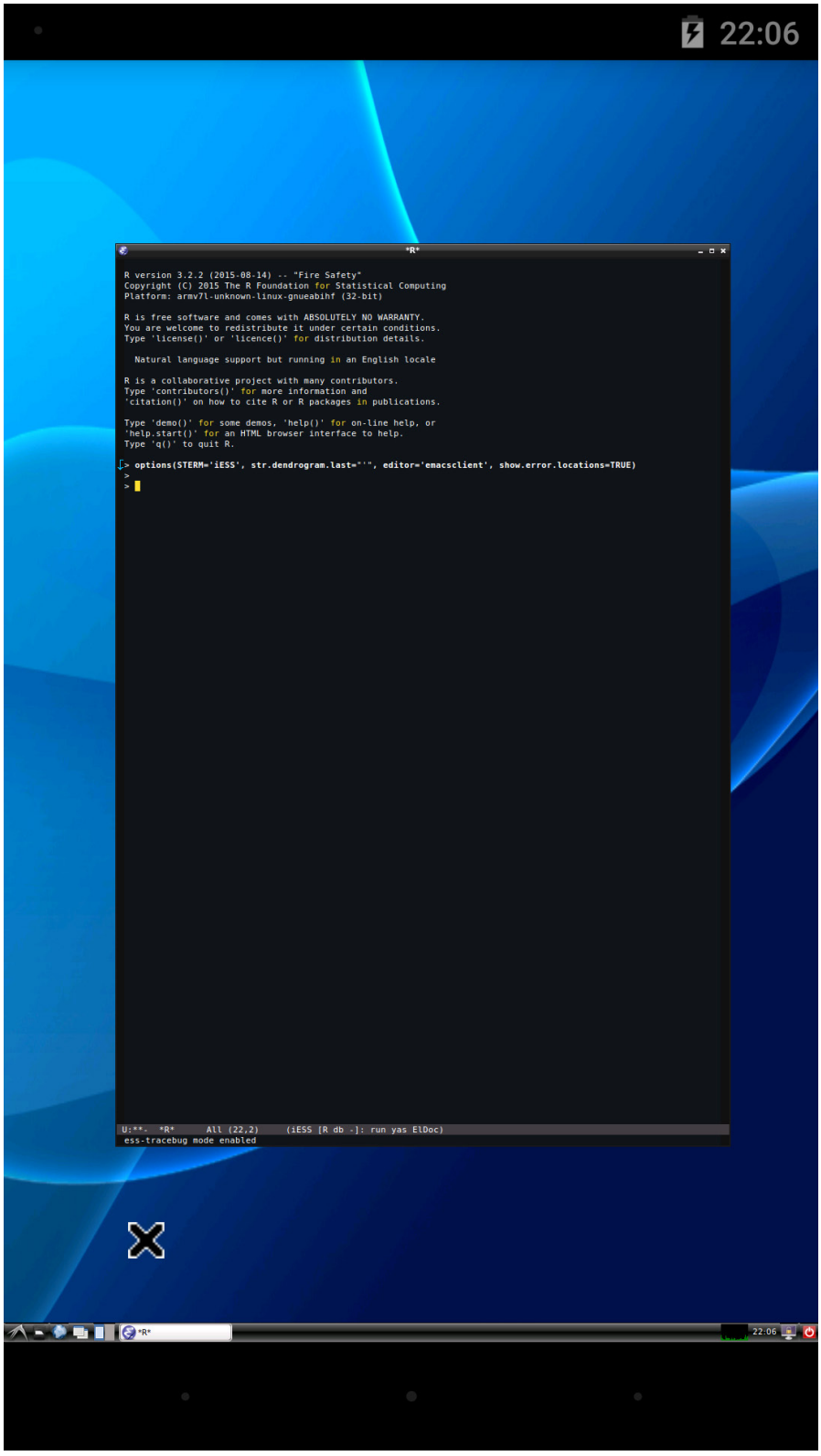


Figure 5. R Running via Emacs (and ESS) over VNC via VNC Viewer

most packages via `install.packages()` or other common methods. We installed a large number of packages (170) on the Linux OS, without problems; see Figures 5 and 6.

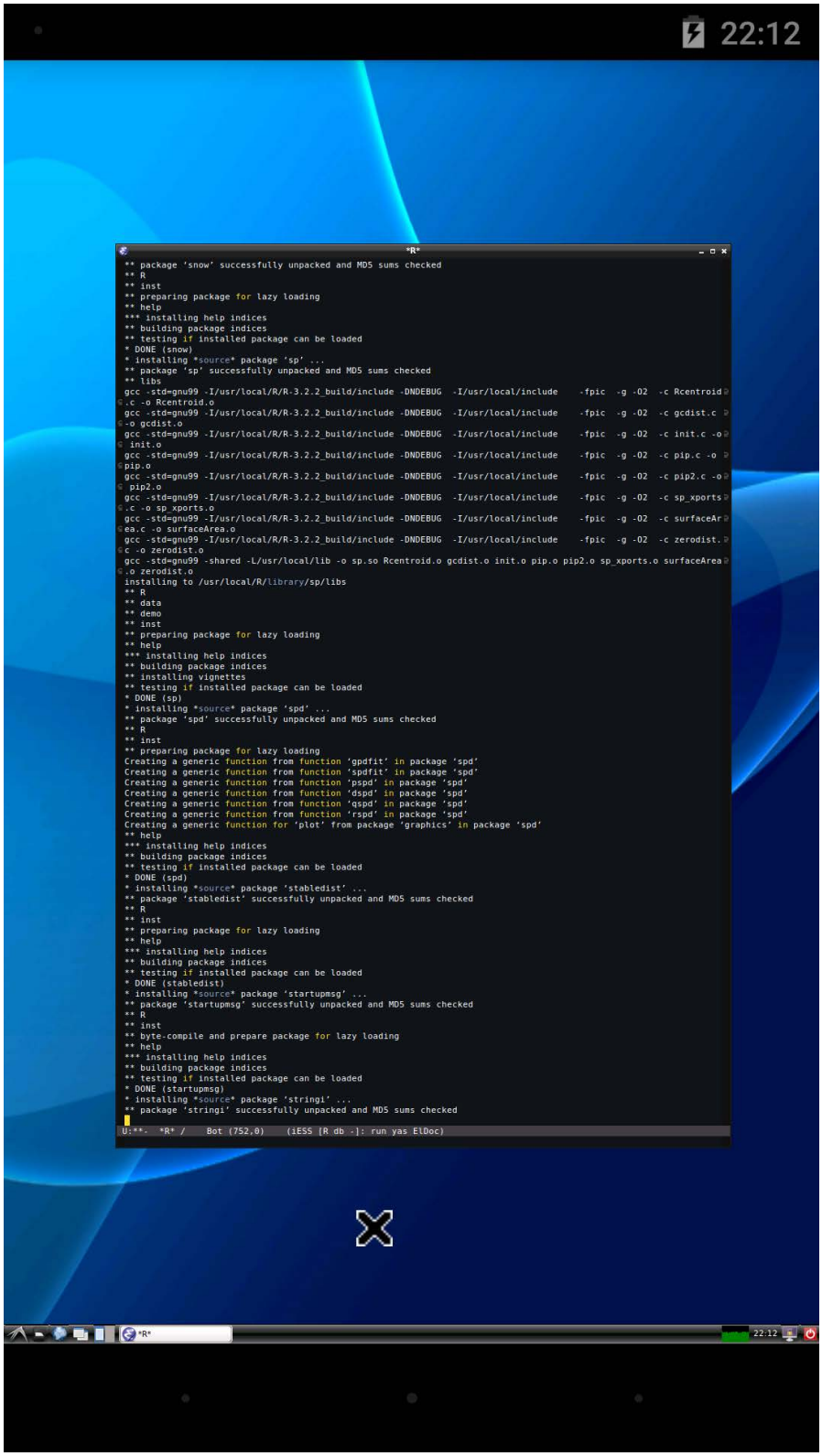


Figure 6. A Snapshot of the Installation Process of Various R Packages

As a proof of concept and for roughly comparing runtimes with the laptop, we briefly ran two R example scripts on the Google Nexus 5 smartphone.

Running R on Multiple Cores on a Smartphone

As a proof of concept and for roughly comparing runtimes with the laptop, we briefly ran two R example scripts on the Google Nexus 5 smartphone. The first is a shorter and slightly adapted version from the vignette of the R package "parallel":

```
require(parallel)
require(boot)
require(microbenchmark)

## Setup
dat <- cd4
B <- 1000 # bootstrap replications
mle <- list(mean=colMeans(dat), var=var(dat))
dat.rng <- function(data, mle) MASS::mvrnorm(nrow(data), mle$m, mle$v)
bstr <- function(...) boot(dat, statistic=corr, R=B, sim="parametric",
                           ran.gen=dat.rng, mle=mle)

## Non-parallel version
N <- 100 # number of repetitions
set.seed(271)
mc <- 2 # number of cores
(mbm.nonpara <- microbenchmark(res.nonpara <- do.call(c,
 ➥lapply(seq_len(mc), bstr)), times=N))

## Multicore version
set.seed(271)
(mbm.mc <- microbenchmark(res.mc <- do.call(c, mclapply(seq_len(mc),
 ➥bstr)), times=N)) # ms
```

As a second example, we ran a slightly adapted version of "demo(VaRsuperadd)" from the R package "simsalapar":

```
require(simsalapar)
require(copula)

## Setup
n.obs <- 1e4 # sample size
n.alpha <- 128 # number of quantiles
varList <- # list of variables
    varlist(
        n = list(value = n.obs), # sample sizes
        d = list(type="grid", value = c(4, 20, 100)), # dimensions
        family = list(type="grid", expr = quote(C), # copula families
 ➥(t=t_4)
                      value = c("normal", "t", "Clayton", "Gumbel")),
        tau = list(type="grid", value = c(0.2, 0.5, 0.8)),
 ➥# Kendall's tau
        qmargin = list(type="inner", expr = quote(F[j]), # margins
                      value = c(norm = qnorm,
                                t4   = function(p) qt(p, df=4),
                                Par2 = function(p) (1-p)^(-1/2))),
 ➥# Pareto(2)
        alpha = list(type="inner", value = 0:n.alpha/n.alpha))
 ➥# VaR confidence levels

## Function to Compute F_{X_1+..+X_d}(d*F_1^-(\alpha))
doOne <- function(n, d, family, tau, qmargin, alpha)
{
    cop <- switch(family,
                "normal" =
                  ellipCopula("normal",
 ➥param=iTau(ellipCopula("normal"), tau=tau),
                              dim=d),
                "t" =
                  ellipCopula("t", param=iTau(ellipCopula("t"),
```

```
 ➥tau=tau), dim=d),
                "Clayton" =
                  onacopulaL("Clayton",
 ➥list(th=iTau(archmCopula("clayton"), tau),
                                              seq_len(d))),
                "Gumbel" =
                  onacopulaL("Gumbel",
 ➥list(th=iTau(archmCopula("gumbel"), tau),
                                            seq_len(d))),
                stop("unsupported 'family'"))
    U <- rCopula(n, copula=cop) # sample from the copula

    ## compute F_{X_1+..+X_d}(d*F_1^-(\alpha)) for all confidence
 ➥levels alpha
    ## => VaR_alpha superadditive <=> F_{X_1+..+X_d}(d*F_1^-(\alpha)) -
 ➥alpha < 0
    t(sapply(qmargin, function(FUN)
 ➥ecdf(rowSums(FUN(U)))(d*FUN(alpha)) - alpha))
}

## Run
(dc <- parallel::detectCores())
(nc <- if(dc <= 2) 1 else 2)
system.time(res <- doMclapply(varList, cores=nc, doOne=doOne))

## Results
val  <- getArray(res) # array of values
err  <- getArray(res, "error") # array of error indicators
warn <- getArray(res, "warning") # array of warning indicators
time <- getArray(res, "time") # array of user times in ms

## Warnings, errors, run time
if(any(err > 0))
ftable(100* err, col.vars="tau") # percentage of errors
if(any(warn > 0))
ftable(100*warn, col.vars="tau") # percentage of warnings
```

```
ftable(time, row.vars=c("family", "d"), col.vars="tau") # run time

## Plot of VaR estimates (for t_4 margins)
m <- "t4"
dimnames(val)[["tau"]] <- paste0("tau==", dimnames(val)[["tau"]])
mayplot(val[qmargin=m,,,,], varList, row.vars="family", col.vars="tau",
        xvar="alpha", ylim="local")
```

The latter example concerns a standard problem in Quantitative Risk Management and involves simulating high-dimensional multivariate random vectors from various distributions, approximating the distribution function of the sum of their components and deciding, for a range of quantiles, whether

BACK UP THE DEVICE

Here's how to back up an Android device (before rooting it). For a full backup, certain backup apps require root access on the wearable device, but here we assume you don't have root yet (which is why you want to do a backup in the first place). So for this, you'll simply use the Android SDK for backup purposes (http://www.redmondpie.com/how-to-completely-backup-your-android-device-on-pc-without-root-access). For a full backup, do the following:

1) Connect the device to the laptop via USB cable.

2) Enable developer mode and USB debugging on the Android device by going to Settings→About phone and tap the build number until developer mode is enabled. Then go to Settings→About phone→Developer options, and check "USB debugging".

3) Open a shell on the laptop and execute the following (make sure the phone is on and unlocked):

```
adb start-server
adb backup -apk -shared -all # then confirm on the Android device
```

After confirmation on the Android device, the backup file backup.ab is created in the current working directory; this may take several hours depending on the storage usage of the device. Note that you can restore a backup via `adb restore backup.ab`.

the risk measure Value-at-Risk is subadditive (see the article by M. Hofert and M. Mächler, "Parallel and other simulations in R made easy: An end-to-end study", *Journal of Statistical Software*, 69(4), 2016 for more details).

Figure 7. R Code Running on Two Cores of a Google Nexus 5 Android Smartphone over a VNC Connection via VNC Viewer

Figure 7 shows the example while being run on two of the four cores of the smartphone (see the `htop` output) and the graphical output of the result (Figure 8). This example was run exactly as on the laptop, by

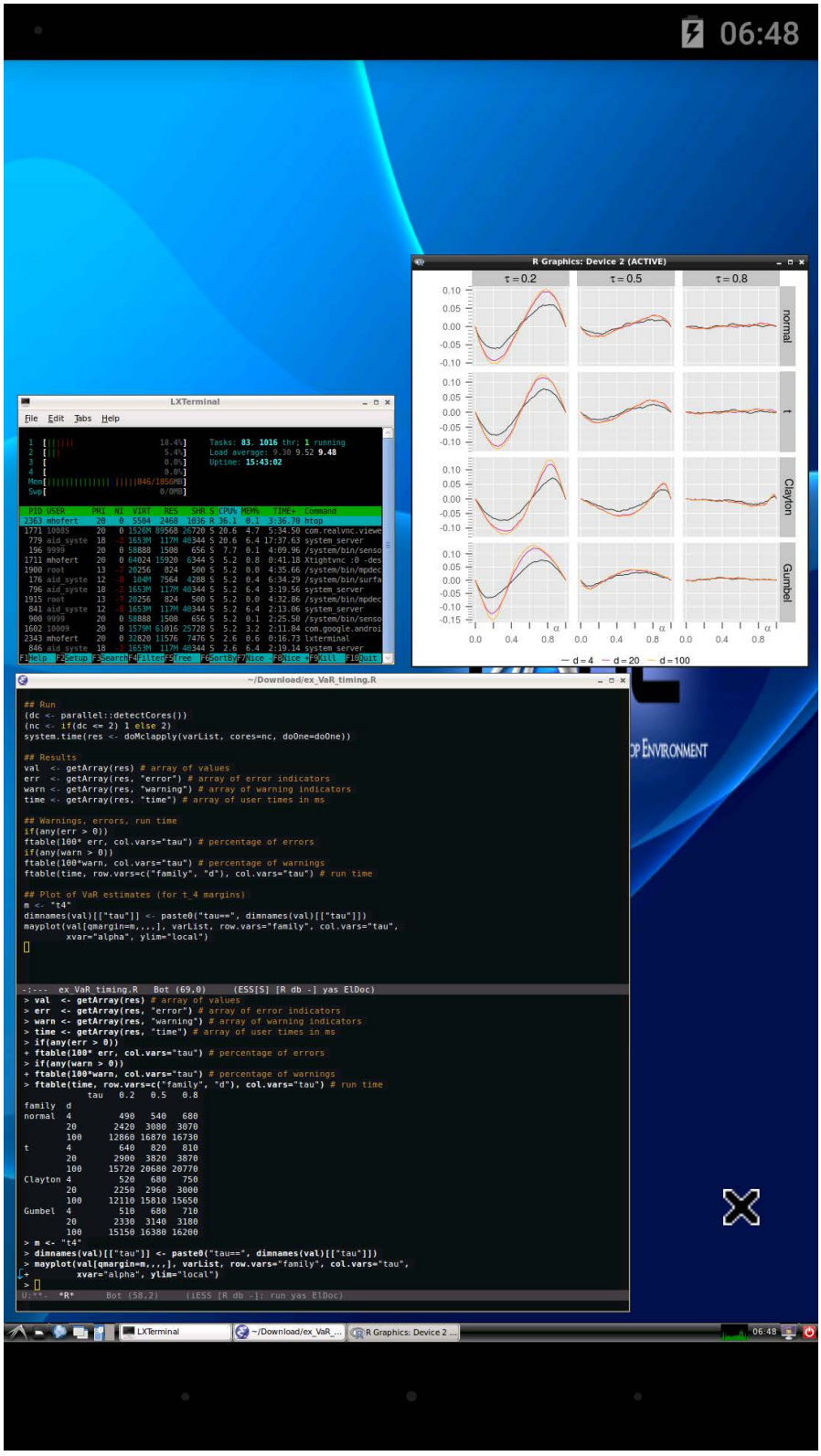


Figure 8. The Graphical Output of the Result from Figure 7

Table 1. Comparison of the runtimes on the Google Nexus 5 vs. the MacBook Pro (2015)

	Example 1 single core	Example 1 double core	Example 2 single core	Example 2 double core
MACBOOK PRO (2015)	156ms	123ms	(18s, 1.8s, 21s)	(18s, 1.7s, 12s)
GOOGLE NEXUS 5	1042ms	631ms	(141s, 4.5s, 159s)	(167s, 4.2s, 96s)

opening Emacs and executing the code line by line as one would do with most R scripts; in particular, we did not have to change our workflow to run this code on Android as we essentially run it on a Linux OS.

Table 1 shows the runtimes of the examples on the Google Nexus 5 smartphone and a MacBook Pro (2015) in a single-core and a multi-core setup. Runtime for Example 1 is measured by `microbenchmark`. Runtime for Example 2 is by `system.time()` in the form (user, system, elapsed).

Clearly, this provides only a limited insight as to R's performance on a mobile device, but there are a few things to note:

- One major aspect of R is that it provides graphics. With the approach described above, we obtain access to graphics on R without the need to print them to files to be able to access them (as is often done when working without GUI access). Even on the Google Nexus 5 (full HD—that is, 1920 x 1280 pixels), graphs looked sharp; see also Figure 4. With more and more Quad HD devices (2560 x 1440 pixels) on the market, the ability to run R (via GUI) on Android devices might become even more interesting.

- To our surprise, even after lengthy installation processes and repetitions of testing and running code on the smartphone, it did not overheat in any way (a 30-minute Skype call or Google Hangout via the smartphone produced much more heat). This might be due to the fact that there is a large gaming market on Android that overall has pushed the hardware development for smartphones quite a bit in recent years; installing and running R and related tools might still be considered rather "light" in contrast to what modern games demand from smartphones.

- The runtimes on the smartphone are, of course, larger than on the new MacBook Pro. However, if necessary, one still would be able to do some serious work on a smartphone (for example, to fix bugs, rerun code, prepare plots and slides for talks and so on), including multicore computations. Furthermore, note that the Lenovo X1 Carbon (2013; Ubuntu 14.10) we originally used during our testing phase led to (user,

UNLOCKING THE BOOTLOADER AND ROOTING (ANDROID 4.4 OR 5.0)

You can root the Google Nexus 5 running Android 4.4 or 5.0, and possibly other versions as well, by following these steps (see http://www.androidauthority.com/chainfire-successfully-roots-android-lollipop-6-nexus-devices-567409). Note: you can download the CF Auto Root files from http://autoroot.chainfire.eu.

1) Make sure that your device's battery status is at least 80% and that developer mode is enabled.

2) Boot the device into fastboot mode.

3) To unlock the device's bootloader, execute in a shell on the laptop:

```
fastboot oem unlock
```

4) Rooting (for this device and the above versions of Android) can be done via "Chainfire's Auto Root" as follows. Download the appropriate version of Chainfire's Auto Root for the device at hand (in our case, this is http://download.chainfire.eu/363/CF-Root/CF-Auto-Root/CF-Auto-Root-hammerhead-hammerhead-nexus5.zip) from http://autoroot.chainfire.eu, and unzip it. Then, navigate to the unzipped folder and execute:

```
chmod +x root-linux.sh # make the unzipped script
 ➥root-linux.sh executable ./root-linux.sh # if unlocked
 ➥already, this fails, but just hit enter/continue
```

After rooting is done, reboot the device. You can disconnect it without harm.

Next, install the Super Su app and reboot the device. Then install the Root Checker app, open it and grant superuser access. The app then indicates whether rooting the device was successful.

system, elapsed) runtimes of (161s, 7.6s, 94s), which is much closer to the runtimes on the smartphone.

- After all, we are using a smartphone from 2013 here. In the fast-developing Android world, this is rather old hardware and a more up-to-date Android flagship (with a faster processor) might be expected to score much better.

Tips and Tricks

The approach described above to run R on Android seems easier than it is. In fact, we had to do the installation procedure described here a couple times to overcome various challenges. Considering some of these issues, here are some tips and tricks:

- External SD card: our Google Nexus 5 did not allow for an external SD card. If an Android device allows for that, you can put an ext4 partition directly on the SD card and "more directly" install Linux (see, for example, http://whiteboard.ping.se/Android/Debian). More and more flagship devices do not provide expandable storage anymore though (such as the Galaxy Note line of Samsung).

- Interacting with Linux via VNC: we successfully used the bVNC app for quite a while, but it did not work properly under Android 5.1.1 and was essentially not usable anymore. Also, we originally worked with "Complete Linux Installer" instead of "Linux Deploy", however, there were other app-related challenges to overcome. As mentioned, you might need to adapt some of the tools discussed here to get the job done. Fortunately, the Android app ecosystem typically provides several apps for the same task, which gives you other options to try.

- Upgrade to Android 5.1.1: if your device can be upgraded, upgrade it to Android 5.1.1, as there have been major improvements in energy consumption over Android 4.4. When we conducted the installation process under Android 4.4, although connected via USB to the laptop the whole time (and thus being charged while in use), the battery still was (slowly) draining. Under Android 5.1.1, battery status remained constantly high.

- How files can be copied (or moved) to or from the device: for productivity, you need to be able to copy files to and from the device. As mentioned, you conveniently can use "Android File Transfer" to transfer files to the phone's root directory (the folder shown by default when Android File Transfer is opened). They then appear in /storage/emulated/0 on the Android device and under /mnt/0 from within the Linux OS. The latter is a great feature of "Linux Deploy" (and much more complicated with the "Complete Linux Installer" app, for example). A terminal-based method to copy files to the Android device is to use `adb push`—for example, doing `adb push foo.R /sdcard/Download` copies foo.R to the respective folder on Android. One limitation of `adb push` is that unlike `cp` or `mv`, it does not allow you to push several files or folders to the device at once. Finally, if you need to access files on a USB stick, many modern Android phones provide USB on-the-go (OTG) adapter cables for this task.

- The size of the Linux OS image: during one iteration of the installation procedure described above we ran out of space, although the disk image seemed to be large enough (even when we selected 18GB). It turned out that the image contained too many files (in the folder ~/.emacs.d due to backup files from Emacs being copied to the Linux OS). We then tried to enlarge the Linux OS image, but ran into several problems (including no longer being able to start the VNC server). Overall, we recommend proceeding carefully with the installation, checking disk space (for example, via `df -h` to keep track of /dev/loop0) and making sure the Linux OS image is large enough. Messing around with the image size after the installation (adjustable from within Linux Deploy) is not recommended.

- Screen size: although a smartphone's screen size is nothing you'd want to work on exclusively, the trend seems to be moving toward bigger screen sizes and phablets. Even on the 5" Google Nexus 5, it was possible to work for several hours. Besides tablets, a future alternative might be to use virtual reality goggles, which might be interesting when working on an airplane where space is very limited.

- Installing RStudio: RStudio needs to be compiled from source as no ARM binaries are available. During our testing, we tried to install RStudio by downloading https://github.com/rstudio/rstudio/archive/master.zip, copying it to the Linux OS, unzipping it, installing the required dependencies (see the install-dependencies-debian subfolder; this took several hours) and then doing `cmake . -DRSTUDIO_TARGET=Desktop -DCMAKE_BUILD_TYPE=Release`. The latter command failed though with the message `CMake Error at src/cpp/desktop/CMakeLists.txt:69 (get_filename_component): get_filename_component called with incorrect number of arguments`. An alternative would be to follow along the lines of https://github.com/dashaub/ARM-RStudio (see https://github.com/dashaub/ARM-RStudio/blob/master/ARM-RStudio.sh for the required steps), but we didn't have the required disk space left on the Google Nexus 5 to try this installation process (and it might take several hours to install).

- The input device: although technically possible with the Hacker's Keyboard, it's not practical to interact with the smartphone only over the software keyboard (a large tablet might be a different story). We connected a Bluetooth keyboard to the Google Nexus 5, but had problems allowing features like a more productive remap of the Ctrl key to Caps Lock. The app "External Keyboard Helper Settings" might be a solution here, but we could not get this (and other features) to work. We did not go as far as utilizing `xmodmap` for that purpose, as ideally, this should be solved on Android. For our proof of concept here, using the Bluetooth keyboard worked well, but the input device used to interact with Android or the Linux OS (and thus R) is still one of the major productivity bottlenecks. Also note, that the behavior was somewhat different when connected via SSH or via VNC. For example, the Emacs key combination C-space for the command M-x set-mark-command did not work when connected via SSH with JuiceSSH, but it worked when connected via VNC with VNC Viewer. Keyboards more specifically designed for Android devices and with more focus on productivity might be interesting here, such as the recently introduced LG's Rolly keyboard (which also would be easy to carry around).

- The input device (once again): one more convenient way to interact with the device during the installation process was the following. Install and run TeamViewer on the laptop (see http://www.teamviewer.com/en/index.aspx) and the "TeamViewer QuickSupport" app on the smartphone. This way you can mirror the smartphone display on the laptop and use the laptop's keyboard as an input device. Two hiccups we encountered here were 1) some Emacs key combinations still were not sent to the device properly (this may be related to the problem mentioned previously), and 2) the laptop shows the smartphone's display with a small but annoying time lag. Nevertheless, using the laptop's keyboard while looking at the smartphone's display worked well and did not require a Bluetooth keyboard.

Conclusion

This article presents an approach for running a full version of R and related productivity software on Android devices within a Linux OS in a chroot environment. SSH or GUI access (via VNC) allow you to connect to the Linux OS and run R and related productivity tools (even on multiple cores).

Some advantages of this approach:

- You don't have to make sacrifices on the software side; you can run full-blown versions of Linux, R, LaTeX, Emacs and so on.
- You don't have to use the Android device as a thin client, so you don't need to have Internet access.
- All software tools used here are freely available at no cost.
- Once the installation process is finished, you can access R simply by starting Linux via Linux Deploy→Start and then connecting to the running Linux OS via SSH (with JuiceSSH) or VNC (with VNC Viewer).
- With a VNC connection, you have full support of R graphics.
- You easily can access and transfer files from a laptop to Android, the Linux OS and back.

- The performance for everyday tasks is good. Also, more and more Android devices come with 64-bit (and much faster) processors (such as the Snapdragon 810), which may lead to significant improvement over the performance we experienced.

Although no real show-stoppers, there are some drawbacks we should mention as well:

- The installation process is (much) more time-consuming than simply installing more apps, simply because you basically are installing full desktop versions of Linux, R, LaTeX and so on.

- Updating the Linux OS via `apt-get` is straightforward, but updating Android requires a complete re-installation, including rooting. Note that this problem might be circumvented when the Linux OS is installed on an external SD card (if supported by the device).

- As mentioned earlier, the device's location affects the IP address of the SSH server. As this is not propagated to JuiceSSH automatically, this is a bit inconvenient (and a typical source of error when the connection to the Linux OS fails).

- As pointed out previously, the main bottleneck is not the weaker processing power of the Android device, but rather the interaction with device itself. On the one hand, you could try harder to make a full keyboard work more easily with Android (including, for example, a faster key repeat rate or more key combinations for productive work). On the other hand, you might raise the question as to why you wouldn't just work with a laptop if a full keyboard is required. More portable input solutions can be expected in the future, but the classical keyboard hasn't changed very much since its introduction. Two interesting mobile approaches for the future seem to be: 1) a keyboard developed by Tactus Technology (http://tactustechnology.com), which can appear/disappear dynamically and, thus, enhance software keyboards (this might be especially interesting on tablets where you can type with ten fingers); and 2) a glove keyboard (https://www.youtube.com/watch?v=vij420qVqCY),

although it's far from the production stage. With more and more mobile computing devices being used, we hope to see more development on mobile input devices in the future, especially for productivity purposes. For example, software keyboards, such as SwiftKey, Swype, Google Keyboard or Fleksy (offering different enhancements for speed-typing text), are widely used on wearable devices, but are practically of no use for productivity purposes, as certain keys or key combinations (for coding, for example) are simply not available.

Overall, we were pleasantly surprised by how "far" mobile productivity (including R) already is, and we hope that the quirks will be solved in the near future. Projects like Remix Mini (https://www.kickstarter.com/projects/1123481999/remix-mini-the-worlds-first-true-android-pc) even aim at bringing this experience (and the related "less is more" paradigm) to the desktop (still in a portable way). For future research, a glance toward Mac OS/iOS certainly would be interesting. Besides available thin clients, the current state of the art on mobile iDevices is unclear to us. A rather old post (http://www.r-bloggers.com/running-r-on-an-iphoneipad-with-rstudio) briefly addresses some issues (such as jailbreaking and performance), but that might not apply to newer Apple devices anymore. ■

After completing a diploma in Mathematics and Management at University of Ulm and a Masters' degree in Mathematics at Syracuse University, **Marius Hofert** obtained his PhD in Mathematics from University of Ulm in 2010. He then held a postdoctoral research position (Willis Research Fellow) at RiskLab, ETH Zurich. After a guest professorship (W2) in the Department of Mathematics at Technische Universität München and a visiting assistant professorship in the Department of Applied Mathematics at University of Washington, Seattle, he joined the Department of Statistics and Actuarial Science at University of Waterloo in July 2014. One of his research interests is computational statistics.

Kurt Hornik holds a diploma and doctorate in applied mathematics and a habilitation for statistics and its mathematical foundations, all from Technische Universität Wien. He joined WU Wirtschaftsuniversität Wien as a full professor for statistics and mathematics in 2003. His main research interests are in statistical computing. He is a member of the core development team for R, and the principal architect and maintainer of the Comprehensive R Archive Network.

INSTALLING THE LATEST VERSIONS OF PRODUCTIVITY TOOLS

To install the latest version of TeX Live (LaTeX) on the Linux OS, do the following (adjust the version numbers accordingly):

```
mkdir ~/Downloads
cd ~/Downloads
curl -OL http://mirror.ctan.org/systems/texlive/tlnet/
➥install-tl-unx.tar.gz
tar -xzf install-tl-unx.tar.gz
cd install-tl-20150906 # use the correct version number
sudo ./install-tl # enter the password, then hit "I"
```

After the installation, a path is displayed. Make TeX Live available by adding this path to /etc/bash.bashrc. In our case, we used:

```
sudo vi /etc/bash.bashrc # Then add the following
PATH=/usr/local/texlive/2015/bin/armhf-linux:$PATH
```

After closing and restarting the Linux OS, LaTeX is available; this installation procedure of LaTeX took roughly 45 minutes.

For Emacs users, you can install the latest version of ESS as follows:

```
cd /usr/share/emacs/site-lisp/
sudo svn co https://svn.r-project.org/ESS/trunk ess
cd ess
sudo make
```

For this to work, put the following in ~/.emacs:

```
(add-to-list 'load-path "/usr/share/emacs/site-lisp/ess/lisp")
(require 'ess-site)
```

To update ESS at any point, simply navigate to /usr/share/emacs/site-lisp/ess and execute `sudo svn up; sudo make`.

Now you can install the latest version of R (from source). Start by installing some Linux preliminaries as follows:

```
sudo apt-get install gsl-bin libgsl0-dbg libgsl0-dev
 ➥libgsl0ldbl # for GNU GSL
sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev
 ➥# for open MPI
sudo apt-get install libmpfr4 libmpfr4-dbg libmpfr-dev # for
 ➥GNU MPFR
```

```
sudo apt-get install libgmp10 libgmp3-dev # for GMP
sudo apt-get install libxml2-dev # for package 'XML'
```

Next, install R (for updating an existing version, simply redo steps 3–7) as follows:

1) Build dependencies:

```
sudo apt-get build-dep r-base # ~ 10min
```

2) Create installation directory:

```
sudo mkdir /usr/local/R
sudo chown <username>:<username> /usr/local/R # put in your
 ➥username
```

3) Get and unpack R:

```
cd /usr/local/R
wget http://cran.r-project.org/src/base/R-3/R-3.2.2.tar.gz
 ➥# see CRAN -> R Sources
tar -xzf R-3.2.2.tar.gz
mv R-3.2.2 R-3.2.2_source
```

4) Set up the build directory:

```
mkdir R-3.2.2_build
```

5) Configure:

```
cd R-3.2.2_build
../R-3.2.2_source/configure # ~ 10min
```

6) Build:

```
make # ~ 40min
make check # ~ 10min
make pdf # ~ 8min
make info # ~ 2min
```

7) Link to R (set /usr/local/R/R to call R-3.2.2_build/bin/R):

```
cd ..
ln -s /usr/local/R/R-3.2.2_build/bin/R R-3.2.2
 ➥# version-dependent link
ln -s /usr/local/R/R-3.2.2_build/bin/R /usr/local/R
 ➥# => 'R' gives the default
```

8) Add /usr/local/R to PATH by putting in ~/.profile:

```
PATH=/usr/local/R:$PATH
```

Resources

"CES 2014: Why wearable technology is the new dress code":
http://www.theguardian.com/technology/2014/jan/08/wearable-technology-consumer-electronics-show

"2014 Will Be The Year Of Wearable Technology" by Evan Spence:
http://www.forbes.com/sites/ewanspence/2013/11/02/2014-will-be-the-year-of-wearable-technology

"GRU professor: 2014 is year of wearable devices" by Wesley Brown:
http://chronicle.augusta.com/news/metro/2014-01-05/gru-professor-2014-year-wearable-devices

Bjarne Stroustrup: "How to Code Like Bjarne Stroustrup" (YouTube):
http://www.youtube.com/watch?v=tj8BoOYvo00

Ubuntu Edge: http://www.indiegogo.com/projects/ubuntu-edge

Smartphone OS Market Share, 2015 Q2:
http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Worldwide Tablet Growth Expected to Slow to 7.2% in 2014 Along With First Year of iPad Decline, According to IDC: http://www.idc.com/getdoc.jsp?containerId=prUS25267314

Usage share of operating systems:
https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Android Studio: https://developer.android.com/sdk/index.html

How to Flash Nexus Factory Images:
http://www.droid-life.com/2013/11/05/how-to-flash-nexus-5-factory-images

XDA-Developers Android Forums: http://www.xda-developers.com

Android Authority: http://www.androidauthority.com

Android Enthusiasts Stack Exchange: http://android.stackexchange.com

How to root Nexus 5 on Android 5.1 Lollipop:
https://www.androidpit.com/how-to-root-nexus-5-on-android-lollipop

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

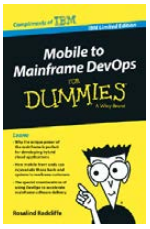
How long does it take to restore a system, whether virtual or physical, back to the exact state it was prior to a failure? Re-installing the operating system, re-applying patches, re-updating security settings takes too damn long! If this is your DR Strategy, we hope you've documented every change that's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

What's Our Next Fight?

We won the battle for Linux, but we're losing the battle for freedom.

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:
How We R on Android

Linux turns 25 in August 2016. *Linux Journal* turned 21 in April 2016. (Issue #1 was April 1994, the month Linux hit version 1.0.) We're a generation into the history of our cause, but the fight isn't there anymore, because we won. Our cause has achieved its effects.

It helps to remember that Linux was a fight, and so were free software and open source. If they weren't fights, they wouldn't have won what they did. They also wouldn't have been interesting, meaning there wouldn't have been any Linux stories, or a *Linux Journal*.

Stories are what make a subject interesting. To program a story, you need three elements:

1. A protagonist, which could be a person, a group or an easily personified cause that people can identify with.

Damn near everything runs on Linux, or on something so similar that you can open a shell on it and get stuff done.

2. A problem, or a set of problems, against which the protagonist struggles.

3. Movement toward resolution.

If you lack one those, you don't have a story.

With Linux, we had a cause and a person who personified it, whether he liked it or not. Our problem was mentalities embodied in opponents such as Microsoft and the herd of dull enterprise suppliers of "solutions" based on proprietary variants of UNIX. As Linux and its opposition grew, we had movement toward what Linus called "world domination".

Which is where we are now. In terms of actual use, Linux's *quo* has more *status* than any of its early opponents ever had. Damn near everything runs on Linux, or on something so similar that you can open a shell on it and get stuff done. (Example: Apple's OS X, which wouldn't be what it is if Linux hadn't already been the leading *nix OS.) Even Microsoft runs lots of its own stuff (such as Bing) on Linux. Bill Gates no longer cares. He's a philanthropist now. Steve Jobs is dead. Linux's old UNIX enemies are zombies or gone. And, most of the world's smart mobile devices run on Android, a derivative of Linux.

So what's our next fight?

Here are some candidates. Rather than argue with any of the cases I make for them (which will all be too brief), tell me the cases we *should be* making. Think: *What will grow our community of readers and writers here at Linux Journal?* and *What effects do we want to have in the world?*

General-Purpose Computing and Networking Linux was born on a generic 386 computer and grew on a boundless population of others that were called "compatibles" or "clones" in the early days.

They were an accident of history. When Intel introduced the 8086 in 1978, the idea wasn't to make its descendants the most ubiquitous CPUs ever. Intel wanted to sell chips to makers of closed and proprietary devices at a time when there were no other kinds. "Open" back then meant "we'd like this thing to get along with some other things as long as it doesn't threaten our market position".

When IBM came out with the 8088-based IBM PC in 1981, the idea was to sell desktop IBM boxes into the business marketplace. The PC succeeded mostly because it had a lot of backplane, a strategy modeled by the Apple II in prior years (and abandoned by Steve Jobs with the first round of Macintosh computers). This opened markets for expansion cards, peripherals, publications, training, events and software to run on MS-DOS—the OS that IBM, in a move deeply out of character, licensed from somebody else. It helped that the PC could run other OSes as well, such as CP/M. But not many bothered with that until Linus broke through nine years later, thanks to another accident of history called the Internet. That too was a general-purpose thing that no company ever would have invented on its own. Even as late as 1994, Microsoft fought the Internet with an "on-line service" of its own called Marvel (http://scripting.com/davenet/1994/10/18/billgatesvstheinternet.html). Fortunately for Microsoft, Marvel failed instantly. Bill Gates then got wise, and issued his "Internet Tidal Wave" memo in early 1995.

IBM saw the PC as an exclusive hardware play. The only reason it failed to remain exclusive was that Phoenix Technologies came out with a compatible BIOS (http://www.computerworld.com/article/2585652/app-development/reverse-engineering.html), which it reverse-engineered to keep it legal. The clone market was born when Phoenix then began selling its knock-off BIOS, and it grew as chip makers did the same with knock-offs of the x86 CPU. As a result, the PC became a generic commodity, and the general-purpose computer was born, and it's still with us.

For now.

In *The Future of the Internet—and How to Stop It* (https://dash.harvard.edu/handle/1/4455262), Jonathan Zittrain calls general-purpose computers and networks *generative*, meaning by nature they generate and support countless other inventions and services, and the markets that grow around them.

The term "platform" suggests a bottom level that supports stuff

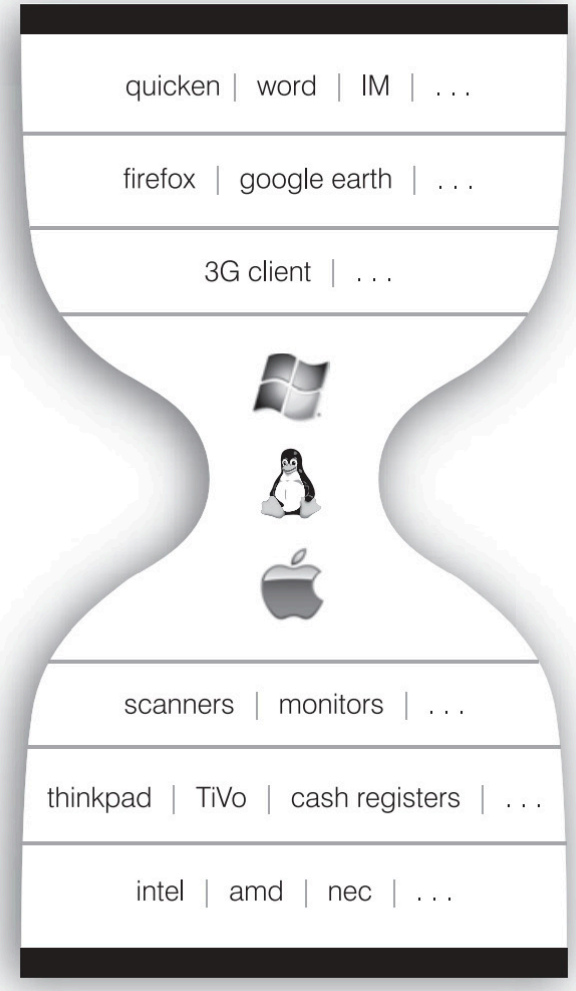


Figure 1a. PC Hourglass

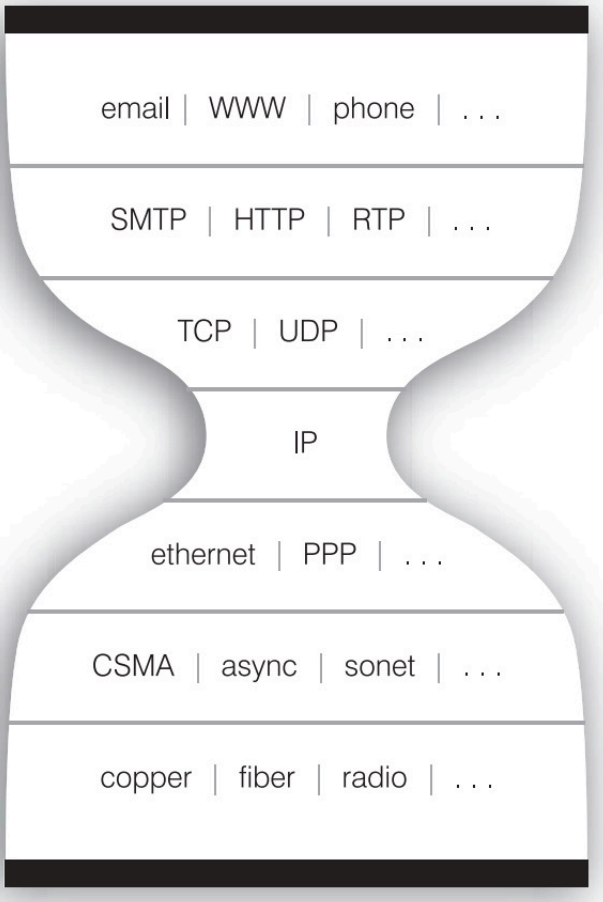


Figure 1b. IP Hourglass

above. But generative computers and networks support whole markets both above and below their own level in the stack. To illustrate this, Zittrain uses an hourglass, with the generative thing at the waist in the middle (Figure 1).

He also notes how Apple, for example, limits hardware generativity below the waist of the hourglass by preventing other companies from making devices that run its OS.

More important, he warns that generativity itself is under threat by a new model exemplified by giant controlling vendors such as Google, Apple, Facebook and Amazon (together called "GAFA" in Europe). Each, he says, has a "model for lockdown" that:

> ...exploits near-ubiquitous network connectivity to let vendors

It's much worse now. The general-purpose PC business is itself a zombie.

> change and control their technologies long after they've left the factory—or to let them bring us, the users, to them, as more and more of our activities shift away from our own devices and into the Internet's "cloud".
>
> These technologies can let geeky outsiders build on them just as they could with PCs, but in a highly controlled and contingent way....
>
> This model is likely the future of computing and networking, and it is no minor tweak. It is a wholesale revision to the Internet and PC environment we've experienced for the past thirty years....We are at risk of embracing this model, thinking it is the best of both worlds—security and whimsy—when it may be the worst. Even fully grasping how untenable our old models have become, consolidation and lockdown need not be the only alternative. We can stop the future.

That book came out in 2008. It's much worse now. The general-purpose PC business is itself a zombie. IBM is long gone, having sold its PC business to Lenovo, which makes nice boxes but also likes to install adware on its new laptops. Other clone-makers have left the business or the planet entirely. Microsoft now makes its own closed and proprietary hardware on the Apple model. And, Google has at least as much control over the Android mobile device market as Microsoft had over its PC OEMs, back in the decade.

CyanogenMod (http://www.cyanogenmod.org) is a worthy Android alternative, but Google appears to be controlling the mobile market at least as well as Microsoft controlled its PC one. Mobile hardware also gets old fast, making it a swarm of moving targets, all changing constantly. So it's hard for a generative OS to support whole stacks of hardware below and software above.

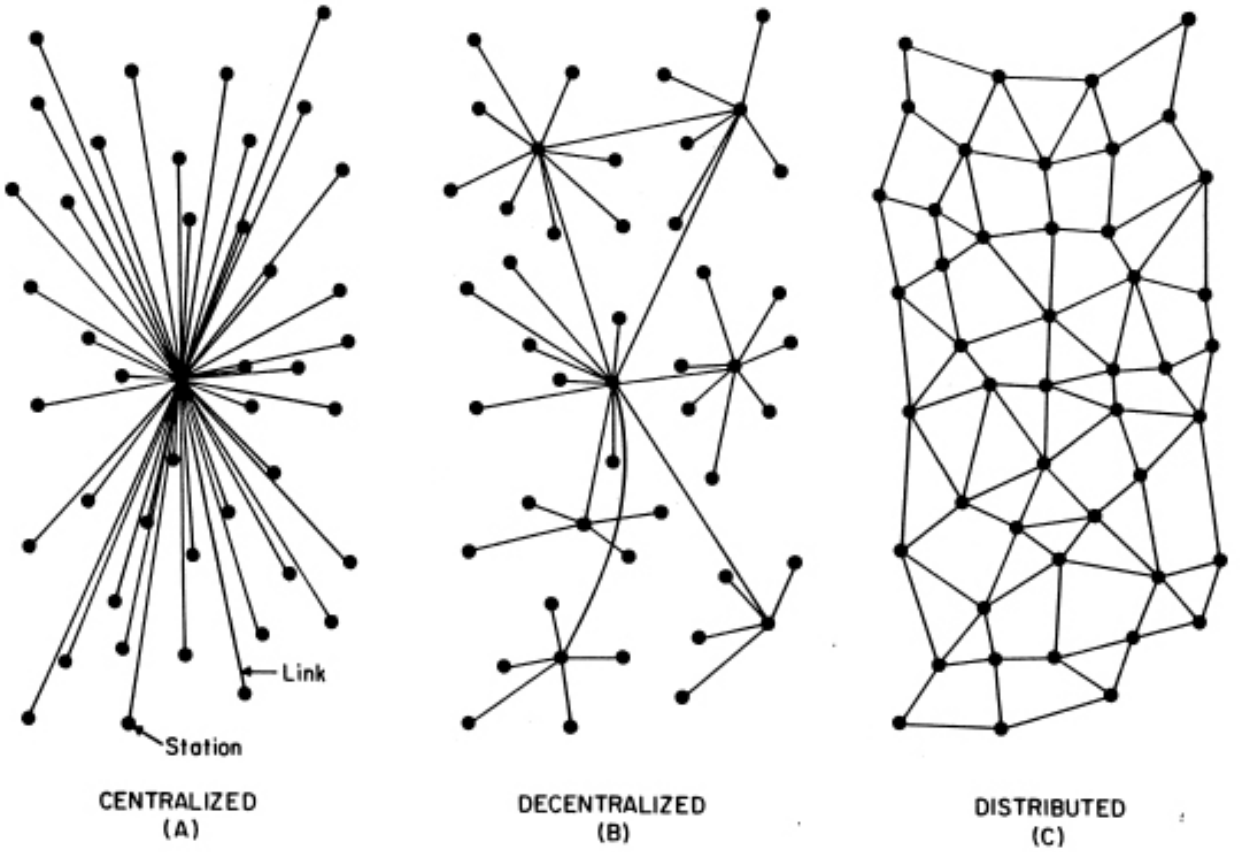


Figure 2. Centralized, Decentralized and Distributed Networks

All four GAFA companies also are better at taking advantage of our next enemy: centralization.

Decentralization and Distributed Everything The original model for the Internet was drawn by Paul Baran (https://en.wikipedia.org/wiki/Paul_Baran) in 1962 (http://www.rand.org/about/history/baran.html). It's the one on the right in Figure 2.

He called it "distributed", to contrast it with "centralized" and "decentralized", which were the prevailing network architectures of the time, and for the foreseeable future. As Baran saw it, a distributed network would be composed of independent peers, each of which could connect to any other peer or combination of peers. TCP/IP (https://en.wikipedia.org/wiki/Internet_protocol_suite), the Internet's base protocol pair, assumed a distributed network to begin with, and that's why it became so generative in 1995, when commercial activity was no longer kept off of it.

Yet nearly all the sites and services using the Web are built on the client-server computing model (https://en.wikipedia.org/wiki/Client-server_model). While client-server is ideal for distributed applications, it still presumes a server as a center, and can be used to overlay many centralized assets and services onto the distributed Internet. This is how dominant companies create worldwide webs of deep dependencies, controlling whole markets for hardware, software, providers, customers and up to a billion and more users.

One of the great inventions on the Web was blogging. Thanks to RSS, anybody could syndicate what he or she wrote to the whole world, meaning anybody's publication easily could get subscribers. I started blogging myself in late 1999. By 2003, I was one of the top 20 bloggers in the world, according to Technorati, the blog search engine that Dave Sifry (https://en.wikipedia.org/wiki/Dave_Sifry) invented while helping me write a story about blogging for *Linux Journal* (http://www.linuxjournal.com/article/6497). My blog (http://doc.weblogs.com) had between a few and many thousands of visitors per day, most from people who subscribed to my RSS feed.

Blogs were part of what my son Allen around the same time called "the live Web" (http://www.linuxjournal.com/article/8549), which he saw branching off the "static" Web of "sites" at "addresses" with "domains" and "locations" that were "built" and "visited" or "browsed".

Back in the early 2000s, it would take search engines like Google up to a month to re-visit and index a static Web site. But over the coming years, three things caused the live Web to eat the static one: search engine time-to-index approached zero, "social media" (starting with Twitter and Facebook) took off, and smartphones (with their apps) became a required accessory.

During that transition, Hossein Derakhshan (https://en.wikipedia.org/wiki/Hossein_Derakhshan), a Canadian journalist from Iran who blogged by the handle Hoder (at hoder.ir, now gone), served six years in an Iranian prison (yes, for his blogging) getting out in 2014. Appalled by what happened to the Web, and especially to blogging, he wrote "The Web We Have to Save" (https://medium.com/matter/the-web-we-have-to-save-2eb1fe15a426#.6ktnkemaw) on the centralized blogging platform Medium (https://medium.com), a recent creation of Ev Williams

(https://en.wikipedia.org/wiki/Evan_Williams_(Internet_entrepreneur)), who co-created Blogger (https://en.wikipedia.org/wiki/Evan_Williams_%28Internet_entrepreneur%29#Pyra_Labs_and_Blogger), which was acquired by Google in 2003 and somehow survives. In that piece, Hossein wrote, "The rich, diverse, free web that I loved—and spent years in an Iranian jail for—is dying. Why is nobody stopping it?"

He especially mourned the loss of hyperlinks that make the Web a web:

> Since I got out of jail, though, I've realized how much the hyperlink has been devalued, almost made obsolete.
>
> Nearly every social network now treats a link as just the same as it treats any other object—the same as a photo, or a piece of text—instead of seeing it as a way to make that text richer. You're encouraged to post one single hyperlink and expose it to a quasi-democratic process of liking and plussing and hearting: adding several links to a piece of text is usually not allowed. Hyperlinks are objectivized, isolated, stripped of their powers.
>
> At the same time, these social networks tend to treat native text and pictures—things that are directly posted to them—with a lot more respect than those that reside on outside web pages. One photographer friend explained to me how the images he uploads directly to Facebook receive a large number of likes, which in turn means they appear more on other people's news feeds. On the other hand, when he posts a link to the same picture somewhere outside Facebook—his now-dusty blog, for instance—the images are much less visible to Facebook itself, and therefore get far fewer likes. The cycle reinforces itself.
>
> Some networks, like Twitter, treat hyperlinks a little better. Others, insecure social services, are far more paranoid. Instagram—owned by Facebook—doesn't allow its audiences to leave whatsoever. You can put up a web address alongside your photos, but it won't go anywhere. Lots of people start their daily online routine in these cul de sacs of social media, and their journeys end there. Many don't even realize that they're using the Internet's infrastructure when

> they like an Instagram photograph or leave a comment on a friend's Facebook video (http://qz.com/333313/milliions-of-facebook-users-have-no-idea-theyre-using-the-internet). It's just an app.
>
> But hyperlinks aren't just the skeleton of the web: they are its eyes, a path to its soul. And a blind webpage, one without hyperlinks, can't look or gaze at another webpage—and this has serious consequences for the dynamics of power on the web.

What made this happen is centralization. The GAFA giants and their like dominate by plying the arts and sciences of centralization to a near-absolute degree. As a result we are forgetting and failing to protect the distributed nature of the Net itself.

I despair of fighting this, and said so in "Giving Silos Their Due" (http://www.linuxjournal.com/content/giving-silos-their-due), an EOF a few months back. Phil Windley responded with "Decentralization Is Hard, Maybe Too Hard" (http://www.windley.com/archives/2016/02/decentralization_is_hard_maybe_too_hard.shtml), which was even more despairing. Writes Phil:

> I remember telling Doc a while back that I'm often afraid that the Internet is an aberration. That it's a gigantic accident brought on by special circumstances. That accident showed us that large-scale, decentralized systems can be built, but those circumstances are not normal.

We have now lived so long as serfs in GAFA's feudal castles (https://www.schneier.com/blog/archives/2012/12/feudal_sec.html) that it is hard to imagine the networked world lacking dependence on overlords to provide much of what we need and take for granted—all on their terms rather than ours. Which brings us to our next cause.

Privacy: At Black Hat 2015, Jennifer Stisa Granick (http://cyberlaw.stanford.edu/about/people/jennifer-granick), Director of Civil Liberties at the Stanford Center for Internet and Society (http://cyberlaw.stanford.edu) gave a keynote talk titled "The End of the

Internet Dream" (https://backchannel.com/the-end-of-the-internet-dream-ba060b17da61#.spambd160). Among many other scary things, she said this:

> The first casualty of centralization has been privacy. And since privacy is essential to liberty, the future will be less free.
>
> This is the Golden Age of Surveillance. Today, technology is generating more information about us than ever before, and will increasingly do so, making a map of everything we do, changing the balance of power between us, businesses, and governments. The government has built the technological infrastructure and the legal support for mass surveillance, almost entirely in secret.
>
> Here's a quiz. What do emails, buddy lists, drive back-ups, social networking posts, web browsing history, your medical data, your bank records, your face print, your voice print, your driving patterns and your DNA have in common?
>
> Answer: The US Department of Justice (DOJ) doesn't think any of these things are private. Because the data is technically accessible to service providers or visible in public, it should be freely accessible to investigators and spies....
>
> The physical design and the business models that fund the communications networks we use have changed in ways that facilitate rather than defeat censorship and control.

Privacy is something we define and control with technology and norms. In the physical world, we've had thousands of years to create those, starting with the original privacy tech: clothing and shelter. In the networked world, we've had only a couple decades. That's not enough. So we have a lot of work to do, starting with the equivalents of clothing and shelter. What are those? The answers need to be ones any muggle can use—not just wizards like us.

The True Internet of Things: The "Internet of Things" today is a mess traveling as a fantasy. Most Internet-connected "things" sold by Amazon,

Google, GE and others live in silo'd systems meant to trap customers and fail—on purpose—to interoperate with things in other companies' silos. Together these comprise what Phil Windley calls The CompuServe of Things (http://www.windley.com/archives/2014/04/the_compuserve_of_things.shtml).

Worse, many of them are designed to spy on you. As Jennifer Granick puts it:

> Now we have networked devices, the so-called Internet of Things, that will keep track of our home heating, and how much food we take out of our refrigerator, and our exercise, sleep, heartbeat, and more. These things are taking our off-line physical lives and making them digital and networked, in other words, surveillable.

Shoshana Zuboff says this is inevitable (http://www.faz.net/aktuell/feuilleton/the-surveillance-paradigm-be-the-friction-our-response-to-the-new-lords-of-the-ring-12241996.html?printPagedArticle=true#pageIndex_2), because it follows three laws:

> First, that everything that can be automated will be automated. Second, that everything that can be informated will be informated. And most important to us now, the third law: in the absence of countervailing restrictions and sanctions, every digital application that can be used for surveillance and control will be used for surveillance and control, irrespective of its originating intention.

So, in obedience to an original intention of giving you better advertising, new Samsung TVs watch you while you watch them. Exactly what nobody will ever ask for.

Phil Windley sums up the challenge this way:

> On the Net today we face a choice between freedom and captivity, independence and dependence. How we build the Internet of Things has far-reaching consequences for the humans who will use—or be used by—it. Will we push forward, connecting things using forests of silos that are reminiscent of the on-line services of the 1980s, or will we learn the lessons of the Internet and build a true Internet of Things?

Good question.

Freedom: I have a long list of other topics, but every one I can think of goes back to where we were in the first place—or before the first place. To freedom.

Linux is called Gnu Linux by many in the Free Software movement. Their ethos and their code helped make Linux possible, and Linux still embodies both.

The problem with "free software", besides the fact that it needed explanation ("free as in freedom, not as in beer"), was that it had no box office. "Open source" did have box office, and we (myself included) did a pretty good job of getting it known, if not well understood, by the whole technical world.

I don't think that making a big thing about open source hurt the cause of freedom. But I also don't think it helped much, if at all. Regardless of the causalities involved, we took our eye off the freedom ball. Here's how Eben Moglen put it in a talk at Freedom to Connect in 2012 called "Innovation under Austerity" (https://www.softwarefreedom.org/events/2012/freedom-to-connect_moglen-keynote-2012.html):

> ...if we'd had a little bit more disintermediated innovation, if we had made running your own Web server very easy, if we had explained to people from the very beginning how important the logs are, and why you shouldn't let other people keep them for you, we would be in a rather different state right now.
>
> The next Facebook should never happen. It's intermediated innovation serving the needs of financiers, not serving the needs of people. Which is not to say that social networking shouldn't happen, it shouldn't happen with a man-in-the-middle attack built in to it. Everybody in this room knows that. The question is how do we teach everybody else....
>
> The nature of the innovation established by Creative Commons, by the Free Software Movement, by Free Culture, which is reflected in the Web, in Wikipedia, in all the Free Software operating systems now running everything, even the insides of all those locked-down vampiric Apple things I see around the room. All of that innovation comes from the simple process of letting the kids play and getting out of the way. Which, you are

aware, we are working as hard as we can to prevent now completely.

Increasingly, all around the world the actual computing artifacts of daily life for human individual beings are being made so you can't hack them. The computer science laboratory in every twelve-year-old's pocket is being locked-down.

How did we let that happen? And who are "we" anyway? In "A Tale of Three Cultures" (http://www.linuxjournal.com/article/5912), which ran in *LJ* in 2002, I tried to pull apart the separate cultures in our community:

One is purely technical. It's pre-Net, pre-UNIX and maybe even pre-cultural. It shows up where raw technology meets the real world, and its concerns are utterly practical. "Here's the problem", it says. "Let's solve it." This is a heads-down culture and civilization depends on it. Embedded systems are what run our cash registers and brake systems, our airplane guidance systems, our factory robotics, our flow meters, our stoplights and our heating systems. The Net and Linux are both handy ways to solve countless embedded systems problems—extremely handy, it turns out. One morning at SXSW I read that embedded Linux soon will run in something like 60,000 cash registers at Home Depot. It's a big story, but mostly a technical one. Does Home Depot give a damn about Linux as a cause? Or about the lawmaking that threatens to turn the Net into nothing more than a backbone for industrial-grade commerce, plus a bunch of culverts for moving "content" stamped and sanitized by ubiquitous digital content management? I kind of doubt it.

The other two cultures are the geeks and the entertainment industry, what Larry Lessig and others like to characterize geographically as Silicon Valley and Hollywood.

The geeks built the Net and want to keep it free. Hollywood wants to control it. That's the basic conflict. Since the beginning, the geeks have had resolute faith in the Net's ability to resist control by government and commercial interests. Geeks interpret attempts at control as mere problems the Net will naturally route around. The same goes for Linux, which has proven handy

Yet Google, for all the good it does in the world, is a colossus that plays a huge role in countless lives, yet has almost zero accountability to individual human beings. By design.

for extending the Net upward into the operating system and outward into the world. That geek philosophy was manifest in John Perry Barlow's "A Declaration of the Independence of Cyberspace", even six years after it was written in February 1996. The provocations have changed, but the sides remain the same. And, like I said, those sides dwell in our own heads.

Turns out it wasn't just Hollywood. Geeks who succeeded went both Hollywood and Wall Street—also away from themselves as ordinary folks like you and me. Google, for example, does lots of good in the world. I know lots of people there, and they're all very nice—including the founders, who I've met and like. Yet Google, for all the good it does in the world, is a colossus that plays a huge role in countless lives, yet has almost zero accountability to individual human beings. By design.

A good example comes from a recent post at BoingBoing titled "A Plea for Help From Someone Being Casually Crushed Under Google's Heel" (https://bbs.boingboing.net/t/a-plea-for-help-from-someone-being-casually-crushed-under-googles-heel/76666). It's by a couple who says say they are paying customers of Google, for storage of Google Apps data; yet Google has yanked their accounts for some machine reason no humans can be found to fix. My wife and I are in the same boat with Gmail. Something went wrong a few months back, and Gmail barely works any more for either of us. Fortunately, Gmail was a secondary system for us, but we feel the pain.

An irony here is that Google prides itself on knowing people extremely well. Yet even that has a Matrix-like inhumanity to it.

We see the same thing with Facebook. Mark Zuckerberg is another super-smart geek who now runs a giant company involved in more than a billion human lives, with almost no accountability to the individuals who

depend on the company's services.

Like Google, Facebook is a B2B business that sells data about its consumers to its actual customers, which are corporations. So, while Facebook talks one game—about doing good things for individuals—it plays another. For example, at the latest F8 conference, in April, Mark Zuckerberg said this (http://money.cnn.com/2016/04/12/technology/facebook-messenger-bots/index.html):

> Now that Messenger has scaled, we're starting to develop ecosystems around it. And the first thing we're doing is exploring how you can all communicate with businesses.
>
> You probably interact with dozens of businesses every day. And some of them are probably really meaningful to you. But I've never met anyone who likes calling a business. And no one wants to have to install a new app for every service or business they want to interact with. So we think there's gotta be a better way to do this.
>
> We think you should be able to message a business the same way you message a friend. You should get a quick response, and it shouldn't take your full attention, like a phone call would. And you shouldn't have to install a new app.

Let's pause here. It looks like he's going to give us a better way to talk to businesses, right? Maybe a new way to issue a call for help, or to send out a request for a plumber or a licensed electrician—something that helps us deal with the typical pains of being a customer of many products and services in the real world. Now, let's hit Play again:

> So today we're launching Messenger Platform. So you can build bots for Messenger.

Who is the "you" he's talking about here? It's not the "you" who wants a better way to talk to businesses. It's developers working for businesses that doesn't want human beings to talk to customers, a decision they already made

by replacing customer service people with apps customers install. Zuck again:

> And it's a simple platform, powered by artificial intelligence, so you can build natural language services to communicate directly with people. So let's take a look.
>
> CNN, for example, is going to be able to send you a daily digest of stories, right into messenger. And the more you use it, the more personalized it will get. And if you want to learn more about a specific topic, say a Supreme Court nomination or the zika virus, you just send a message and it will send you that information.

And thus he obeys all three of Zuboff's Laws.

And he's not the only one misdirecting attention away from surveillance. Nearly every story about Facebook's new bot thing focuses on lost jobs or the threatened app marketplace. Not on the loss of freedom.

In "'Bot' is the wrong name...and why people who think it's silly are wrong" (https://medium.com/lightspeed-venture-partners/bot-is-the-wrong-name-and-why-people-who-think-they-are-silly-are-wrong-dc0c0b76ae18#.mkl3r99xo), Aaron Batalion says all kinds of functionality now found only in apps will move to Messenger. "In a micro app world, you build one experience on the Facebook platform and reach 1B people."

Nobody suggests building one method for connecting a billion people to

every business they deal with—which, in case you don't know by now, is what I've been evangelizing with ProjectVRM (http://blogs.harvard.edu/vrm) for the last ten years. Because it's easier to think big than think right. And by "right" I mean free.

So where are we headed here?

"In The End of the Internet Dream" (https://backchannel.com/the-end-of-the-internet-dream-ba060b17da61#.wf135a2n1), Jennifer Granick writes,

> Twenty years from now:
>
> - You won't necessarily know anything about the decisions that affect your rights, like whether you get a loan, a job, or if a car runs over you. Things will get decided by data-crunching computer algorithms and no human will really be able to understand why.
>
> - The Internet will become a lot more like TV and a lot less like the global conversation we envisioned 20 years ago.
>
> - Rather than being overturned, existing power structures will be reinforced and replicated, and this will be particularly true for security.
>
> - Internet technology design increasingly facilitates rather than defeats censorship and control.

And it will all be done on Linux.

Remember how Zuboff's Third Law said "In the absence of countervailing restrictions and sanctions"?

It's our job to correct that absence.■

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

RETURN TO CONTENTS