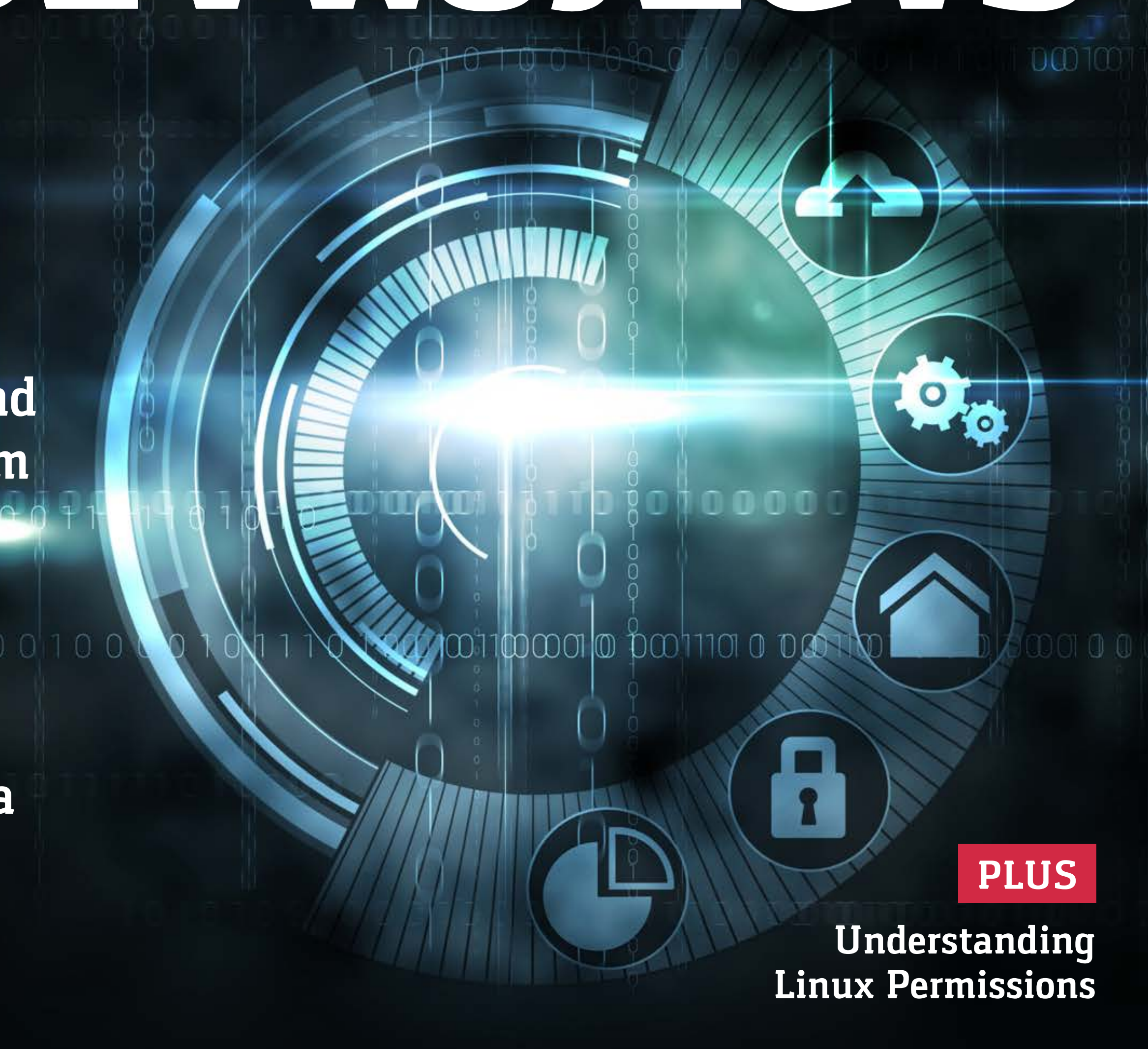
LINUX JOURNAL™

Since 1994: The Original Magazine of the Linux Community

MAY 2015 | ISSUE 253 | www.linuxjournal.com

HOW TO: Home Automation with Raspberry Pi

COOL PROJECTS



BUILD A Vehicle Monitoring and Control System

CREATE A Safe to Store Your Sensitive Data

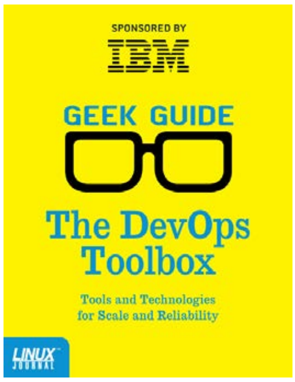
PLUS Understanding Linux Permissions

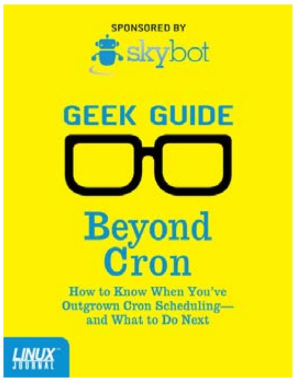
Send SMS Notifications to Your Smart Watch

Working with Django Models and Migrations

Modify the Boot Menu—Libreboot on an X60

WATCH: ISSUE OVERVIEW





The Advanced Foundation for Your Success

12

SUSE Linux Enterprise 12

+ Increase Uptime
+ Improve Operational Efficiency
+ Accelerate Innovation

www.suse.com/sle12

SUSE

CONTENTS

MAY 2015
ISSUE 253

COOL PROJECTS

FEATURES

54 Home Automation with Raspberry Pi
Have a home with a life of its own.
Bharath Bhushan Lohray

68 Embed Linux in Monitoring and Control Systems
Linux touches the real world.
Rick Brown

ON THE COVER
- How To: Home Automation with Raspberry Pi, p. 54
- Build a Vehicle Monitoring and Control System, p. 68
- Create a Safe to Store Your Sensitive Data, p. 80
- Send SMS Notifications to Your Smart Watch, p. 38
- Working with Django Models and Migrations, p. 30
- Modify the Boot Menu—Libreboot on an X60, p. 44
- Plus: Understanding Linux Permissions, p. 24

INDEPTH

80 Hacking a Safe with Bash
How to script a safe to keep your sensitive data secure.
Adam Kosmin

COLUMNS

30 Reuven M. Lerner's At the Forge
Django Models and Migrations

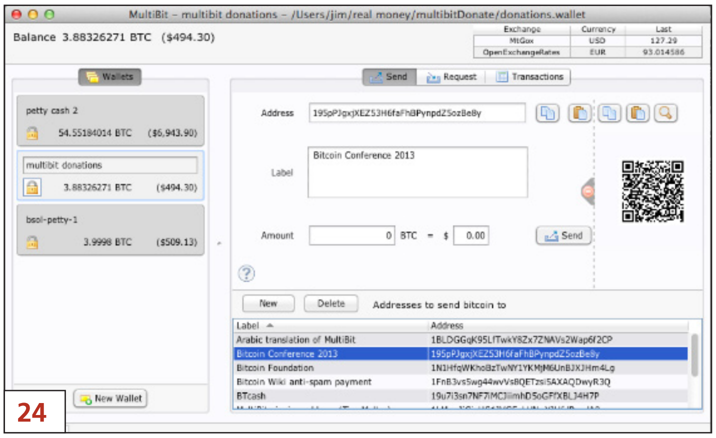
38 Dave Taylor's Work the Shell
System Status as SMS Text Messages

44 Kyle Rankin's Hack and /
Libreboot on an X60, Part III: Modify the Boot Menu

90 Doc Searls' EOF
An Easy Way to Pay for Journalism, Music and Everything Else We Like

IN EVERY ISSUE

8 Current_Issue.tar.gz
10 Letters
16 UPFRONT
28 Editors' Choice
50 New Products
93 Advertisers Index


24

MultiBit – multibit donations – /Users/jim/real money/multibitDonate/donations.wallet

Balance 3.88326271 BTC ($494.30)

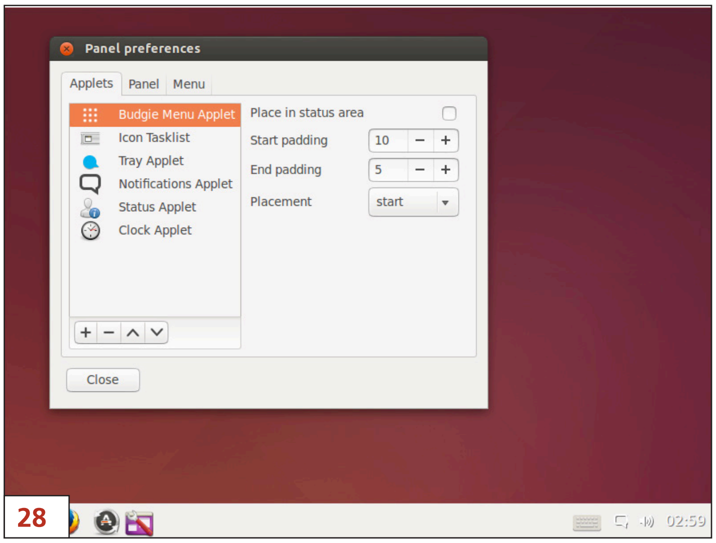
Exchange	Currency	Last
MtGox	USD	127.29
OpenExchangeRates	EUR	93.014586

petty cash 2 54.55184014 BTC ($6,943.90)
multibit donations 3.88326271 BTC ($494.30)
bsol-petty-1 3.9998 BTC ($509.13)

Address 195pPJgxjXEZ53H6faFhBPynpdZ5ozBe8y
Label Bitcoin Conference 2013
Amount 0 BTC = $ 0.00

Addresses to send bitcoin to

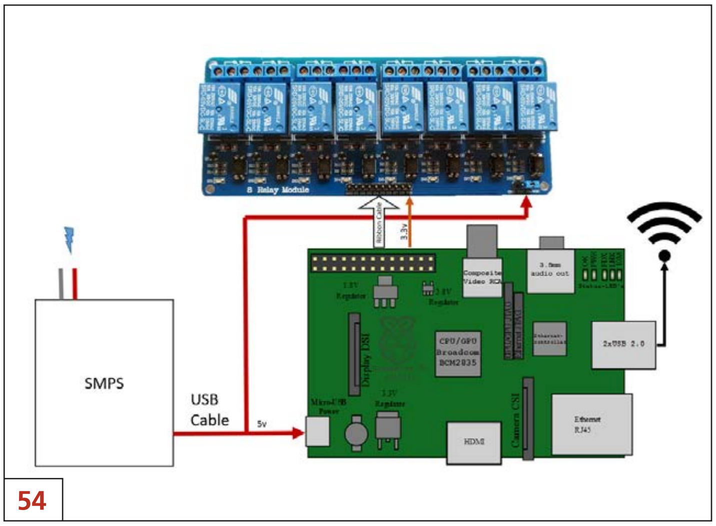
Label	Address
Arabic translation of MultiBit	1BLDGGqK95LfTwkY8Zx7ZNAVs2Wap6f2CP
Bitcoin Conference 2013	195pPJgxjXEZ53H6faFhBPynpdZ5ozBe8y
Bitcoin Foundation	1N1HfqWKhoBzTwNY1YKMjM6UnBJXJHm4Lg
Bitcoin Wiki anti-spam payment	1FnB3vs5wg44wvVs8QETzsi5AXAQDwyR3Q
BTcash	19u7i3sn7NF7iMCJiimhD5oGFfXBLJ4H7P


28

Panel preferences

Budgie Menu Applet
Icon Tasklist
Tray Applet
Notifications Applet
Status Applet
Clock Applet

Place in status area
Start padding 10
End padding 5
Placement start


54

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Belltown Media, Inc., 2121 Sage Road, Ste. 395, Houston, TX 77056 USA. Subscription rate is $29.50/year. Subscriptions start with the next issue.

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

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

President	Carlie Fairchild publisher@linuxjournal.com
Publisher	Mark Irgang mark@linuxjournal.com
Associate Publisher	John Grogan john@linuxjournal.com
Director of Digital Experience	Katherine Druckman webmistress@linuxjournal.com
Accountant	Candy Beauchamp acct@linuxjournal.com

***Linux Journal* is published by, and is a registered trade name of, Belltown Media, Inc.**
PO Box 980985, Houston, TX 77098 USA

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

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

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

Bases loaded?
Who's in *your* Lineup?

Minnowboard Max

Netgate **Rookie**

Ht: 1 in. L: 4 in. W: 3 in. Wt. 5oz.

League Statistics

CPU: Intel® Dual Core Atom™ E3825 1.33GHz
RAM: 2GB
Storage: MicroSD Slot
I/O: (1)Gb Ethernet, SATA2, (1)USB 3.0, (1)USB 2.0, micro HDMI, expansion headers
Integrated Intel HD Graphics, 1920x1080 max
Open Source hardware. Pick your OS: Windows® 8.1, Android 4.4, FreeBSD 10, Linux , Yocto™ Compatible

RCC-VE 4860

Netgate **MVP**

Ht: 1.5 in. L: 6.8 in. W: 7 in. Wt. 17 oz.

League Statistics

CPU: Intel Quad Core Atom C2558 "Rangeley" 2.4GHz
RAM: 8GB
Storage: 4GB eMMC onboard
I/O: (6)Gb Intel Ethernet, SATA2, mSATA, (2) mPCI sockets (one with microSIM), USB
Fanless, low-power, NFV edge device
Ships with CentOS 7

APU1D4

Netgate **Southpaw**

Ht: 1 in. L: 6.3 in. W: 6.5 in. Wt. 16 oz.

League Statistics

CPU: AMD® G Series Dual Core "Bobcat" T40E APU 1Ghz
RAM: 4 GB
Storage: mSATA slot, SD card slot
I/O: (3)Gb Ethernet, USB, (1)SATA, DB9 serial, (2) miniPCIe sockets (one with micro sim)
Runs many flavors of BSD and Linux

RCC-VE 2440

Netgate **Slugger**

Ht: 1.5 in. L: 6.8 in. W: 7 in. Wt. 17 oz.

League Statistics

CPU: Intel Dual Core Atom C2358 "Rangeley" 1.7GHz
RAM: 4GB
Storage: 4GB eMMC onboard
I/O: (4)Gb Intel Ethernet, SATA2, mSATA, (2) mPCI sockets (one with microSIM), USB
Fanless, low-power internet gateway system
Ships with CentOS 7

C2758

Netgate **Heavy Hitter**

Size: 1U Rackmount Wt. TBA

League Statistics

CPU: Intel Eight Core Atom C2758 "Rangeley" 2.4GHz
RAM: 8GB RAM
Storage: 64GB eMMC onboard. Yes, 64 GB!
I/O: (6)Gb Intel Ethernet, SATA2, mSATA, (2) mPCI sockets (one with microSIM), USB
Flexible low-power networking platform
Arrives with CentOS 7 and a low noise fan!

BeagleBone Black

Netgate **The Mascot**

Ht: <1in. L: 3.5in. W: 2.5in. Wt. 2oz.

League Statistics

CPU: AM3358 1Ghz ARM® Cortex®-A8
RAM: 512MB
Storage: 4GB eMMC, microSD slot
I/O: (1)10/100 Ethernet, USB, HDMI, (2)46-pin headers, 3D graphics accel.
100% Open Source. Runs FreeBSD, Android, Angstrom, Debian, Nintendo, Beaglenmt, etc!

Netgate®

7212 McNeil Drive Suite 204 Austin, TX 78729 +1.512.646.4100

SHAWN POWERS

Robotic Sharks with Laser Eyes

I love the Cool Projects issue. Don't get me wrong, most issues of *Linux Journal* are full of cool things to do, but this month, we do it just because of the cool factor. As you can imagine, no Cool Projects issue is complete without a Raspberry Pi article, and this one is particularly awesome. But let me start off with a bit about our columns.

Reuven M. Lerner continues his series on Django, and this month, he covers migrations and updating databases. If you're a developer looking for a framework to start with, or if you're already using Django and want to learn more, Reuven's series is a great way to begin. Dave Taylor follows Reuven with a new topic this month (you might remember Dave was working on a word search project in his last column). In this issue, he takes on the topic of how to make your shell scripts send text messages. It's a great way to get instant notifications to users, which isn't usually possible from inside a script.

I describe a couple cool programs in this month's upfront section, starting with Budgie. If you like the simplicity of the Chrome desktop interface, but prefer a full-blown Linux system underneath, Budgie is perfect. I also talk about the intricacies of the Linux permissions system and even a few Bitcoin clients. It's hard to beat the cool factor of Kyle Rankin's column this month, however, as he continues his series on Libreboot. People have been installing open source on hard drives for years, but with

VIDEO:
Shawn Powers runs through the latest issue.

Kyle's assistance, you will learn to install the open-source BIOS replacement as well!

Be sure to check out Bharath Bhushan Lohray's article for an incredible home automation project. Starting from scratch with a Raspberry Pi, some relays and some wiring, Bharath walks through the steps of using the GPIO pins to manage multiple systems. Although it's certainly possible to buy one of the many embedded home automation kits available, starting from scratch allows for some serious customization and infinite programability. If you've been struggling to choose a brand of home automation systems to try, perhaps after reading this article, that question will become moot!

Rick Brown describes another awesome project, but this time it integrates with existing systems. Specifically, he explains how he connected a Linux system to a vehicle to get real-time operation data. Rick also shows how to design a display for the information, so that you're not grepping log files while driving!

In past issues, you have learned how to do basic encryption with Linux tools in order to keep your sensitive data safe. This month, Adam Kosmin goes much further and describes his complete system for keeping data secure. Using freely available tools and a handful of scripts and methods, he shows how to integrate secure encryption into your daily routine. If you want to encrypt your data, but find it complex and frustrating, be sure to read Adam's article.

The Cool Projects issue is a favorite of mine year after year. Not only is it a chance to start working on those ideas you've been putting off for months, but it's also a great way to learn while playing. I learned more about how keyboards function while making my MAME cabinet than ever before or since. As a kid who took apart everything I got my hands on, the Cool Projects issue is an awesome way to learn how to put a few things back together! Whether you love projects or just want some tech tips, product announcements and programming lessons, this issue of *Linux Journal* should provide lots of entertainment and education.■

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.

letters

LINUX JOURNAL
Since 1994: The Original Magazine of the Linux Community
LUCI4HPC: the Lightweight, User-Friendly Cluster Management Tool
APRIL 2015 | ISSUE 252 | www.linuxjournal.com
HIGH-PERFORMANCE COMPUTING
JAILHOUSE Real-Time Security-Wise Virtualization
PLUS
Explore the Concepts of STDIN, STDOUT and STDERR
Write a Script to Generate Word Searches
HOW-TO: Installing Libreboot on an x60
CREATE Dynamic Content with Django Templates
WATCH: ISSUE OVERVIEW

Non-Linux Foss: MenuMeters?

Regarding Shawn Powers' Non-Linux FOSS Upfront piece in the March 2015 issue: MenuMeters? Hmm... that's pretty much the same thing as GKrellM (which has been around since forever-ish). GKrellM not only displays all the stuff on your working system but also on any of your networked servers with little, if any, fooling around. I've used it for years to keep an eye on the headless database servers sitting in the closet mumbling to themselves. I've also installed it on friends' Windows machines for quick and dirty troubleshooting and cores at 100% (ain't Windows wonderful, eh?). I don't know for sure, but I'll bet there's a version for Mac? Sometimes the oldies but goodies are a pretty good choice, methinks.

—Thomas

Thomas, yes, GKrellM is awesome, but there's not a readily downloadable version for OS X. (There is for Windows, however.) Honestly, the "Non-Linux FOSS" piece is one of the toughest for me to write, because most FOSS software that is available for Windows and/or OS X is also available for Linux. Finding non-Linux FOSS is...challenging!—Shawn Powers

Runtime "Stuff Happens"—Linux, Tell Me about It

I was pleased and interested to read about GNOME's pending enhancements to its desktop notification component(s). With so many things going on behind the scenes of a contemporary workstation, it is important to have effective (aka, "the right things") and efficient (aka, "things done well") ways to tell the end user that something happened. One might also hope that any such "telling" might also offer clues about what an end user might do as a corrective action or workaround for whichever "stuff happened".

That's nice—where is the admin and end-user dispatch table that says, in part, "when EVENTx happens, do ACTIONx", etc.? Why isn't this dispatch table as clean and obvious as /etc/anacrontab?

"Another tool suite" called systemd is causing a lot of chatter trying to do mostly the same thing—"do ACTION when EVENT happens":

- FancyPrinter just dropped off-line. Show a dialog box to announce that fact.

- eSATA or USB drive connected. "Do a little dance ..." make the content available.

Decades ago, I worked on TOPS-10/TOPS-20 mainframes. There was a utility suite that 1) handled all of the notifications; 2) routed notifications to end users, operators or others who were on the lists to care about those notices; and 3) provided a per-user portal for trying to recover, correct or resolve whatever was going on. I'm all for innovation, but isn't it time we re-solved (sic) this feature set and move on?
—Dan

Scale13x

I was listening to Kyle Rankin's "Tails" talk at Scale13x. Sitting in the third row, I heard a call for assistance on my two-way radio in another room; I was an A/V volunteer. I wanted to get up and sneak into the back of the room to exit so I didn't disturb the great presentation. Wow! When I turned around, the room was jam-packed with people sitting all over the aisle floors. I had to walk right in front of the room to exit. I hope Kyle didn't think I was bored with his presentation. I did have a shirt on that had "A/V Volunteer" written on it in big letters. I came back after a few minutes and Kyle had finished, and there was a crowd of people asking him questions. I hope he made it out of there okay. Most Linux folks are friendly though, and we did have a sense of community at the conference.
—Roman

Quizzes

I thought it would be interesting to share these quizzes with you and your readers. At http://dcjtech.info/quizzes, I made some interactive quizzes about Linux that people

can take for fun, practice, school or interviews. My Web site is ad-free, so it should not annoy visitors.
—Devyn Collier Johnson

Cool, thanks for the link, Devyn!
—Shawn Powers

Linux Newbie Request

I really appreciate Shawn Powers' enthusiasm and his approach to teaching Linux in the CBT Nuggets videos he's done for the LPIC 1 certification.

Currently, I'm in a software support role in LDN with descent bash knowledge, and my next step would be to move on to a system administrator career. Can you recommend any Web sites with LPI practice exams?

The Internet is packed with Web sites, but forum reviews are quite concerning, as many are scams, outdated, crippled with errors or have no support.
—Patrick

I've found the folks at the LPI to be great to work with. I'd recommend contacting them directly (even via Twitter) for advice on practice exams.—Shawn Powers

Initializing and Managing Services in Linux

Thank you for the very informative and timely "Managing Services in Linux: Past, Present and Future" article by Jonas Gorauskas in the March 2015 issue of *LJ*. The following is a URL of a comparison chart of different init systems (yes, it's Gentoo-centric): http://wiki.gentoo.org/wiki/Comparison_of_init_systems.
—Richard

Individual Contribution to Open-Source Project

Have you ever chronicled the experience of a software developer contributing to an open-source project for the first time? Some of the highlights could be finding a project, finding something to work on in the project, submitting work, responding to rejected work and first accepted work. Then as the developer becomes experienced, contrast getting involved in more projects, so as to describe how valuable the lessons learned were going forward.
—Jon Redinger

That might be an interesting story. If you're offering, submit a pitch to ljeditor@linuxjournal.com. See http://linuxjournal.com/author *for more information.—Shawn Powers*

systemd

I always hated the init solution—System V Init is definitely something of an old technology. Therefore, I welcome systemd. It promises to auto-restart; support "new" technologies like USB, Bluetooth, storage devices and so on; and to support future technologies as well. It's a modern approach to tackle the current and future problems in services. There always are initial start problems with new technologies, but we shouldn't hold on to past technologies and lose the technology advantage we have with GNU/Linux.

"Managing Services in Linux: Past, Present and Future" by Jonas Gorauskas in the March 2015 issue was a good article! I advise Linus Torvalds to implement systemd as soon as possible in the mainstream kernel as the default service manager and drop the older technologies in favor of systemd. You could add a compatibility modus with the older one if it is necessary, but the way forward is systemd. The fact that it's implemented in Debian and Red Hat is already proof for me that it is good technology. Maybe this is a good time to change to a higher version—for example, 3.4 or 4.0—to indicate a major improvement in the kernel?

—Patrick Op de Beeck

Question for Dave Taylor

I was trying to put together a simple script to recurse through a directory tree:

```
#/bin/bash

function recurse_dir()
{
    for f in * ; do
            #do stuff ;
            if [ -d "${f}" ] ; then
                    pushd "${f}" ;
                    recurse_dir ;
                    popd ;
            fi ;
    done ;
}

pushd ~/dir ;
recurse_dir ;
popd ;
```

When running this script, I got errors about invalid options. It turned out that some of the subdirectories had leading dashes (-) that were being interpreted as options. I remembered that it is possible to add a dash, or double dash, to turn off further option processing. Have you any other tips for dealing with difficult filenames? I think it is possible to have quotes in some, and that has caused me problems in the past.

—Jeremy

***Dave Taylor replies:** That's an interesting script you're trying to build there, Jeremy. I'm not clear why you're using push/pop as you traverse the directories too. Why not just have* `cd ${f}` *followed by* `cd ..` *to get back up a level and simplify things?*

In terms of difficult filenames, yes, Linux wasn't really written to deal with filenames that start with a dash, have a space or other punctuation, etc. The best you can do is experiment to see if the commands you're using accept -- as a way to delineate that you're done with command arguments, and quote the directory names themselves, as you've done.

Good luck with this!

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

PHOTO OF THE MONTH
Remember, send your Linux-related photos to ljeditor@linuxjournal.com!

LINUX JOURNAL

At Your Service

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

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

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

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

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

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

O'REILLY®

Velocity

CONFERENCE

BUILDING A FASTER, STRONGER BUSINESS

May 27–29, 2015 · Santa Clara, CA

"I saved our company close to a half a million dollars in just 6 months, that is the kind of ROI you get from attending."

—**Phil Chen,**
DevOps Advisor, Rev Software

Save 20% use code LINUXJ

diff -u
WHAT'S NEW IN KERNEL DEVELOPMENT

One ongoing question kernel developers face is the best way to delete data so no one else can recover it. Typically there are simple tools to undelete files that are deleted accidentally, although some filesystems make this easier than others.

Alexander Holler wanted to make it much harder for anyone to recover deleted data. He didn't necessarily want to outwit the limitless resources of our governmental overlords, but he wanted to make data recovery harder for the average hostile attacker. The problem as he saw it was that filesystems often would not actually bother to delete data, so much as they would just decouple the data from the file and make that part of the disk available for use by other files. But the data would still be there, at least for a while, for anyone to recouple into a file again.

Alexander posted some patches to implement a new system call that first would overwrite all the data associated with a given file before making that disk space available for use by other files. Since the filesystem knew which blocks on the disk were associated with which files, he reasoned, zeroing out all relevant data would be a trivial operation.

There were various objections. **Alan Cox** pointed out that hard drives have become so smart these days that it's hard to know exactly what they're doing in response to a given command. As he put it, "If you zero a sector [the disk is] perfectly entitled to set a bit in a master index of zeroed sectors, and not bother actually zeroing the data at all." Alan said that the disk simply had to accept user inputs and return the correct outputs, and everything happening behind the curtain was entirely up to the hardware manufacturer.

Russ Dill pointed out that a lot of user programs also made it more difficult to know exactly where a file's data was on disk. The **vim** program, for example, created temporary backup files, as did many other programs.

There was not much support for Alexander's patch. But I imagine the ability to delete files permanently will come up again at some point. For kernel features though, the goal always tends to be doing a thorough job that, in this case at least, would indeed outwit the government overlords' efforts to recover the data.

There's an ongoing debate about **cgroups**, between the group of people who want to implement cool features and the group of people who want to ensure security. The security people always win, but the debate is rarely simple, and sometimes a cool feature just needs to be rotated a little in order to match the security requirements.

For example, **Aleksa Sarai** wanted to add a cool and useful feature limiting the number of open processes allowed in a given virtual machine. This would prevent certain types of denial-of-service attacks. The problem, as pointed out by **Tejun Heo**, was that an open process limit doesn't correspond to any actual limit on a real system. And, there's a strong reluctance to put limits on anything that's not a true resource, like RAM, disk space, number of CPUs and so on.

On the other hand, as **Austin Hemmelgarn** said, process IDs (PIDs)

Powerful: Rhino

Rhino M4800/M6800
- Dell Precision M6800 w/ Core i7 Quad (8 core)
- 15.6"-17.3" QHD+ LED w/ X@3200x1800
- NVidia Quadro K5100M
- 750 GB - 1 TB hard drive
- Up to 32 GB RAM (1866 MHz)
- DVD±RW or Blu-ray
- 802.11a/b/g/n
- Starts at $1375
- E6230, E6330, E6440, E6540 also available

- High performance NVidia 3-D on an QHD+ RGB/LED
- High performance Core i7 Quad CPUs, 32 GB RAM
- Ultimate configurability — choose your laptop's features
- One year Linux tech support — phone and email
- Three year manufacturer's on-site warranty
- Choice of pre-installed Linux distribution:

Tablet: Raven

Raven X240
- ThinkPad X240 by Lenovo
- 12.5" FHD LED w/ X@1920x1080
- 2.6-2.9 GHz Core i7
- Up to 16 GB RAM
- 180-256 GB SSD
- Starts at $1910
- W540, T440, T540 also available

Rugged: Tarantula

Tarantula CF-31
- Panasonic Toughbook CF-31
- Fully rugged MIL-SPEC-810G tested: drops, dust, moisture & more
- 13.1" XGA TouchScreen
- 2.4-2.8 GHz Core i5
- Up to 16 GB RAM
- 320-750 GB hard drive / 512 GB SSD
- CF-19, CF-52, CF-H2, FZ-G1 available

EmperorLinux
...where Linux & laptops converge

www.EmperorLinux.com
1-888-651-6686

Model specifications and availability may vary.

were an actual limit on a real system, and Tejun agreed it might make sense to allow them to be limited within a cgroup. And because that could be used to limit the number of open processes, everyone could end up happy. But the feature had to be presented as limiting an actual system resource, rather than limiting a relatively arbitrary characteristic of a system.

The **tracing system** has been showing signs of outgrowing its infrastructure lately, and **Steven Rostedt** recently posted some patches to fix that. Up until now, the tracing directory used **DebugFS**. But as Steven said, tracing needed to create and remove directories, and DebugFS didn't support that. So, tracing had been using various hacks to get around it. Steven's solution was to create a new filesystem called **TraceFS**, specifically for the tracing system.

There were no major objections, but there were some technical obstacles to get past. In particular, Steven discovered that the **perf system** was hard-coded to assume that the tracing system used DebugFS, so that had to be fixed before TraceFS could go into the kernel.

Other issues came up; for example, **Greg Kroah-Hartman** suggested basing TraceFS on **KernFS**, and Steven considered that for a while. But it turned out that KernFS had a lot of cgroup-related complexity that TraceFS didn't need, and **Al Viro** remarked, "It's not a good model for anything, other than an anti-hard-drugs poster ('don't shoot that shit, or you might end up hallucinating *this*')." Ultimately, Steven decided against KernFS.—**ZACK BROWN**

They Said It

If you want to be free, there is but one way; it is to guarantee an equally full measure of liberty to all your neighbors. There is no other.
—Charles Schurz

It's not the things we do in life that we regret on our death bed, it is the things we do not.
—Randy Pausch

Getting fired is nature's way to telling you that you had the wrong job in the first place.
—Hal Lancaster

Little by little, one travels far.
—J.R.R. Tolkien

You can't deny laughter. When it comes, it plops down in your favorite chair and stays as long as it wants.
—Stephen King

Drupal is increasingly the CMS of choice for large scale websites.

Join us at DrupalCon Los Angeles to learn how to scale Drupal for the enterprise and optimize for top performance. Who knows what heroic feats you will accomplish with your new skills? You may just save the day at your organization.

Become a hero at events.drupal.org. Join us in Los Angeles this May 11 - 15, 2015.

DRUPALCON 2015 ★ MAY 11 - MAY 15

Brought to you by the Drupal Association.

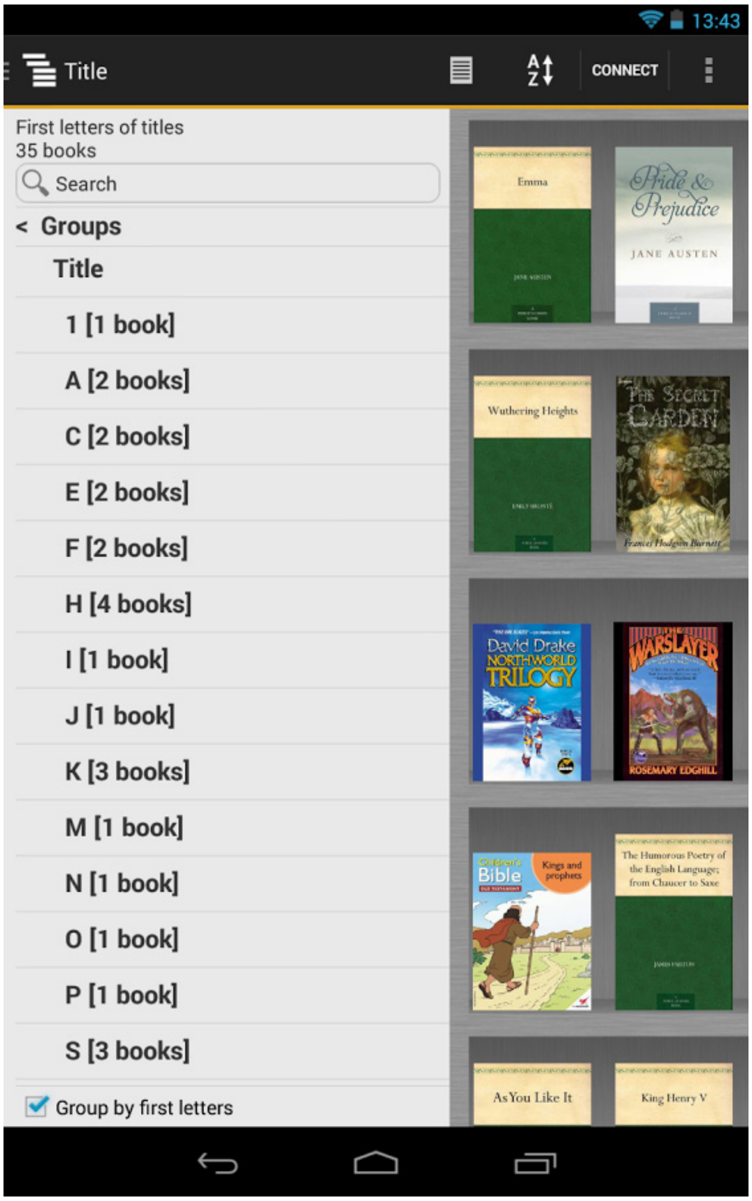
Image credit to Jimmy Baikovicius (jikatu) on Flickr. Some rights reserved.

Android Candy: Every Hero Needs a Sidekick

I've touted the awesomeness of Calibre in the past (http://www.linuxjournal.com/content/calibre-cloud). And although the Web-based calibre2opds still is an awesome way to access your eBook library, using a native Android app is even smoother. If you have your Calibre library on your local network, using Calibre Companion ($3.99 in the Google Play Store), your Android device connects to your library like a device connected via USB. It's possible to load books directly onto your device without syncing your entire collection into the cloud!

I admittedly still use calibre2opds in combination with Dropbox to make my library accessible remotely. But, if you're concerned about your books being on the Web, Calibre Companion is a local network solution. Check it out today at https://play.google.com/store/apps/details?id=com.multipie.calibreandroid.

—SHAWN POWERS



(Image from the Google Play Store)

peer1 hosting

Where every interaction matters.

break down
your innovation barriers

power your business to its full potential

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

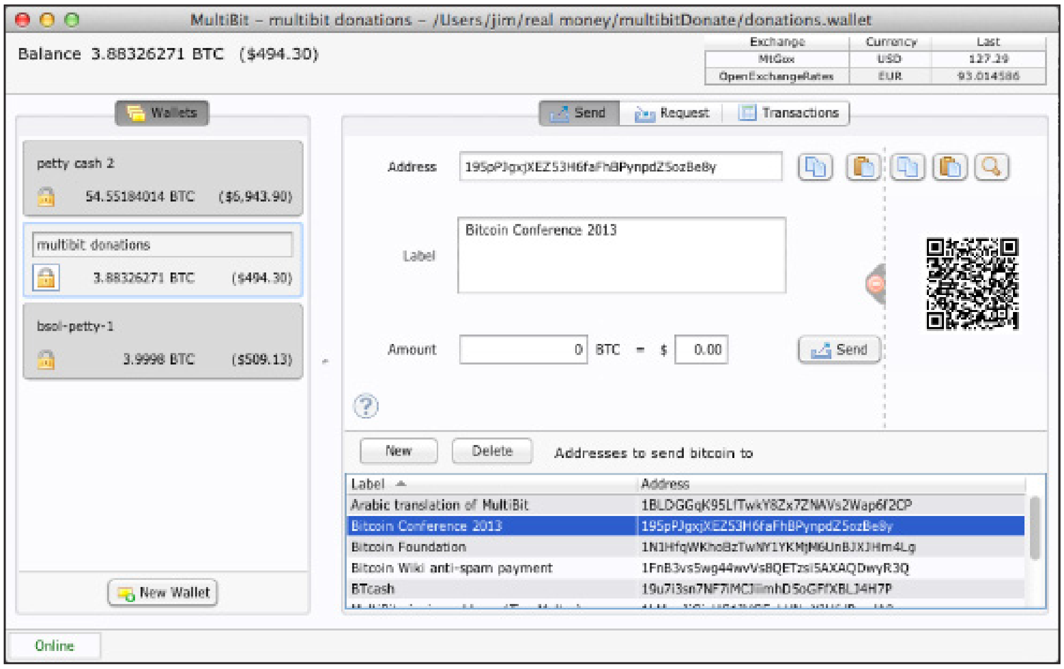
Peer 1 Hosting powers your business with our wholly owned FastFiber Network™, global footprint, and offers professionally managed public and private cloud solutions that are secure, scalable, and customized for your business.

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

Want more on cloud?
Call: 844.855.6655 | go.peer1.com/linux | Vew Cloud Webinar:

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

Non-Linux FOSS: All the Bitcoin, None of the Bloat



MultiBit – multibit donations – /Users/jim/real money/multibitDonate/donations.wallet

Balance 3.88326271 BTC ($494.30)

Exchange	Currency	Last
MtGox	USD	127.29
OpenExchangeRates	EUR	93.014586

Wallets

- petty cash 2 — 54.55184014 BTC ($6,943.90)
- multibit donations — 3.88326271 BTC ($494.30)
- bsol-petty-1 — 3.9998 BTC ($509.13)

Send | Request | Transactions

Address: 195pPJgxjXEZ53H6faFhBPynpdZ5ozBe8y

Label: Bitcoin Conference 2013

Amount: 0 BTC = $ 0.00

Addresses to send bitcoin to

Label	Address
Arabic translation of MultiBit	1BLDGGqK95LfTwkY8Zx7ZNAVs2Wap6f2CP
Bitcoin Conference 2013	195pPJgxjXEZ53H6faFhBPynpdZ5ozBe8y
Bitcoin Foundation	1N1HfqWKhoBzTwNY1YKMjM6UnBJXJHm4Lg
Bitcoin Wiki anti-spam payment	1FnB3vs5wg44wvVsBQETzsi5AXAQDwyR3Q
BTcash	19u7i3sn7NF7iMCJimhD5oGFfXBLJ4H7P

Online

I love Bitcoin. Ever since I first discovered it in 2010 and mined thousands of them, I've been hooked on the technology, the concept and even the software. (Sadly, I sold most of those thousands of Bitcoin when they were less than a dollar. I'm still kicking myself.) One of the frustrations with using Bitcoin, however, is that the blockchain has gotten so large. It currently weighs in at a little less than 20GB, and it takes about a week to download the first time you do so. There are ways to jumpstart the download with a bootstrap file, but still, it's a huge undertaking to run the standard Bitcoin client.

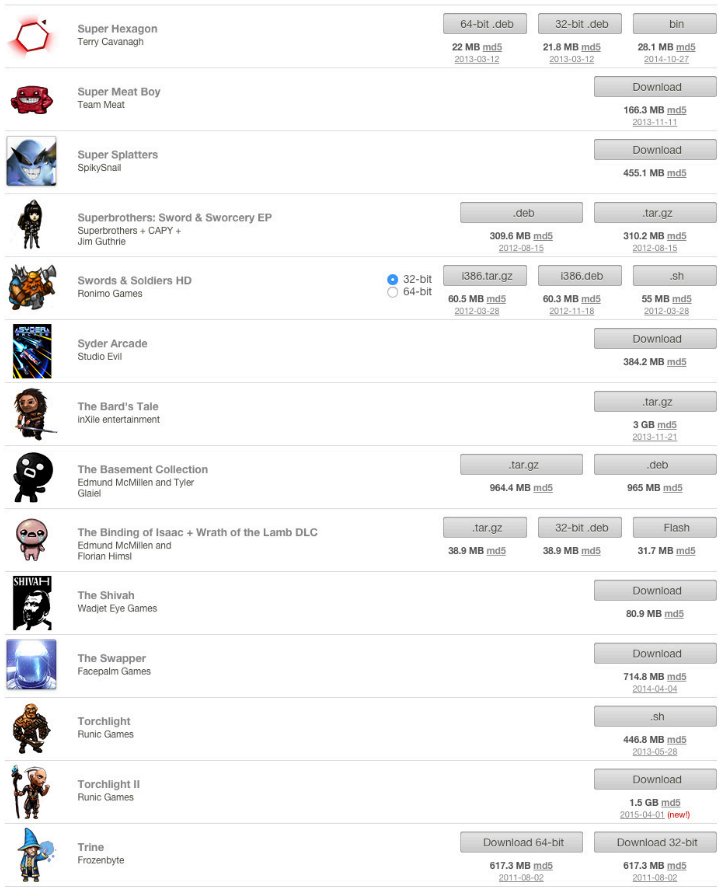
Enter MultiBit.

Although it doesn't have the entire blockchain, MultiBit does have all of the security and encryption the standard Bitcoin client does. Because it reads the Bitcoin network in real time, it takes seconds to sync up as opposed to days. Sending and receiving Bitcoin with MultiBit is fast, efficient and secure. Oh, and it's open-source, multiplatform and under constant development! If you want to run a local Bitcoin client, but don't want to download the entire blockchain, check out MultiBit at http://multibit.org. (Also check out Electrum, a Python-based alternative at http://electrum.org.)

—SHAWN POWERS

My Humble Little Game Collection

I currently have the flu. Not the "sorta queasy" stomach flu, but the full out Influenza with fever, aches and delirium-ridden nightmares. Bouts of crippling illness tend to be my only chance to play games. Thankfully, since I'm such a terrible gamer, being sick doesn't really hurt my skills very much!



Super Hexagon	Terry Cavanagh	64-bit .deb 22 MB md5 2013-03-12	32-bit .deb 21.8 MB md5 2013-03-12	bin 28.1 MB md5 2014-10-27
Super Meat Boy	Team Meat			Download 166.3 MB md5 2013-11-11
Super Splatters	SpikySnail			Download 455.1 MB md5
Superbrothers: Sword & Sworcery EP	Superbrothers + CAPY + Jim Guthrie		.deb 309.6 MB md5 2012-08-15	.tar.gz 310.2 MB md5 2012-08-15
Swords & Soldiers HD	Ronimo Games [✓ 32-bit] 64-bit	i386.tar.gz 60.5 MB md5 2012-03-28	i386.deb 60.3 MB md5 2012-11-18	.sh 55 MB md5 2012-03-28
Syder Arcade	Studio Evil			Download 384.2 MB md5
The Bard's Tale	inXile entertainment			.tar.gz 3 GB md5 2013-11-21
The Basement Collection	Edmund McMillen and Tyler Glaiel		.tar.gz 964.4 MB md5	.deb 965 MB md5
The Binding of Isaac + Wrath of the Lamb DLC	Edmund McMillen and Florian Himsl	.tar.gz 38.9 MB md5	32-bit .deb 38.9 MB md5	Flash 31.7 MB md5
The Shivah	Wadjet Eye Games			Download 80.9 MB md5
The Swapper	Facepalm Games			Download 714.8 MB md5 2014-04-04
Torchlight	Runic Games			.sh 446.8 MB md5 2013-05-28
Torchlight II	Runic Games			Download 1.5 GB md5 2015-04-01 (new!)
Trine	Frozenbyte		Download 64-bit 617.3 MB md5 2011-08-02	Download 32-bit 617.3 MB md5 2011-08-02

Today I was playing *Torchlight II* from Runic Games. I realized it was just as simple for me to run this new, awesome game on Linux using Steam as it is for any Windows user. Thanks to the Humble Bundle sales, I have dozens and dozens of games that aren't cheesy knockoffs, but are actual versions of real games. I think Steam plays a big part in getting more and more games released for Linux, but whatever the reason, it's a great time to be a gamer in an open-source operating system!

The screenshot here is just a section of my past Humble Bundle game purchases. The Humble Bundle (which I've mentioned before) is a great way to get inexpensive games. When you add Steam, you have yet another way to play and buy games that work natively on your Linux system. If you're a gamer, or just have the flu, go download some games:

- http://www.humblebundle.com
- http://www.steampowered.com
- http://www.runicgames.com

—**SHAWN POWERS**

It's Easier to Ask Forgiveness...

...than to understand Linux permissions! Honestly though, that's not really true. Linux permissions are simple and elegant, and once you understand them, they're easy to work with. Octal notation gets a little funky, but even that makes sense once you understand why it exists.

Users and Groups: First I need to address that Linux does have ACL support for file and folder permissions. It's not how things work by default, however, and ACLs were added to address the fine controls needed for some situations. Most Linux systems rely on standard POSIX permissions. That's what I'm covering here.

Every file has an owner and a group membership. If you type `ls -l`, you'll see the owner and group on all the files and folders in your current directory. POSIX permissions allow you to control read, write and execute abilities for the user, group and a third set of users, "other". Using those three aspects on three different sets of users allows for some fairly complex "who can do what" on a filesystem.

Figure 1 shows an example of what a file listing looks like. I've separated the different sections showing which fields are which. Note the first field is usually either a "d" or a "-"; the former appears on directories, and the latter appears on regular files. For files, the permissions make straightforward sense. If the "read" bit is turned on, it means that user (user, group or other) can read the contents of the file. If the "write" bit is set, it can be written to, and if the "execute" bit is set, the file can be

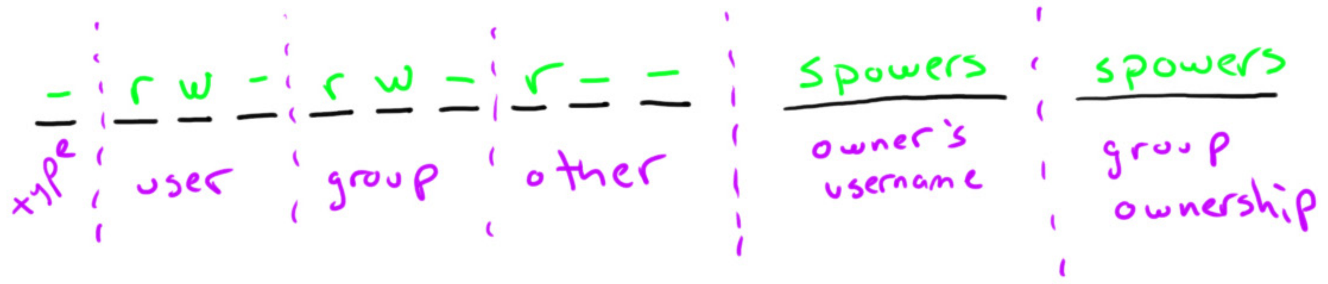


Figure 1. Example File Listing

executed. Execute bits are usually set on binary files or on scripts you want to run directly.

On folders, the permissions work almost the same: read means the ability to see what files are in the folder, and write is the ability to save a file into the folder. Execute is a little less obvious. When the "execute" bit is set, it means the user or group can change into the directory. So the `cd` command will work as long as you have execute access to a folder.

Changing Permissions: Although Linux systems usually come with fairly sane standards for what permissions files are created with, it's often necessary to change them. For that, you use the `chmod` tool. The format is to use:

```
chmod u+rw,g-r,o+rwx file.txt
```

Granted, that's a fairly complex example, but let's break it down. The letters before the + or - are u for user, g for group and o for other. Then you either add or take away (+ or -) whichever aspects you desire. It's not required to specify all three user definitions, and they can be lumped together like this:

```
chmod ugo+rw file.txt
```

It's also possible to leave "ugo" off if you want to change user, group and other at the same time. So the following examples all do the same thing:

```
chmod ugo-rw file.txt
chmod -rw file.txt
chmod u-rw,g-rw,o-rw file.txt
```

Although the "ugo" format is fairly robust, it's sometimes very complicated to craft the string of pluses and minuses to get the exact permissions string you want. That's where octal notation comes into play. It's a little more confusing, but far more convenient once understood.

Octal Notation: In Figure 2, you can see I've shown the numeric value of each permission bit. By simply adding the numbers, it's possible to create any possibility of permissions with three numbers. Figure 3 shows a few examples of how those can be figured out.

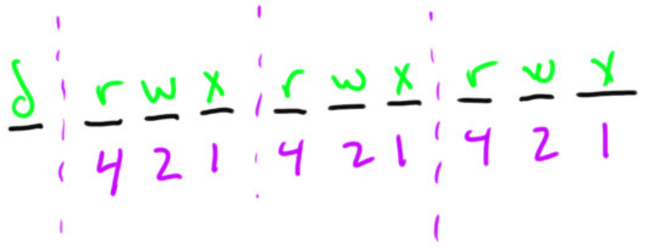


Figure 2. Numeric Value of Each Permission Bit

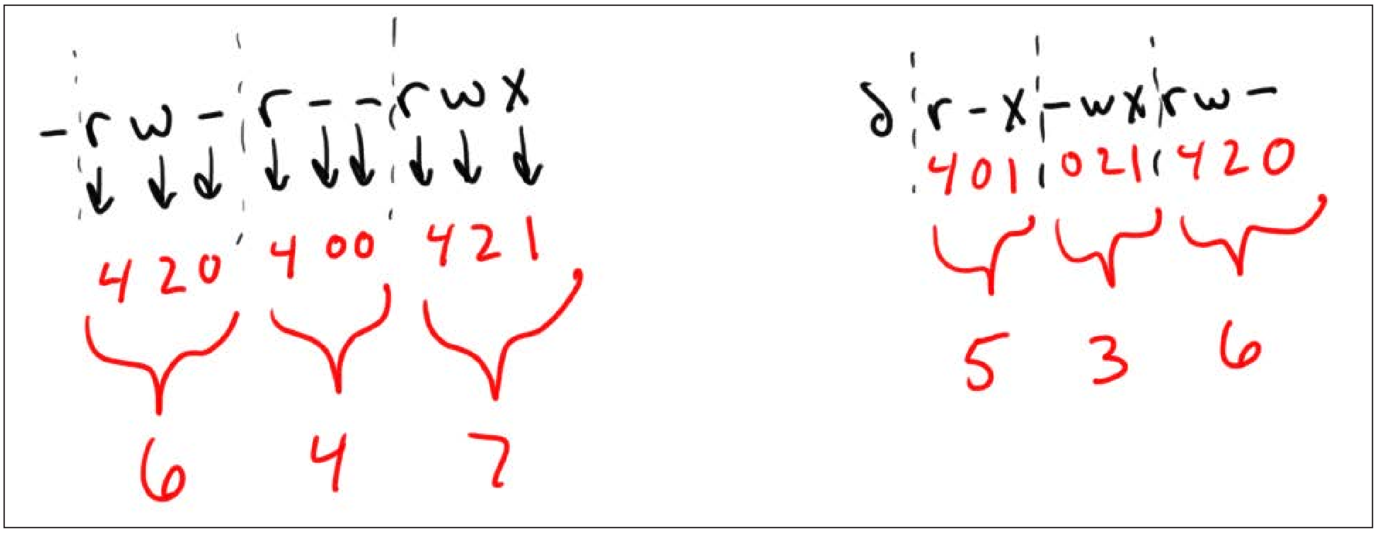


Figure 3. Creating Permissions

So with octal notation, the following two chmod statements are functionally the same:

```
chmod ug+rw,ug-x,o+r,o-wx file.txt
chmod 662 file.txt
```

Although it's not a requirement to use octal notation for setting permissions, it's usually the way it's done.

I urge you to play with `chmod` a bit until you get the results you expect when using octal notation. Then, just for fun, see if you can string together a bunch of pluses and minuses to get the same results!

Linux permissions are simple, elegant and allow for some very flexible file-sharing options on a filesystem. We use them most often when installing Web applications, because the Web server runs as a particular user, and that user (or group) needs to have access to the files it's attempting to serve.—**SHAWN POWERS**

2015

WOMEN IN TECHNOLOGY

SUMMIT

Executive women, entrepreneurs, and technology thought leaders from around the world will converge in Silicon Valley this spring... ***Join Us!***

May 31 - June 2
DoubleTree by Hilton
San Jose, California

witi.com/summit

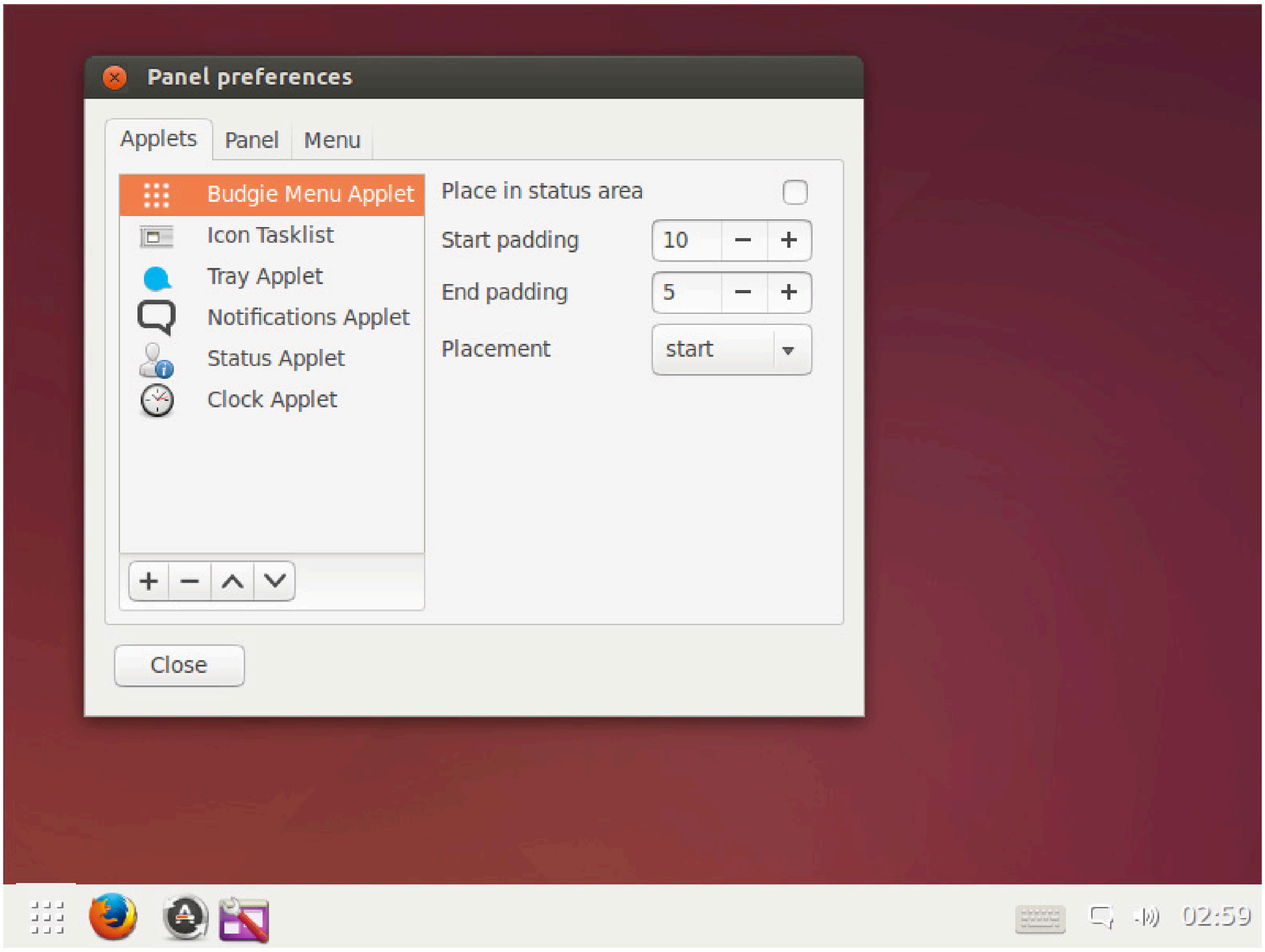
WITI

Special Offer: Use promo code **WOMEN** by May 15th and **Save $100** Off Registration!

Chrome-Colored Parakeets

I personally like Google's Chrome interface. It's simple, fast, elegant and did I mention fast? Unfortunately, I don't like how locked down the Chrome OS is on a Chromebook, nor do I like its total dependence on Google. I also don't like the lack of ability to install Chrome easily on generic hardware. Thankfully, Budgie is here to help.

If you like the simplicity and speed of the Chrome interface, but want a full-blown system underneath that deceptively simple GUI, I urge you



Place in status area ☐
Start padding 10
End padding 5
Placement start

to give Budgie a try. You either can download the Evolve-OS (http://evolve-os.com), or just install the PPA into a standard Ubuntu system. I simply typed:

```
sudo apt-add-repository ppa:evolve-os/ppa
sudo apt-get update
sudo apt-get install budgie-desktop
```

Then log out, and when logging in, choose the Budgie desktop instead of Unity. You'll find a very Chrome-like interface but on top of a full-blown Linux system instead of Chrome! The preferences are fairly simplistic, but the entire interface is designed to get out of the way and let you work. Due to its blazing-fast speed and ease of use, the Budgie Desktop is this month's Editors' Choice. Give it a try today!

—SHAWN POWERS

REUVEN M. LERNER

Django Models and Migrations

Django's migrations make it easy to define and update your database schema.

In my last two articles, I looked at the Django Web application framework, written in Python. Django's documentation describes it as an MTV framework, in which the acronym stands for model, template and views.

When a request comes in to a Django application, the application's URL patterns determine which view method will be invoked. The view method can then, as I mentioned in previous articles, directly return content to the user or send the contents of a template. The template typically contains not only HTML, but also directives, unique to Django, which allow you to pass along variable values, execute loops and display text conditionally.

You can create lots of interesting Web applications with just views and templates. However, most Web applications also use a database, and in many cases, that means a relational database. Indeed, it's a rare Web application that doesn't use a database of some sort.

For many years, Web applications typically spoke directly with the database, sending SQL via text strings. Thus, you would say something like:

```
s = "SELECT first_name, last_name FROM Users where id = 1"
```

You then would send that SQL to the server via a database client library and retrieve the results using that library. Although this approach does allow you to harness the power of SQL directly, it means that your application code now contains text strings with another language. This mix of (for example) Python and SQL can become difficult to maintain and work with. Besides, in Python, you're used to working with objects, attributes and methods. Why can't you access the database that way?

The answer, of course, is that you can't, because relational databases eventually do need to receive SQL in order to function correctly. Thus, many programs use an ORM (object-relational mapper), which translates method calls and object attributes into SQL. There is a well established ORM in the Python world known as SQLAlchemy. However, Django has opted to use its own ORM, with which you define your database tables, as well as insert, update and retrieve information in those tables.

So in this article, I cover how you create models in Django, how you can create and apply migrations based on those model definitions, and how you can interact with your models from within a Django application.

Models

A "model" in the Django world is a Python class that represents a table in the database. If you are creating an appointment calendar, your database likely will have at least two different tables: People and Appointments. To represent these in Django, you create two Python classes: Person and Appointment. Each of these models is defined in the models.py file inside your application.

This is a good place to point out that models are specific to a particular Django application. Each Django project contains one or more applications, and it is assumed that you can and will reuse applications within different projects.

In the Django project I have created for this article ("atfproject"), I have a single application ("atfapp"). Thus, I can define my model classes in atfproject/atfapp/models.py. That file, by default, contains a single line:

```
from django.db import models
```

Given the example of creating an appointment calendar, let's start by defining your Appointment model:

```
from django.db import models

class Appointment(models.Model):
starts_at = models.DateTimeField()
ends_at = models.DateTimeField()
meeting_with = models.TextField()
notes = models.TextField()
def __str__(self):
    return "{} - {}: Meeting with {} ({})".format(self.starts_at,
                                      self.ends_at,
                                      self.meeting_with,
                                      self.notes)
```

Django provides a large number of field types that you can use in your models, matching (to a large degree) the column types available in most popular databases.

Notice that in Django models, you define the columns as class attributes, using a Python object known as a descriptor. Descriptors allow you to work with attributes (such as appointment.starts_at), but for methods to be fired in the back. In the case of database models, Django uses the descriptors to retrieve, save, update and delete your data in the database.

The one actual instance method in the above code is __str__, which every Python object can use to define how it gets turned into a string. Django uses the __str__ method to present your models.

Django provides a large number of field types that you can use in your models, matching (to a large degree) the column types available in most popular databases. For example, the above model uses two DateTimeFields and two TextFields. As you can imagine, these are mapped to the DATETIME and TEXT columns in SQL. These field definitions not only determine what type of column is defined in the database, but also the way in which Django's admin interface and forms allow users to enter data. In addition to TextField, you can have BooleanFields, EmailFields (for e-mail addresses), FileFields (for uploading files) and even GenericIPAddressField, among others.

Beyond choosing a field type that's appropriate for your data, you also can pass one or more options that modify how the field behaves. For example, DateField and DateTimeField allow you to pass an "auto_now" keyword argument. If passed and set to True, Django automatically will set the field to the current time when a new record is stored. This isn't necessarily behavior that you always will want, but it is needed frequently enough that Django provides it. That's true for the other fields, as well—they provide options that you might not always need, but that really can come in handy.

Migrations

So, now you have a model! How can you start to use it? Well, first you somehow need to translate your model into SQL that your database can use. This means, before continuing any further, you need to tell Django what database you're using. This is done in your project's configuration file; in my case, that would be atfproject/atfproject/settings.py. That file defines a number of variables that are used throughout Django. One of them is DATABASES, a dictionary that defines the databases used in your project. (Yes, it is possible to use more than one, although I'm not sure if that's normally such a good idea.)

By default, the definition of DATABASES is:

```
DATABASES = {
'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
}
}
```

In other words, Django comes, out of the box, defined to use SQLite. SQLite is a wonderful database for most purposes, but it is woefully underpowered for a real, production-ready database application that will be serving the general public. For such cases, you'll want something more powerful, such as my favorite database, PostgreSQL. Nevertheless, for the purposes of this little experiment here, you can use SQLite.

One of the many advantages of SQLite is that it uses one file for each database; if the file exists, SQLite reads the data from there. And if the file doesn't yet exist, it is created upon first use. Thus, by using SQLite, you're able to avoid any configuration.

However, you still somehow need to convert your Python code to SQL definitions that SQLite can use. This is done with "migrations".

Now, if you're coming from the world of Ruby on Rails, you are familiar with the idea of migrations—they describe the changes made to the database, such that you easily can move from an older version of the database to a newer one. I remember the days before migrations, and they were significantly less enjoyable—their invention really has made Web development easier.

Migrations are latecomers to the world of Django. There long have been external libraries, such as South, but migrations in Django itself are relatively new. Rails users might be surprised to find that in Django, developers don't create migrations directly. Rather, you tell Django to examine your model definitions, to compare those definitions with the current state of the database and then to generate an appropriate migration.

Given that I just created a model, I go back into the project's root directory, and I execute:

```
django-admin.py makemigrations
```

This command, which you execute in the project's root directory, tells Django to look at the "atfapp" application, to compare its models with the database and then to generate migrations.

Now, if you encounter an error at this point (and I often do!), you should double-check to make sure your application has been added to the project. It's not sufficient to have your app in the Django project's directory. You also must add it to INSTALLED_APPS, a tuple in the project's settings.py. For example, in my case, the definition looks like this:

```
INSTALLED_APPS = (
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'atfapp'
)
```

The output of `makemigrations` on my system looks like this:

```
Migrations for 'atfapp':
  0001_initial.py:
- Create model Appointment
```

In other words, Django now has described the difference between the current state of the database (in which "Appointment" doesn't exist) and the final state, in which there will be an "Appointment" table. If you're curious to see what this migration looks like, you can always look in the atfapp/migrations directory, in which you'll see Python code.

Didn't I say that the migration will describe the needed database updates in SQL? Yes, but the description originally is written in Python. This allows you, at least in theory, to migrate to a different database server, if and when you want to do so.

Now that you have the migrations, it's time to apply them. In the project's root directory, I now write:

```
django-admin.py migrate
```

And then see:

```
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, atfapp, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying atfapp.0001_initial... OK
  Applying sessions.0001_initial... OK
```

The above shows that the "atfapp" initial migration was run. But where did all of these other migrations come from? The answer is simple. Django's user model and other built-in models also are described using migrations and, thus, are applied along with mine, if that hasn't yet happened in my Django project.

You might have noticed that each migration is given a number. This allows Django to keep track of the history of the migrations and also to apply more than one, if necessary. You can create a migration, then create a new migration and then apply both of them together, if you want to keep the changes separate.

Or, perhaps more practically, you can work with other people on a project, each of whom is updating the database. Each of them can create their own migrations and commit them into the shared Git repository. If and when you retrieve the latest changes from Git, you'll get all of the migrations from your coworkers and then can apply them to your app.

Migrating Further

Let's say that you modify your model. How do you create and apply a new migration? The answer actually is fairly straightforward. Modify the model and ask Django to create an appropriate migration. Then you can run the newly created migration.

So, let's add a new field to the Appointment model, "minutes", to keep track of what happened during the meeting. I add a single line to the model, such that the file now looks like this:

```
from django.db import models


class Appointment(models.Model):
starts_at = models.DateTimeField()
ends_at = models.DateTimeField()
meeting_with = models.TextField()
notes = models.TextField()
minutes = models.TextField()    # New line here!
def __str__(self):
    return "{} - {}: Meeting with {} ({})".format(self.starts_at,
                      self.ends_at,
                      self.meeting_with,
                      self.notes)
```

Now I once again run

makemigrations, but this time, Django is comparing the current definition of the model with the current state of the database. It seems like a no-brainer for Django to deal with, and it should be, except for one thing: Django defines columns, by default, to forbid NULL values. If I add the "minutes" column, which doesn't allow NULL values, I'll be in trouble for existing rows. Django thus asks me whether I want to choose a default value to put in this field or if I'd prefer to stop the migration before it begins and to adjust my definitions.

One of the things I love about migrations is that they help you avoid stupid mistakes like this one. I'm going to choose the first option, indicating that "whatever" is the (oh-so-helpful) default value. Once I have done that, Django finishes with the migration's definition and writes it to disk. Now I can, once again, apply the pending migrations:

```
django-admin.py migrate
```

And I see:

```
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, atfapp, sessions
Running migrations:
  Applying atfapp.0002_appointment_minutes... OK
```

Sure enough, the new migration has been applied!

Of course, Django could have guessed as to my intentions. However, in this case and in most others, Django follows the Python rule of thumb in that it's better to be explicit than implicit and to avoid guessing.

Conclusion

Django's models allow you to create a variety of different fields in a database-independent way. Moreover, Django creates migrations between different versions of your database, making it easy to iterate database definitions as a project moves forward, even if there are multiple developers working on it.

In my next article, I plan to look at how you can use models that you have defined from within your Django application.■

Reuven M. Lerner is a Web developer, consultant and trainer. He recently completed his PhD in Learning Sciences from Northwestern University. You can read his blog, Twitter feed and newsletter at http://lerner.co.il. Reuven lives with his wife and three children in Modi'in, Israel.

How to succeed in a post-screen world: Find out at Solid.

The Internet of Things is changing everything. Not long ago, if you wanted to work with machines, you needed specialized knowledge of things like electrical engineering or assembly language. But with tools like node.js for embedded systems or Spark.io, programming physical objects has become as easy as programming a website.

The O'Reilly Solid Conference is coming to San Francisco's waterfront June 23–25. It's a unique event: a mash-up of MIT and Disneyland for the IoT—deep, intelligent conversations about the vital issues like security, business models, data, and standards; along with demos of some of the coolest devices, drones, robots, and wearables that exist—or are imagined—today.

Solidcon.com

Save 20% on your ticket
Use code LINUXJ

@oreillysolid

"The future has a funny way of sneaking up on you. You don't notice it until you're soaking in it. That was the feeling at O'Reilly's Solid Conference."

-Wired

O'REILLY®

Solid

CONFERENCE

HARDWARE, SOFTWARE & THE INTERNET OF THINGS

JUNE 23–25, 2015
SAN FRANCISO, CA

DAVE TAYLOR

System Status as SMS Text Messages

Loadwatch—intrepid shell script columnist Dave Taylor admits that it's not the tool for all jobs and shows how to create a smart load alert script that's perfect for sending SMS notifications to your new smart watch.

If you're paying really close attention, you'll remember that in my last article, I was exploring the rudiments of a script that would accept a list of words as input and create a word search grid, suitable for printing. It turns out that's crazy hard to do as a shell script—it just doesn't have the muscle to implement any sort of functional algorithm in an elegant fashion. So, I'm going to bail on it, at least until I can find someone else's open-source code I can explore for inspiration.

Or, of course, if you're motivated and have some time to experiment, go back to my April 2015 column, read through it, then try your own hand at implementing something.

Before I get letters about the oddity of being the shell script programming columnist who is bailing on a script, I will point out that there's a lot to learn from this experience actually. Most specifically, although it's nice to imagine that the Linux environment is completely egalitarian, and that every script, every language and every program is as powerful and well designed as every other, it's clear that's not the case.

Take Perl versus Awk, for example. Awk is powerful and I use it frequently, but although there are major software programs written in Perl, you'd be hard-pressed to find any significant software, functions, applications or utilities programmed directly in Awk. The same goes for C++ versus PHP, for example, or any modern structured language versus, well, the Bourne Again Shell. There, I said it. Shell script programming can take you only so far, and then you realize that you've hit the edges of the environment and its capabilities, and it's time to jump to another language.

Indeed, when I wrote my popular book *Wicked Cool Shell Scripts*, there was a tiny C program that snuck in by necessity: it was a few lines of C to do a certain date calculation that would have been dozens, if not hundreds, of lines of shell script.

Having said that, I will rush back to defend the shell as a powerful, lightweight programming and prototyping environment perfect for a variety of tasks because of its super-easy access to the power and capabilities of the entire Linux environment and, by extension, the entire Internet.

What's your take? You read this column, so it's reasonable for me to conclude that you are interested in learning more about programming within the Linux shell environment. How often do you bump into the bleeding edge of your shell and realize you have to flip into Perl, Ruby, C, Cobol (just kidding!) or another more sophisticated development environment to solve the problem properly?

Let's Talk about Text Messages

I was watching the Apple introduction of its new Apple Watch and was struck by the fact that

Table 1. E-mail Addresses of SMS Gateways for Cellular Carriers

WIRELESS CARRIER	DOMAIN NAME
At&T	@txt.att.net
Cricket	@mms.mycricket.com
Nextel	@messaging.nextel.com
Qwest	@qwestmp.com
Sprint	@messaging.sprintpcs.com
T-Mobile	@tmomail.net
US Cellular	@email.uscc.net
Verizon	@vtext.com
Virgin	@vmobl.com

like a few of the high-end Android smart watches, it will show you the entirety of e-mail and text messages on the tiny watch screen. This means it's a great device for sysadmins and Linux IT folk to keep tabs on the status of their machine or set of machines.

Sure, you could do this by having the system send an e-mail, but let's go a bit further and tap into one of the e-mail-to-SMS gateways instead. Table 1 shows a list of gateway addresses for the most common cellular carriers in the United States.

For example, I can send a text message to someone on the AT&T network with the number (303) 555-1234 by formatting the e-mail like this:

```
3035551234@txt.att.net
```

Armed with this information, there are a lot of different statuses that you can monitor and get a succinct text message if something's messed up.

Worried about load averages becoming excessive? That's a figure easily accessible through the one-line output of `uptime`:

```
$  uptime
11:20  up 4 days, 22:44, 3 users, load averages: 1.08 1.40 1.46
```

The last three figures are the load average over the last 1, 5 and 15 minutes. In this case, the system barely is being tapped at all. But what if it jumped up to 10, or 35 or more than 100? Then everything would slow down. Here's how you could write a simple script to test for that condition:

```
#!/bin/sh
# loadwatch.sh - send an alert if uptime > MAXOK
MAXOK=10
loadavg=$(uptime | cut -d\   -f11 | cut -d. -f1)
if [ $loadavg -gt $MAXOK ] ; then
  echo "Alert: Load avg $(uptime | cut -d\  -f11)"
fi
exit 0
```

Armed with the information about the various SMS gateways, it's easy to hard code a recipient address, which changes just the `echo` line within the conditional:

```
mail -s "Alert: Load avg $(uptime|cut -d\  -f11)" $recipient
```

where earlier in the script "recipient" is formatted similar to:

```
recipient=3035551234@txt.att.net
```

or as appropriate for your own smart watch or, um, other device.

For this script to be useful, you'd likely want to run it every few minutes so that when there is a spike in usage, you're alerted as soon as possible. This most easily would be a cron job, and if you haven't explored how your own custom cron jobs can make your life as even the most rudimentary of Linux users better, well, you're missing out!

To make the script run every ten minutes, here's how it might look in the root or even just your user crontab file:

```
0,10,20,30,40,50 * * * *  /home/taylor/bin/loadwatch.sh
```

Modern crontabs have a more sophisticated notational language that can make this a wee bit more succinct:

```
*/10 * * * *  /home/taylor/bin/loadwatch.sh
```

For this really to be useful, it might be better to have the script monitor state changes, so it'd notify you when the load rose above a specified threshold but not notify you again until it then went back down below that threshold.

This is done with what we old-school programmers call a semaphore, a state variable that remembers what's happening. Because a shell script is transient in nature, the semaphore needs to be a file. Typically these are located in a protected directory of some sort, but let's just drop it in your home directory for the purposes of this demo script.

The command-line function that's useful to know at this point is lockfile(1). This manages the atomic creation of the semaphore so that you never hit what's called a "race condition" where two instantiations of the script might collide on who is creating the file.

Here's how it'll work with the addition of the semaphore:

```
statefile=/home/taylor/bin/.loadavg
if [ -f "$statefile" ] ; then
  # statefile already exists, we're in a high load situation
  if [ $loadavg -gt $MAXOK ] ; then
    # still in high load situation
    echo "nothing to do, still in high load situation"
  else
    # high load situation has ended
    /bin/rm -f $statefile
    mail -s "Alert: load average back to normal" $recipient \
      < /dev/null > /dev/null 2>&1
  fi
else
  # statefile doesn't exist, let's create it.
  if [ $loadavg -gt $MAXOK ] ; then
    # load average has jumped above OK level
    lockfile $statefile
    load=$(uptime | cut -d\  -f11)
    mail -s "Alert: load average is $load" $recipient \
      < /dev/null > /dev/null 2>&1
  else
    # load average was okay and still is.
    echo "nothing to do, load average still ok."
  fi
fi
```

Of course, there are two of the four possible scenarios where you'd really want to remove the debugging code, clean up the if-then-else chain and shorten the script, because if this is going to run every ten minutes, you most assuredly do not want "no change" messages generated!

With that in mind, here's the more succinct code block:

```
if [ -f "$statefile" ] ; then
  # statefile already exists, we're in a high load situation
  if [ $loadavg -le $MAXOK ] ; then
    # high load situation has ended
    /bin/rm -f $statefile
    mail -s "Alert: load average back to normal?" $recipient \
      < /dev/null > /dev/null 2>&1
  fi
else
  # statefile doesn't exist, let's create it.
  if [ $loadavg -gt $MAXOK ] ; then
    # load average has jumped above OK level
    lockfile $statefile
    load=$(uptime | cut -d\  -f11)
    mail -s "Alert: load average is $load?" $recipient \
      < /dev/null > /dev/null 2>&1
  fi
fi
```

Note the extra work involved in using the command-line Mail program, where you have to redirect input so that it's not waiting for a message from stdin and redirecting the resultant "null message body" warning message. That's what this does:

```
< /dev/null > /dev/null 2>&1
```

Otherwise, hopefully you can read through and see what it does.

What Else Could You Monitor?

Tracking load average is rather trivial when you think about all the many things that can go wrong on a Linux system, including processes that get wedged and use an inordinate amount of CPU time, disk space that could be close to filling up, RAM that's tapped out and causing excessive swapping, or even unauthorized users logging in.

All of those situations can be analyzed, and alerts can be sent to you via e-mail or SMS text message, even to your shiny gold $17,000 Apple Watch. Now, you tell me, what do you think is worth monitoring on your system?■

Dave Taylor has been hacking shell scripts for more than 30 years—really. He's the author of the popular *Wicked Cool Shell Scripts* (10th anniversary update coming very soon from O'Reilly and NoStarch Press). He can be found on Twitter as @DaveTaylor and more generally at his tech site http://www.AskDaveTaylor.com.

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

LINUX JOURNAL
on your
Android device

Download the app now on the **Google Play Store**

LINUX JOURNAL
Since 1994: The Original Magazine of the Linux Community
A LOOK AT 20 YEARS OF WEB DEVELOPMENT
HOW-TOs
ND-LINE TRICKS AND TIPS
ES FOR HARD DRIVE CACHING
TO MANAGE YOUR CONFIGS
gcalcli
THE CLOUD ON THE COMMAND LINE
SciDAVis
FOR YOUR DATA ANALYSIS TASKS

www.linuxjournal.com/android

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

KYLE RANKIN

Libreboot on an X60, Part III: Modify the Boot Menu

Your BIOS is freer than ever, but now what? It's time to change that boot menu.

In the first two articles of this series, I explained the fundamentals behind the Libreboot free software BIOS project and why you might want to replace your BIOS with it. I followed up by describing how to install Libreboot on a ThinkPad X60. In this final article of the series, I explain how to perform one major task that so far I've left unexplained: how to modify the default GRUB boot menu.

A traditional BIOS provides users with a configuration menu where they can change boot orders and enable and disable devices. Typically there is an assigned key on the keyboard (Esc, F11 and F12 are common), so that you can select your boot device instead of going with the default order. With Libreboot, all of the device settings are set inside the ROM itself, and you use a GRUB menu to select a boot device.

The existing GRUB menu provides a number of common boot options that hopefully should work on your system. The default menu item attempts to boot off the first partition, and after that, there are options to boot removable devices and finally an option to search for and load any local GRUB configuration that might be on a hard drive. Ideally this default menu would be sufficient, but there are some cases (such as booting the Tails USB disk) that might require some tweaks.

On the one hand, if you are

familiar with GRUB commands, you can boot more less any device you want on the fly with the right incantation. On the other hand, it can be a pain to type in GRUB commands every time you want to boot something, so if you find yourself tweaking the default menu items to boot a special device, you probably will want to modify the GRUB menu more permanently.

What I suggest is that you experiment with sample GRUB configuration changes directly from the GRUB boot menu, because it allows you to edit the configuration of any menu item directly. This way, you quickly can test any sample changes without having to go through the full process of writing to and flashing a new ROM. Once you know what changes you'd like to make, you are ready to move on to make them permanent.

The Setup

If you have followed the previous two articles in this series, you already should have downloaded and validated the Libreboot binary and installed Libreboot on an X60. Let's pick up from that point by opening a terminal and changing to the libreboot_bin directory that contains all of the Libreboot binaries, ROMs and supporting scripts. You already should have installed the Libreboot build dependencies when setting up Libreboot, but if not, first run either the deps-trisquel or deps-parabola script if you are on a Debian-based or Arch-based distribution, respectively. If you are using another distribution, inspect the packages those scripts install and map them to the package names for your distribution.

The Libreboot ROM actually contains a small filesystem called CBFS, so to edit it, you need to install the cbfstool binary. Within the libreboot_bin directory is a script called builddeps-cbfstool, so run that script, and you should see a cbfstool and rmodtool binary appear under libreboot_bin:

```
$ ./builddeps-cbfstool
```

Modify the ROM

Once cbfstool is installed, the next step is to choose the ROM to modify so you can view the files within it and extract a copy of the GRUB configuration. For this example, I'm going to use one of the ROMs provided by Libreboot itself. First, run `cbfstool` along with the path to the ROM and the `print` argument. The `print` argument will then list all of the files within the ROM:

```
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb.rom print
libreboot_usqwerty_vesafb.rom: 2048 kB, bootblocksize 1424,
 ➥romsize 2097152, offset 0x0
alignment: 64 bytes, architecture: x86

Name                       Offset     Type         Size
cmos_layout.bin            0x0        cmos_layout  1788
cmos.default               0x740      cmos_default 256
fallback/romstage          0x880      stage        50924
fallback/ramstage          0xcfc0     stage        81695
fallback/payload           0x20f40    payload      541644
etc/ps2-keyboard-spinup    0xa5380    raw          8
config                     0xa53c0    raw          4504
background.jpg             0xa6580    raw          67907
dejavusansmono.pf2         0xb6f00    raw          100513
grub.cfg                   0xcf800    raw          1637
grubtest.cfg               0xcfec0    raw          1629
(empty)                    0xd0580    null         1242264
```

As you can see, there are two different GRUB config files: grub.cfg and grubtest.cfg. The former is the default GRUB config that is loaded, and the second can be loaded by the first for testing new configs. The fact is, if you make some major mistake in your GRUB config, you potentially could lock yourself out of booting your system (or at least make it very difficult), so it's important to validate your changes in a safe way. The recommended workflow is to modify grubtest.cfg first, update the ROM and flash your BIOS with it, then select the option in the GRUB menu to load grubtest.cfg. Then you can validate that your config works before you copy the same change to grub.cfg.

With that in mind, start by extracting the grubtest.cfg file using `cbfstool`:

```
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb.rom extract
 ➥-n grubtest.cfg -f grubtest.cfg
```

Here, instead of `print`, you are passing the `extract` command to `cbfstool` along with two new arguments. The `-n` option specifies the name of the file within the CBFS filesystem to extract, and the `-f` option specifies what to name the copy of the file on the local filesystem. Since the grub.cfg file references this specific filename, it's best to keep it the same.

The grubtest.cfg will contain a number of GRUB settings at the top of the file, but the more interesting settings will be found down in the `menuentry` sections:

```
menuentry 'Load Operating System' {
        set root='ahci0,msdos1'
        linux  /vmlinuz root=/dev/sda1
        initrd /initrd.img
}
```

```
menuentry 'Parse ISOLINUX menu (USB)' {
        set root='usb0'
        syslinux_configfile -i (usb0)/isolinux/isolinux.cfg
}
menuentry 'Parse ISOLINUX menu (CD)' {
        set root='ata0'
        syslinux_configfile -i (ata0)/isolinux/isolinux.cfg
}
```

For instance, the above three sections are for menu items to boot a Linux kernel from the first disk, a USB disk and a CD, respectively. If you find, for example, that your root partition isn't /dev/sda1 but instead /dev/sda2, you would edit the first `menuentry` section to reflect that. In my case, I noticed that the Tails live USB disks created prior to version 1.3 required a special set of boot options. After some experimentation, I came up with the following addition for GRUB:

```
menuentry 'Tails (USB)' {
        set root='usb0,gpt1'
        syslinux_configfile -i (usb0,gpt1)/syslinux/live486.cfg
}
```

A Quick Warning

Once you have made changes, it's time to copy the modified grubtest.cfg to your ROM. If you are using one of the standard Libreboot ROMs, I recommend first making a copy for your changes. This is important, because the default Libreboot ROMs are created with particular sections of the ROM blank to work well with initial flashing. I've personally bricked an X60 by attempting the first flash with one of my custom ROMs, so it's worth keeping the original ROMs intact:

```
$ cp bin/x60/libreboot_usqwerty_vesafb.rom
bin/x60/libreboot_usqwerty_vesafb-custom.rom
```

Now remove the old grubtest.cfg from your custom ROM and use the `print` command to confirm that it no longer exists:

```
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom remove
➥-n grubtest.cfg
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom print
libreboot_usqwerty_vesafb-custom.rom: 2048 kB,
➥bootblocksize 1424, romsize 2097152, offset 0x0
alignment: 64 bytes, architecture: x86

Name                          Offset     Type         Size
cmos_layout.bin               0x0        cmos_layout  1788
cmos.default                  0x740      cmos_default 256
fallback/romstage             0x880      stage        50924
fallback/ramstage             0xcfc0     stage        81695
fallback/payload              0x20f40    payload      541644
etc/ps2-keyboard-spinup       0xa5380    raw          8
config                        0xa53c0    raw          4504
background.jpg                0xa6580    raw          67907
dejavusansmono.pf2            0xb6f00    raw          100513
```

```
grub.cfg                    0xcf800     raw           1637
(empty)                     0xcfec0     deleted       1688
(empty)                     0xd0580     null          1242264
```

Now you are ready to add your custom version and use the `print` command to confirm it exists:

```
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom
 ➥add -n grubtest.cfg -f grubtest.cfg -t raw
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom print
libreboot_usqwerty_vesafb-custom.rom: 2048 kB, bootblocksize 1424,
 ➥romsize 2097152, offset 0x0
alignment: 64 bytes, architecture: x86

Name                         Offset     Type         Size
cmos_layout.bin              0x0        cmos_layout  1788
cmos.default                 0x740      cmos_default 256
fallback/romstage            0x880      stage        50924
fallback/ramstage            0xcfc0     stage        81695
fallback/payload             0x20f40    payload      541644
etc/ps2-keyboard-spinup      0xa5380    raw          8
config                       0xa53c0    raw          4504
background.jpg               0xa6580    raw          67907
dejavusansmono.pf2           0xb6f00    raw          100513
grub.cfg                     0xcf800    raw          1637
grubtest.cfg                 0xcfec0    raw          1629
(empty)                      0xd0580    null         1242264
```

Flash the BIOS

Now you can flash your BIOS with the modified ROM. You can use the flashrom utility that's included inside your Libreboot binary directory. If you are running this from the same system you used to install Libreboot in the first place, you already should have flashrom built and available. Otherwise, if you are running this from a system like Tails, or if you haven't yet installed flashrom, first run the `builddeps-flashrom` script from the base of the libreboot_bin directory as root. When you are ready to flash your BIOS, make sure you are in the libreboot_bin directory and run:

```
$ sudo ./flashrom/flashrom -p internal -w
bin/x60/libreboot_usqwerty_vesafb-custom.rom
flashrom v0.9.7-unknown on Linux 3.16.0-4-586 (i686)
flashrom is free software, get the source code at
 ➥http://www.flashrom.org

Calibrating delay loop... delay loop is unreliable, trying
 ➥to continue OK.
coreboot table found at 0xcf6bd000.
Found chipset "Intel ICH7M". Enabling flash write... OK.
Found Macronix flash chip "MX25L1605D/MX25L1608D/MX25L1673E"
 ➥(2048 kB, SPI) mapped at physical address 0xffe00000.
Reading old flash chip contents... done.
Erasing and writing flash chip...
Erase/write done.
```

Of course, replace the above ROM with the full path to your custom ROM. Once the flash succeeds, reboot your machine and at the boot menu, select the menu item that switches you to your custom grubtest.cfg. You then should see whatever changes you made, and you can attempt to boot from them. If everything works as expected, you are ready to make it the default. If not, especially if your changes made GRUB not work at all, just be glad it's the test file. You've been given a second chance to iterate through grubtest.cfg until it does work, and then you can move on.

Warning: make sure before you move on from here that you have completely tested your grubtest.cfg changes and everything works as expected.

Edit the Default Menu

Boot back in to your system and go back to your working directory. Since grubtest.cfg works, the next step is to create a copy of it named grub.cfg that you will use as the default GRUB config. The official Libreboot documentation for the GRUB menu lists this following sed script that will do all of the work of creating a grub.cfg based on your grubtest.cfg, but it will change the menu entries to make sure they still reference grubtest.cfg and grub.cfg where appropriate (be sure to run this in the directory that contains your custom grubtest.cfg):

```
$ sed -e 's:(cbfsdisk)/grub.cfg:(cbfsdisk)/grubtest.cfg:g'
➥-e 's:Switch to grub.cfg:Switch to grubtest.cfg:g'
➥< grubtest.cfg > grub.cfg
```

Now you can repeat the steps you performed to delete and re-add grubtest.cfg from the ROM, only this time with grub.cfg:

```
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom
➥remove -n grub.cfg
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom print
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom add
➥-n grub.cfg -f grub.cfg -t raw
$ ./cbfstool bin/x60/libreboot_usqwerty_vesafb-custom.rom print
```

Confirm that grub.cfg has been added properly to your ROM, and then flash your BIOS with the new custom ROM:

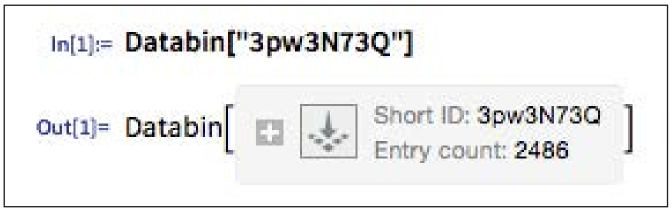
```
$ sudo ./flashrom/flashrom -p internal -w
bin/x60/libreboot_usqwerty_vesafb-custom.rom
```

Once you reboot, you should be able to use your new modified GRUB menu. Just be sure to take the extra steps of validating changes with grubtest.cfg first each time you do this—you wouldn't want to get locked out of your system!■

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.

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

Wolfram Research's Wolfram Data Drop



```
In[1]:= Databin["3pw3N73Q"]

Out[1]= Databin[ Short ID: 3pw3N73Q
                 Entry count: 2486 ]
```

It's always enlightening to learn what inventor, scientist and MacArthur Genius Grant recipient Dr Stephen Wolfram is sharing with the world. The latest dispatch from his company, Wolfram Research, is the Wolfram Data Drop, a solution for handling data from the emergent Internet of Things. Dr Wolfram notes the power of the Wolfram Language for interpreting, visualizing, analyzing, querying and otherwise doing interesting things with this data. The question however, notes Dr Wolfram, is this: how should the data from all those connected devices and everything else actually get to where good things can be done with it? The Wolfram Data Drop is Wolfram Research's next-step contribution toward making the world computable. The Wolfram Data Drop not only gathers and stores information from connected devices and the Internet of Things, but once data is in, it also becomes both universally interpretable and universally accessible to the Wolfram Language and any system that uses it. Dr Wolfram adds that the Wolfram Data Drop will be of great value to organizations or individuals that create connected devices, enabling them to store their data in the Wolfram Cloud, or a private version of it, where it will be readily accessible to analyze, visualize, query or deploy.
http://www.wolfram.com

Jolla Ltd.'s Sailfish Secure

Mobile developer Jolla shows how our tribe shines brightest when options are few. Chafing at the vanilla-chocolate, iOS-Android dominance, the team at Jolla has released the secure—and first "truly open"—mobile phone platform called Sailfish Secure. Based on Sailfish OS and utilizing SSH Communications Security's SSH communication encryption and key management platform, Sailfish Secure is a secure, adaptable mobile phone solution for government officials, corporations and consumers. With Sailfish Secure, Jolla and partner SSH seek to satiate the increased demand for privacy in mobile communications as well heed the call for a secure, transparent and open mobile solution alternative that is not controlled by any country or major industry player. Sailfish Secure also enables significant adaptation to local needs and hardware configurations. Jolla and SSH welcome other industry players to join the initiative.
http://jolla.com

Ryft Systems Inc.'s Ryft ONE

To illustrate the power of its new Ryft ONE analytics platform, Ryft Systems claims that a single device, using less power than a hair dryer, can store and analyze the equivalent of the contents of Wikipedia in 4.5 seconds. Ryft ONE, says its producer, is the first commercially available 1U analytics platform capable of an unprecedented 10+ GB/second performance without any data indexing, preprocessing, tuning or partitioning. The company further shared that the Ryft ONE, powered by a new massively parallel, hardware-accelerated architecture, analyzes historical and streaming data together at speeds 200X faster than conventional hardware, enabling data scientists and business analysts to slash operational costs by 70%. The new architecture, called Ryft Analytics Cortex, is a platform built for real-time analysis, optimizing compute, storage and I/O in tandem. The Ryft ONE, adds Ryft, "is open and compact like a Linux server but executes like a high-performance computer".
http://ryft.com

EarthLCD.com's Pi-Raq

EarthLCD's new Pi-Raq, an open-source Raspberry-Pi-based 1U rackmount Internet appliance, was inspired by the potential of an earlier company innovation, the "world's first" 10" x 1" (25.4cm x 2.5cm) TFT LCD. That display is integrated directly into the Pi-Raq. This was possible thanks to EarthLCD completing a comprehensive open-source reference design for the new product, which includes packaging, software and firmware to build a 1U rackmount appliance. EarthLCD notes the importance of Raspberry Pi for allowing it to get the TFT LCD into customers' hands quickly and allowing them to design high-value Internet appliances rapidly, merely by adding their software- and application-specific I/O via USB or I2C interfaces. Running Debian Linux opens up the Pi-Raq standard network analyzer software, allowing music servers and numerous open-source applications to be ported to the Pi-Raq.
http://earthlcd.com

Red Hat's Certified Container Ecosystem Program

Red Hat's newly announced Certified Container Ecosystem Program is really a three-"product" solution set for delivering secure, reliable and verified Docker-based containers to the enterprise world. The first element of the program for ISV partners consists of access to the Red Hat Container Development Kit, a collection of tools and resources to create enterprise-ready containers. The second element is the toolset for delivering the Red Hat Container Certification, which verifies that a container's contents are secure, unmodified, free of vulnerabilities and supported on Red Hat infrastructure. And finally, the third element is the distribution mechanism, the fully supported Red Hat Container Registry. This inaugural registry eventually will be part of a network of federated, standardized container registries hosted by partners and ISVs. The new product is based on Red Hat's outlook of Linux containers as the next wave of enterprise application architecture. Containers facilitate the creation of an efficient, composable fabric of lightweight "microservices" that can be woven into more complex applications, yet still are flexible enough to adapt to changing IT needs.
http://redhat.com

SUSE OpenStack Cloud

Formerly known as SUSE Cloud, the new SUSE OpenStack Cloud 5 is available—the latest edition of the company's OpenStack distribution for building Infrastructure-as-a-Service private clouds. SUSE OpenStack Cloud 5 is based on the newest OpenStack Juno release and provides increased networking flexibility and improved operational efficiency to simplify private cloud infrastructure management. Version 5 also provides "as-a-service" capabilities to enable development and big-data analytic teams to deliver business solutions that integrate with SUSE Enterprise Storage and SUSE Linux Enterprise Server 12 data-center solutions rapidly. Additional version 5 benefits include greater networking functionality and support for third-party OpenStack networking plugins, seamless incorporation of existing servers running outside the private cloud and centralized log collection and search.
http://suse.com

NVIDIA SHIELD

NVIDIA is establishing itself further outside the confines of the PC with the newest member of the SHIELD family of gaming devices. NVIDIA SHIELD is NVIDIA's first living-room entertainment device, and the company calls it "the world's first Android TV console to deliver music, apps, console-quality games and video". SHIELD is built on Android TV and powered by NVIDIA's powerful Tegra X1 processor. The artistically designed SHIELD console streams high-quality 4K video content and enables native and streaming gameplay at 1080p resolution/60fps. The console provides complete access to Android TV's rich app and games ecosystem, with more than 50 games optimized for the platform. Other games, such as *Crysis 3*, *Doom 3: BFG Edition* and *Borderlands: The Pre-Sequel!*, are under development. SHIELD also is the gateway to the NVIDIA GRID game-streaming service, which the company sells as "the Netflix for Games". SHIELD comes pre-packaged with NVIDIA's console-grade controller, and optional accessories include a remote control, additional controllers and a vertical stand.
http://shield.nvidia.com

Jeffrey Haemer's *Git Under the Hood,* LiveLessons Video (Addison-Wesley Professional)

Since Linus Torvalds invented Git in 2005, it rapidly has become the standard distributed version control system in existence. If you need or want to stop stumbling around in Git and truly understand what you're doing, a professionally developed training resource is available. Jeffrey Haemer's *Git Under the Hood* contains more than six hours of video instruction covering four in-depth lessons to help developers gain a deeper understanding of Git so that they can use it more effectively. Intermediate to advanced developers comfortable using a terminal window/command line will learn the varied pieces of Git's repository, basic Git commands and how to use them, design parallels between Git and Linux, how to experiment when Git does something unexpected and how to extend Git with new commands.
http://informit.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.

Home Automation with Raspberry Pi

How-to project: time the lights in your home and control them remotely over the Internet.

BHARATH BHUSHAN LOHRAY

The Raspberry Pi has been very popular among hobbyists and educators ever since its launch in 2011. The Raspberry Pi is a credit-card-sized single-board computer with a Broadcom BCM 2835 SoC, 256MB to 512MB of RAM, USB ports, GPIO pins, Ethernet, HDMI out, camera header and an SD card slot. The most attractive aspects of the Raspberry Pi are its low cost of $35 and large user community following.

The Pi has several pre-built images for various applications (http://www.raspberrypi.org/downloads), such as the Debian-based Raspbian, XBMC-based (now known as Kodi) RASPBMC, OpenELEC-based Plex Player, Ubuntu Core, RISC OS and more. The NOOBS (New Out Of the Box Setup) image presents a user-friendly menu to select and install any of the several distributions and subsequently boot into any of the installed OSes. The Raspbian image comes with the Wolfram language as part of the setup.

Since its initial launch in February 2011, the Raspberry Pi has been revised four times, each time receiving upgrades but maintaining the steady price of $35. The newest release of the Pi (the Raspberry Pi 2) boasts a 900MHz quad core cortex A7 and 1GB of RAM. Moreover, Microsoft announced Windows 10 for the Raspberry Pi 2 through its IoT developer program for no charge (https://dev.windows.com/en-us/featured/raspberrypi2support). This, in addition to its versatile features, has caused fans like me to upgrade to the Raspberry Pi 2. With a few new Raspberry Pi 2 boards in hand, I set out to find some useful ways to employ my older Pi boards.

In this article, I briefly describe the requirements of the project that I outlined, and I explain the various tools I decided to use to build it. I then cover the hardware I chose and the way to assemble the parts to realize the system. Next, I continue setting up the development environment on the Raspbian image, and I walk through the code and bring everything together to form the complete system. Finally, I conclude with possible improvements and hacks that would extend the usefulness of a Pi home automation system.

The Internet of Things

An ongoing trend in embedded devices is to have all embedded devices connected to the Internet. The Internet was developed as a fail-safe network that could survive the destruction of several nodes. The Internet of Things (IoT) leverages the same redundancy. With the move to migrate to IPv6, A multitude of IoT-connected devices in a home has the potential to act as a living entity that exhibits response to stimuli.

the IP address space would be large enough for several trillion devices to stay connected. A connected device also makes it very convenient to control it from anywhere, receive inputs from various sensors and respond to events. A multitude of IoT-connected devices in a home has the potential to act as a living entity that exhibits response to stimuli.

Raspberry Pi Home Automation

Inspired by the idea of having a home that has a life of its own, I settled on a home automation project to control the lights in my living room. The goal of my project was to be able to time the lights in my living room and control them remotely over the Internet using a Web browser. I also wanted to expose an API that could be used to control the device from other devices programatically.

The interesting part of this project is not the hardware, which is fairly simple and easy to construct, but the UI. The UI that I had in mind would support multiple users logged in to the same Pi server. The UI state had to keep up with the actual state of the system in real time indicating which lights actually were on when multiple users operated the system simultaneously. Apart from this, the lights may toggle on or off when triggered by the timer. A UI running on a device, such as a phone or a tablet, may be subject to random connection drops. The UI is expected to handle this and attempt to reconnect to the Pi server.

Hardware

Having outlined the requirements, I began to build the hardware. Table 1 shows the bill of materials that I used to build the hardware part of the system, and Figure 1 shows a block diagram of the hardware system.

Wiring this is time-consuming but easy. First, wire the SMPS to the wall outlet by cutting off an extension cord

Table 1. Bill of Materials

COMPONENT	QUANTITY	APPROXIMATE PRICE	PROCURED FROM	FUNCTION
Raspberry Pi	1	$35	Newark	The CPU
SD card	1	$25	amazon.com	To boot the RPi
Edimax WiFi	1	$10	amazon.com (http://www.amazon.com/Edimax-EW-7811Un-150Mbps-Raspberry-Supports/dp/B003MTTJOY)	To give the RPi wireless connectivity
Relay module	1	$10	amazon.com (http://www.amazon.com/JBtek-Channel-Relay-Arduino-Raspberry/dp/B00KTELP3I)	Used for switching
Ribbon cable	1	$7	amazon.com (http://www.amazon.com/Veewon-Flexible-Multicolored-Breadboard-Jumper/dp/B00N7XWXRK)	To connect the RPi header to the relay module
Power supply	1	$8	amazon.com (http://www.amazon.com/gp/product/B00HF3G7NO)	To power the RPi and the relay module
Extension cord	9	$54	Walmart (http://www.walmart.com/ip/Qvs-PC3PX-10-10ft-3-Outlet-3-Prong-Power-Cabl-Extension-Cord-Ac-Male-To-Female/41440394)	To power the SMPS and to provide a plug interface to the relays
Pencil box	1	$2	Walmart	To house the entire setup
USB cable	1	$5	amazon.com (http://www.amazon.com/AmazonBasics-USB-Cable-Micro-Meters/dp/B003ES5ZSW)	To power the RPi
14 gauge wire	1	6	Home Depot	To wire the relay terminals to the live wire from the wall outlet
Cable clamp	1	$2	Home Depot	As a strain relief

at the socket end. Strip the wires and screw them into the screw terminals of the SMPS. Next, wire the Raspberry Pi to the SMPS by cutting off the type A end of the USB cable and wiring it to the wire ends of the SMPS and the micro B end to the RPi. Strip out two strands of wires from the ribbon cable, and wire

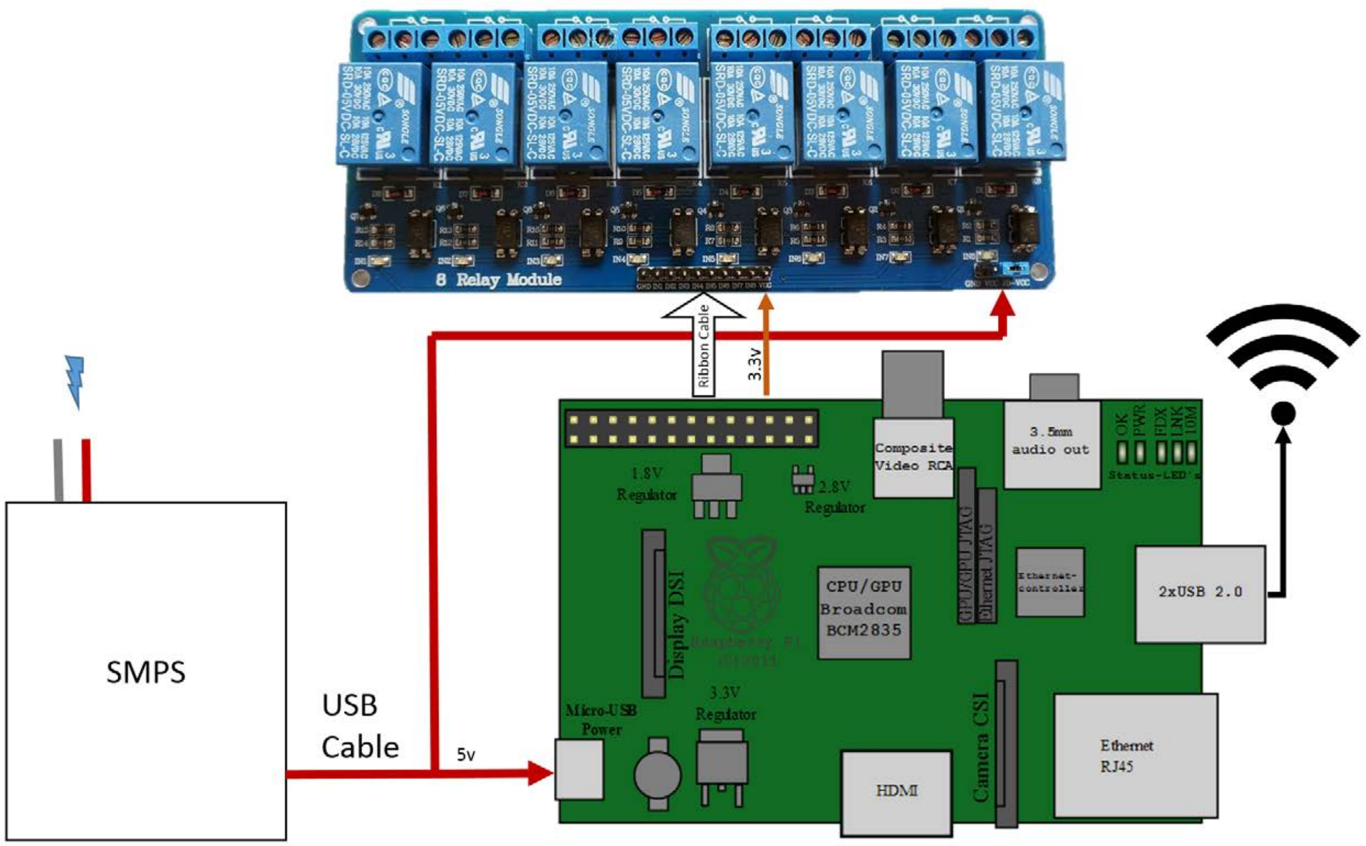
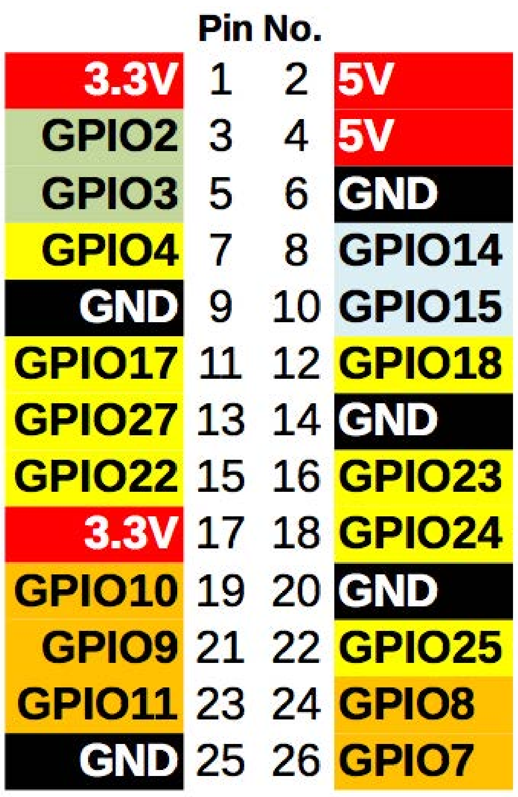


Figure 1. Block Diagram of the Hardware System

the appropriate terminals to GND and JDVcc. Remove the jumper that connects the JDVcc and Vcc. Not removing this jumper will feed back 5v to the 3.3v pins of the Pi and damage it.

Now that all the terminals are wired for power, connect the IN1-IN8 lines of the relay module to the appropriate GPIO pins of the RPi using more of the ribbon cable as shown in Figure 2. The code I present here is written for the case where I wire IN1-IN8 to GPIO1-GPIO7. Should you decide to wire them differently, you will need to modify your code accordingly.

The RPi's GPIO pins are shown in Figure 2. The RPi's IO ports operate at 3.3v, and the relay module works at 5v. However, the relays are isolated from the RPi's GPIO pins using optocouplers. The optocouplers may be supplied 3.3v over the Vcc pin. The Vcc pin of the relay module may be supplied 3.3v from the GPIO header of the Pi. *Make sure you have removed the jumper that bridges the Vcc and JDVcc on the relay module board.* The JDVcc pin should be supplied 5v for proper operation of the relay. The relay module is designed to be active low. This means that you



	Pin No.		
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7

Figure 2. The RPi's GPIO Pins

have to ground the terminals IN1-IN8 to switch on a relay.

Warning: handle all wiring with caution. Getting a shock from the line can be fatal!

Cut the remaining extension cables at the plug end, and screw in the wire end to the relay. Also daisy-chain the live wire from the wall outlet to the relay terminals. The entire setup can be housed in a pencil box or something similar. Plan this out in advance to avoid having to unwire and rewire the terminals. Additionally, I added a few screw cable clamps to the holes I made in my housing to act as a strain relief (Figure 3).

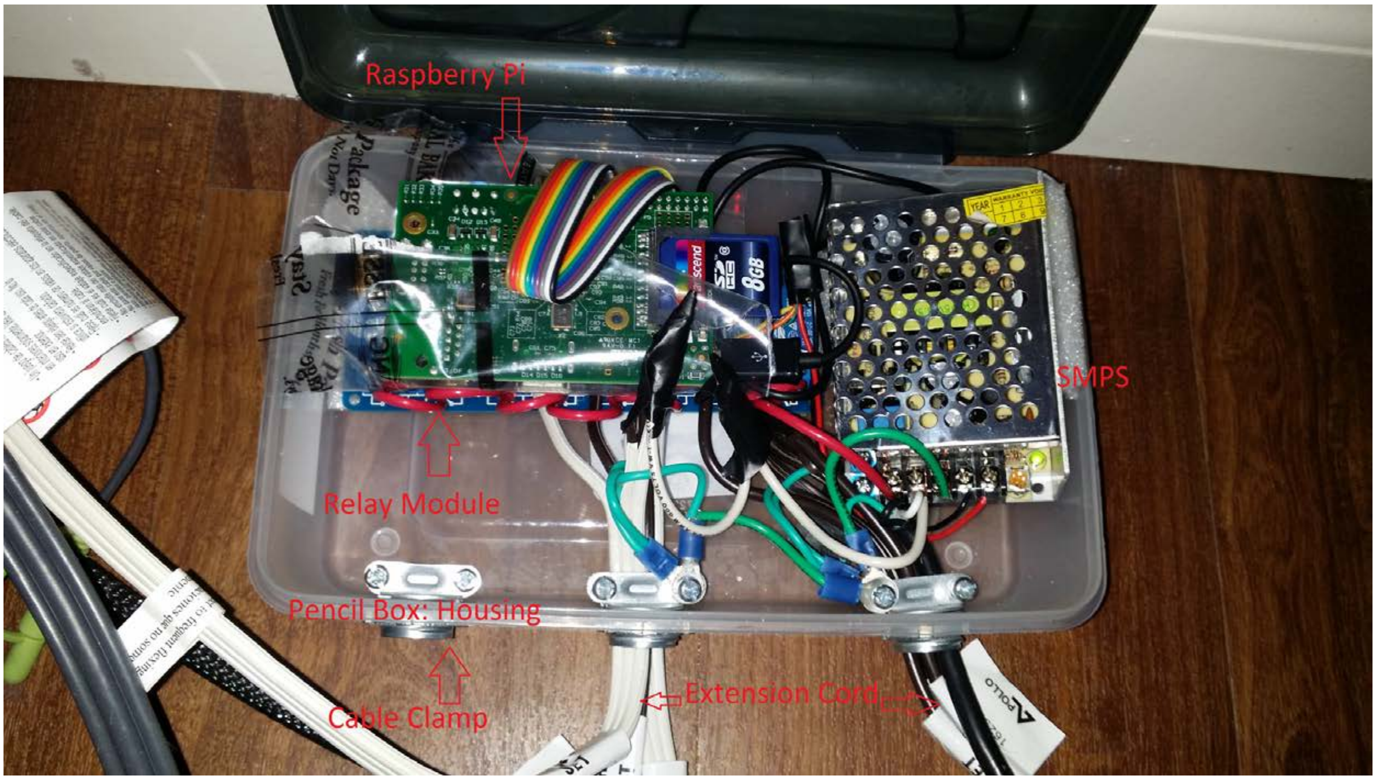


Figure 3. The Hardware Setup

Environment

I built my environment starting with a fresh install of Raspbian. For the initial installation, you need an HDMI-capable display, a USB keyboard, mouse and a wired Ethernet connection. You also optionally may connect a Wi-Fi adapter. Build the SD card for the first boot by following the instructions given at http://www.raspberrypi.org/documentation/installation/installing-image. During the first boot, the installer sets up the OS and expands the image to fill the entire card. After the first boot, you should be able to log in using the default credentials (user "pi" and password "raspberry").

Once you successfully log in, it's good practice to update the OS. The Raspbian image is based on Debian and uses the aptitude package manager. You also will need `python`, `pip` and `git`. I also recommend installing Webmin to ease administration processes. Instructions for installing Webmin are at http://www.webmin.com/deb.html (follow the directions in the "Using the Webmin APT repository" section):

```
sudo apt-get update && sudo apt-get dist-upgrade
sudo apt-get install python python-pip git git-core
```

Next, you need to set up the Wi-Fi connection. You can find detailed instructions for this at http://www.raspberrypi.org/documentation/configuration/wireless. I recommend the `wicd-curses` option. At this point, you can make changes to the RPi setup using the `sudo raspi-config` command. This will bring up a GUI that lets you choose options like the amount of RAM you share with the GPU, overclocking, GUI Boot and so on.

Another useful tool is the Cloud 9 IDE (https://github.com/c9/core). The Cloud9 IDE allows you to edit your code on the RPi using a Web browser. It also gives you a shell interface in the browser. You can develop and execute all your code without leaving the Web browser. The Cloud 9 IDE requires a specific version of NodeJS. Using the wrong version will cause frequent crashes of the Cloud 9 server, resulting in constant frustration. Instructions for installing NodeJS on the Raspberry Pi are outlined at http://weworkweplay.com/play/raspberry-pi-nodejs.

Software

I decided to build my front-end UI using HTML5, CSS3 and JavaScript. The combination of these three form a powerful tool for building

The back-end server on the Raspberry Pi needs to control the GPIO pins on the Raspberry Pi board. It also needs an HTTP interface to serve the UI and a WebSocket interface to pass command and status messages.

UIs. JavaScript provides easy communication APIs to servers. There also are a lot of JavaScript libraries like JQuery, Bootstrap and so on from which to choose. HTML5 supports the WebSocket API that allows the browser to keep a connection alive and receive communication over this connection. This makes WebSocket useful for implementing live and streaming apps, such as for games and chat interfaces. CSS is useful for styling the various HTML elements. When used properly, it lets one build dynamic UIs by switching the styles on an element in response to events. For this project, I chose JQuery to handle events, Bootstrap CSS (http://getbootstrap.com/css) to lay out the buttons in a grid form and pure JavaScript to handle WebSocket communications.

Libraries

The back-end server on the Raspberry Pi needs to control the GPIO pins on the Raspberry Pi board. It also needs an HTTP interface to serve the UI and a WebSocket interface to pass command and status messages. Such a specific server did not exist for off-the-shelf deployment, so I decided to write my own using Python. Python has prebuilt modules for the Raspberry Pi GPIO, HTTP server and WebSockets. Since these modules are specialized, minimum coding was required on my part.

However, these modules are not a part of Python and need to be installed separately. First, you need to be able to control the RPi's GPIO pins. The easiest way to do this from Python is by using the RPi.GPIO library from https://pypi.python.org/pypi/RPi.GPIO. Install this module with:

```
sudo pip install RPi.GPIO
```

Using the RPi.GPIO module is very simple. You can find examples of its usage at http://sourceforge.net/p/raspberry-gpio-python/wiki/Examples. The first step in using the module is to import it into the code. Next, you need to select the mode. The mode can be either GPIO.BOARD or GPIO.BCM. The mode decides whether the pin number references in the subsequent commands will be based on the BCM chip or the IO pins on the board. This is followed by setting pins as either input or output. Now you can use the IO pins as required. Finally, you need to clean up to release the GPIO pins. Listing 1 shows examples of using the RPi.GPIO module.

CherryPy is a Web framework module for Python (http://www.cherrypy.org). It is easily extendible to support WebSocket using the ws4py module (https://github.com/Lawouach/WebSocket-for-Python). CherryPy and ws4py also can be installed using `pip`:

```
pip install cherrypy
pip install ws4py
```

Examples of using the CherryPy framework and the ws4py plugin can be found in the CherryPy docs (https://cherrypy.readthedocs.org/en/latest) and the ws4py docs (http://ws4py.readthedocs.org/en/latest). A basic CherryPy server can be spawned using the code shown in Listing 2.

Slightly more advanced code would pass the `quickstart` method an object with configuration. The partial code in Listing 3 illustrates this. This code serves requests to /js from the js folder. The js folder resides in the

Listing 1. Using the RPi.GPIO Module

```
import RPi.GPIO as GPIO          # import module
GPIO.setmode(GPIO.BOARD)         # use board pin numbering
GPIO.setup(0, GPIO.IN)           # set ch0 as input
GPIO.setup(1, GPIO.OUT)          # set ch1 as output
var1=GPIO.input(0)                       # read ch0
GPIO.output(1, GPIO.HIGH)        # take ch1 to high state
GPIO.cleanup()                           # release GPIO.
```

home directory of the server code.

To add WebSocket support to the CherryPy server, modify the code as shown in Listing 4. The WebSocket handler class needs to implement three methods: `opened`, `closed` and `received_message`. Listing 4 is a basic WebSocket server that has been kept small for the purpose of explaining the major functional parts of the code; hence, it does not actually do anything.

On the client side, the HTML needs to implement a function to connect to a WebSocket and handle incoming messages. Listing 5 shows simple HTML that would do that. This code uses the `jQuery.ready()`

Listing 2. Spawning a Basic CherryPy Server

```
# From the CherryPy Docs at
# https://cherrypy.readthedocs.org/en/latest/tutorials.html

import cherrypy    # import the cherrypy module

class HelloWorld(object):      #
    @cherrypy.expose           # Make the function available
    def index(self):           # Create a function for each request
        return "Hello world!"  # Returned value is sent to the browser

if __name__ == '__main__':
   cherrypy.quickstart(HelloWorld())    # start the CherryPy server
                                        # and pass the class handle
                                        # to handle request
```

Listing 3. Passing the `quickstart` Method

```
cherrypy.quickstart(HelloWorld(), '', config={
   '/js': {          # Configure how to serve requests for /js
   'tools.staticdir.on': True,     # Serve content statically
                                   # from a directory
   'tools.staticdir.dir': 'js'     # Directory with respect to
                                   # server home.
   }
});
```

event to start connecting to the WebSocket server. The code in this Listing implements methods to handle all events: `onopen()`, `onclose()`, `onerror()` and `onmessage()`. To extend this example, add code to the `onmessage()` method to handle messages.

Listing 4. Basic WebSocket Server

```
import cherrypy                  # Import CherryPy server module
# Import plugin modules for CherryPy
from ws4py.server.cherrypyserver  import WebSocketPlugin, WebSocketTool
from ws4py.websocket import WebSocket    # Import modules for
                                         # the ws4py plugin.
from ws4py.messaging import TextMessage

class ChatWebSocketHandler(WebSocket):
        def received_message(self, m):
                msg=m.data.decode("utf-8")
                print msg
                cherrypy.engine.publish('websocket-broadcast',
                 ➥"Broadcast Message: Received a message")

        def closed(self, code, reason="A client left the room
         ➥without a proper explanation."):
                cherrypy.engine.publish('websocket-broadcast',
                 ➥TextMessage(reason))

class Root(object):
    @cherrypy.expose
    def index(self):
        return "index"

    @cherrypy.expose
    def ws(self):
        print "Handler created: %s" % repr(cherrypy.request.ws_handler)


if __name__ == '__main__':
    WebSocketPlugin(cherrypy.engine).subscribe()    # initialize websocket
                                                    # plugin
    cherrypy.tools.websocket = WebSocketTool()           #
    cherrypy.config.update({'server.socket_host': '0.0.0.0',
        'server.socket_port': 9003,
        'tools.staticdir.root': '/home/pi'})
    cherrypy.quickstart(Root(), '', config={
             '/ws': {
                     'tools.websocket.on': True,
                     'tools.websocket.handler_cls': ChatWebSocketHandler
               }
        });
```

Listing 5. Connecting to WebSocket and Handling Incoming Messages

```
<html>
    <head></head>
    <body>

    <script src="/js/jquery.min.js"></script>
    <script type="text/javascript">
    var ws;
    var addr="ws://127.0.0.1:9000";
    $(document).ready(function (){
            connectWS();
    });
    function dbg(m){
            console.log(m);
    }
    function connectWS(){
            dbg('Connecting...');
            if (window.WebSocket) {
                    ws = new WebSocket(addr);
            }
            else if (window.MozWebSocket) {
                    ws = MozWebSocket(addr);
            }
            else {
                    alert('Your archaic browser does not support
                     ➥WebSockets.');
                    dbg('WebSocket Not Supported');
                    return;
            }

            /* on websocket close */
            ws.onclose=function(){
                    dbg('Connection Closed.');
                    reconnect=setTimeout(connectWS,6000); //try to
                                                          //reconnect
                                                          //every 6 secs.
            }

            /* on websocket connection */
            ws.onopen=function(){
                    dbg('Connected.');
                    ws.send('Some message to send to the
                     ➥WebSocket server.');
            }

            /* on websocket error */
            ws.onerror=function(e){
                    dbg("Socket error: " + e.data);
            }

            /* on websocket receiving a message */
            ws.onmessage = function (evt) {
                    dbg(evt.data);
                    //add functions to handle messages.
            }
            return 0;
    }
    </script>
    </body>
</html>
```

Pi Home Automation

Now that you've seen the basics of WebSockets, CherryPy and the HTML front end, let's get to the actual code. You can get the code from the Git repository at https://bitbucket.org/lordloh/pi-home-automation. You can clone this repository locally on your RPi, and execute it out of the box using the command:

```
git clone https://bitbucket.org/lordloh/pi-home-automation.git
git fetch && git checkout LinuxJournal2015May
cd pi-home-automation
python relay.py
```

The relayLabel.json file holds the required configuration, such as labels for relays, times for lights to go on and off and so on. Listing 6 shows the basic schema of the configuration. Repeat this pattern for each relay. The `dow` property is formed by using one bit for each day of the week starting from Monday for the LSB to Sunday for the MSB.

Figure 4 shows the block diagram of the system displaying the major functional parts. Table 2 enumerates all the commands the client may send to the server and the action that the server is expected to take. These commands are sent from the browser to the server in JSON format. The command schema is as follows:

```
{
    "c":"<command form TABLE 2>",
    "r":<relay Number>
}
```

The `update` and `updateLabels` commands do not take a relay number. Apart from relay.py and relayLabel.json, the only other file required is index.html. The relay.py script reads this file and serves it in response to HTTP requests. The index.html file contains the HTML, CSS and JavaScript to render the UI.

Once the system is up and running, you'll want to access it from over the Internet. To do this, you need to set a permanent MAC address and reserved IP address for the Raspberry Pi on your

Listing 6. Basic Schema of the Configuration

```
{
  "relay1": {
    "times": [
      {
        "start": [
          <hour>,
          <minute>,
          <second>
        ],
        "end": [
          <hour>,
          <minute>,
          <second>
        ],
        "dow":
<Monday<<0|Tuesday<<1|Wednesday<<2|Thursday<<3|
➥Friday<<4|Saturday<<5|Sunday<<6>
      }
    ],
    "id": 1,
    "label": "<Appliance Name>"
  }
}
```

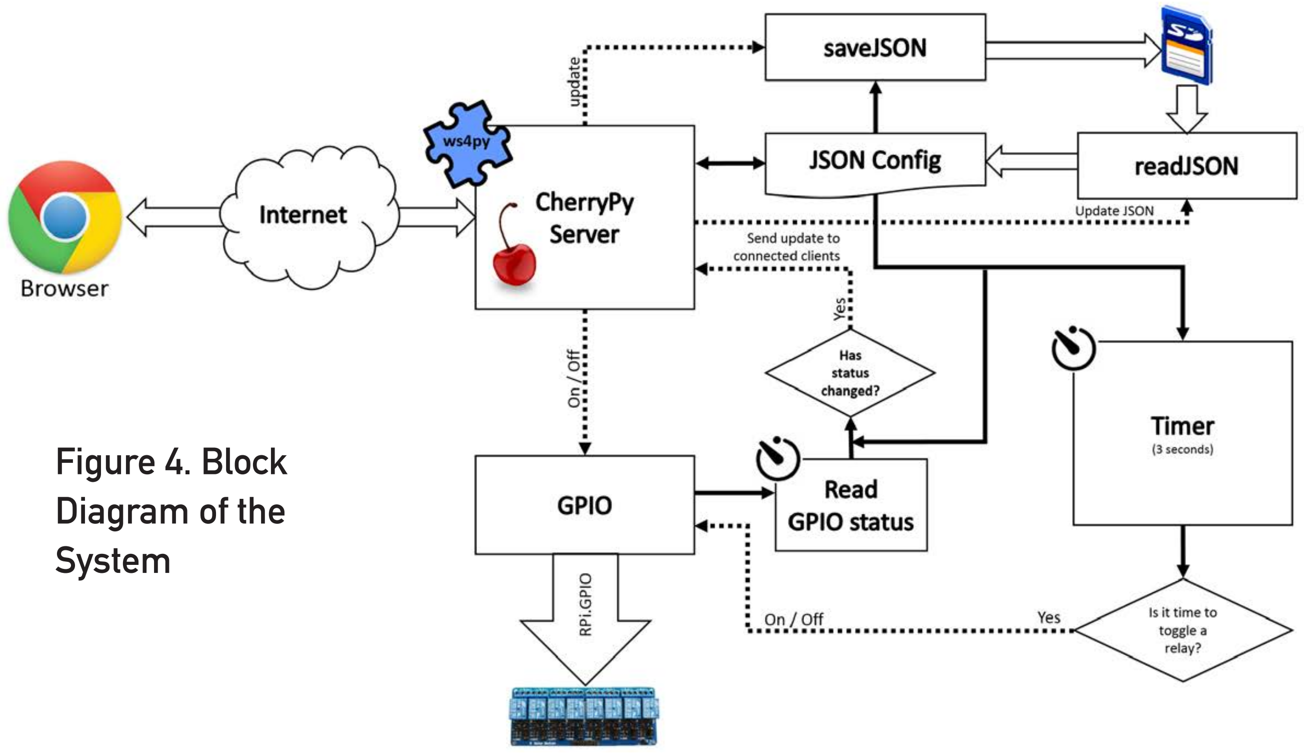


Figure 4. Block Diagram of the System

local network, and set up port forwarding on your router. The process for doing this varies according to router, and your router manual is the best reference for it. Additionally, you can use a dynamic domain name service so that you do not need to type your IP address to access your Pi every time. Some routers include support for certain dynamic DNS services.

Conclusion

I hope this article helps you to build this or other similar projects. This project can be extended to add new features, such as detecting your phone connected to your Wi-Fi and switching on lights. You also could integrate this with applications, such as OnX and Android Tasker. Adding password protection for out-of-network access is beneficial. Feel free to mention any issues, bugs and feature requests at http://code.lohray.com/pi-home-automation/issues. ■

Table 2. Commands

COMMAND	DESCRIPTION
`on`	Switch a relay on
`off`	Switch a relay off
`update`	Send status of GPIO pins and relay labels
`updateLabels`	Save new labels to JSON files

Bharath Bhushan Lohray is a PhD student working on his dissertation on image compression techniques at the Department of Electrical & Computer Engineering, Texas Tech University. He is interested in machine learning.

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

EMBED LINUX IN MONITORING AND CONTROL SYSTEMS

How to use a standard Linux distribution platform to make a user interface embedded into a vehicle monitoring and control system.

RICK BROWN

The target vehicle for this project is a vintage intercity transport bus (think Greyhound) whose instrument panel was sparse and mostly nonfunctional. The speedometer cable was twisted off some place back in 40 feet of cable, and the fuel sensor had sunk long ago. What I wanted was an instrument panel more in line with modern practice.

To bridge the gap, I used a laptop computer running the Fedora 20-KDE distribution of Linux as a host, three digital signal processor boards as hardware interface processors (HIPs), a USB/RS422 converter that connects to an RS422 loop linking the HIPs together and some software that I call the Vehicle Monitoring and Control application.

System Architecture

The HIPs are based on a signal processor chip, programmed in C and with no user interface except a heartbeat LED to show that the processor is working to some degree. The HIPs provide signal conditioning circuitry for analog input scaling and optical isolation for control signals plus a few specials like thermocouple converters and a pressure transducer chip. There also are two RS422 receiver/transmitter pairs. One pair connects up-network (toward the host) and the other down-network (toward the other HIPs).

The way this application works is that a message is originated by the host processor and transmitted down-loop to the first HIP. There it may be modified under HIP program control and relayed on down-loop to the next HIP. The last HIP in the "loop" transmits its message up-loop under physical jumper control. Processors closer to the Host simply pass on up what is coming from below in the "loop". The Host is the ultimate receiver of the messages it originates.

A message consists of an SOM byte, an address byte with acknowledge bit, a command byte, four data bytes and two CRC bytes. Going down loop, the HIPs relay a message on a character-by-character basis with a one-character delay per HIP. The addressee of a message sets the acknowledge bit and inserts or extracts data on the fly. So in a short loop like the one here, the host begins receiving the response from the network before it has finished sending the original message. For this loop, the communication rate was selected, arbitrarily, as 57,600 baud, so the loop message time is (9 + 3)/57600 or 208 microseconds. The left portion of Figure 1 depicts the loop topology.

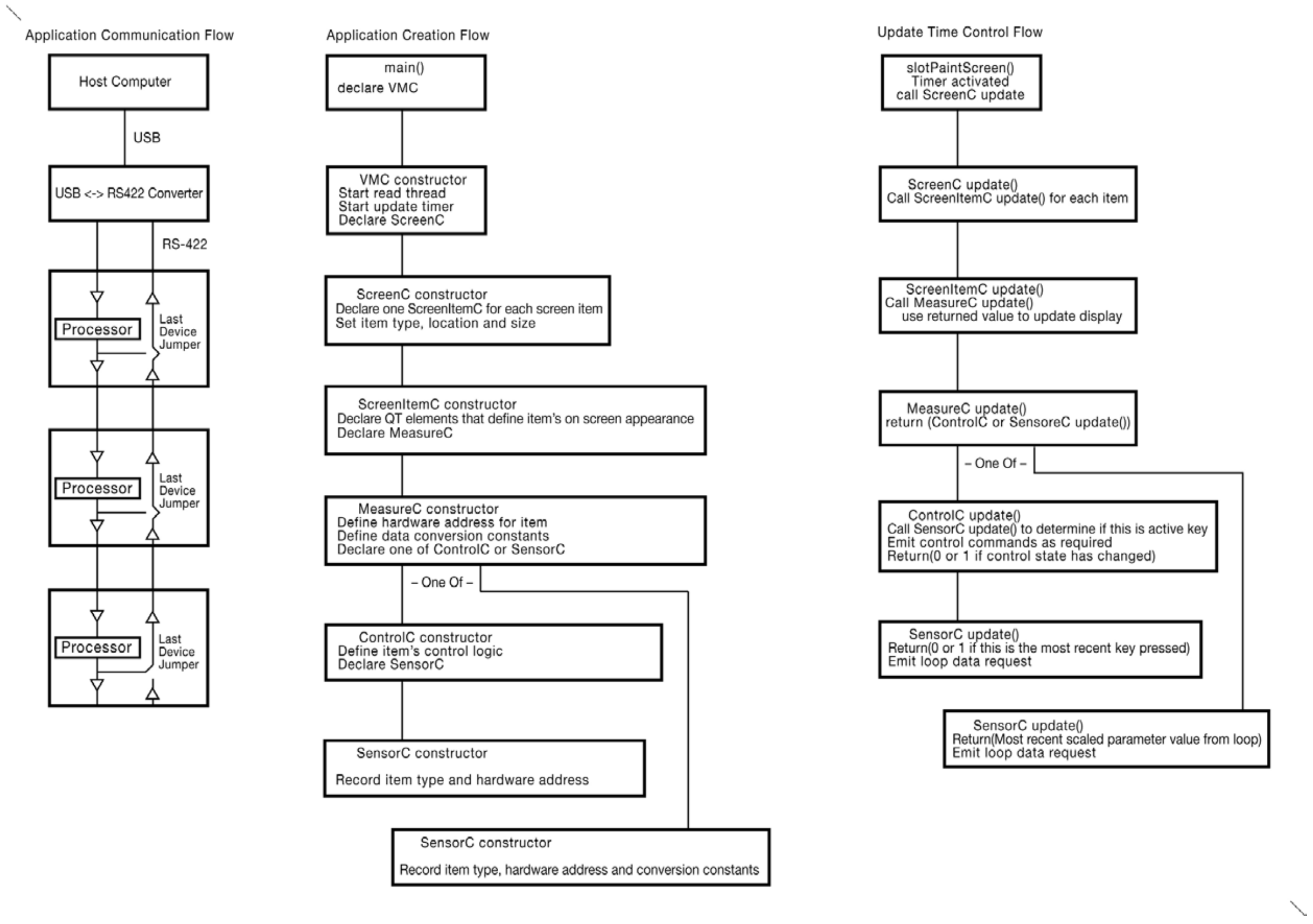

Application Communication Flow
Host Computer
USB
USB <-> RS422 Converter
RS-422
Processor
Last Device Jumper
Processor
Last Device Jumper
Processor
Last Device Jumper
Application Creation Flow
main()
declare VMC
VMC constructor
Start read thread
Start update timer
Declare ScreenC
ScreenC constructor
Declare one ScreenItemC for each screen item
Set item type, location and size
ScreenItemC constructor
Declare QT elements that define item's on screen appearance
Declare MeasureC
MeasureC constructor
Define hardware address for item
Define data conversion constants
Declare one of ControlC or SensorC
– One Of –
ControlC constructor
Define item's control logic
Declare SensorC
SensorC constructor
Record item type and hardware address
SensorC constructor
Record item type, hardware address and conversion constants
Update Time Control Flow
slotPaintScreen()
Timer activated
call ScreenC update
ScreenC update()
Call ScreenItemC update() for each item
ScreenItemC update()
Call MeasureC update()
use returned value to update display
MeasureC update()
return (ControlC or SensoreC update())
– One Of –
ControlC update()
Call SensorC update() to determine if this is active key
Emit control commands as required
Return(0 or 1 if control state has changed)
SensorC update()
Return(0 or 1 if this is the most recent key pressed)
Emit loop data request
SensorC update()
Return(Most recent scaled parameter value from loop)
Emit loop data request

Figure 1. System Architecture

The Vehicle Monitoring and Control (VMC) application will originate messages like "HIP1 set or get register whatever". First, I show how to set up a development environment on a Linux box, and then I talk about how to use the tools made available there to weave together a Linux real-time application that performs the VMC application.

Set Up the Development Environment

My choice for a development environment is KDevelop from KDE, and for a toolkit, it is Qt from the Qt Project. The first step is to get the development environment and then build a "Hello World" application. So, here it is in the Fedora world:

```
yum install kdevelop
yum install qt
yum install gcc
```

...and lots of other stuff. Expect to spend some time if you are not already up and running with Qt.

Qt is an excellent toolkit that is robust and well documented—except for a few quirks.

When you get KDevelop to load, click Session→Start New Session→New Project. This will be "Qt" and "Graphical". Make up a name (VMC for example), accept the defaults, and soon you will be presented the opportunity to "Build" and then "Execute". On "Execute", a Launch Configurations dialog box will enable you to select your project name, "Add New", and your project. A click or two later, and you should see a basic "Hello World" window appear on your screen. This you may expand into your real-time application.

The "Hello World" you see is a Qt application. Qt is an excellent toolkit that is robust and well documented—except for a few quirks. The new project build process creates a directory structure that includes ~/projects/VMC/build. To minimize hard-to-diagnose Qt build errors, keep all of your source and header files in ~/projects/VMC until you know how to do otherwise. The ~/projects/VMC/build directory is the execute directory for the purposes of KDevelop. It is here that run directory files should reside. As you add source files and libraries, you must keep ~/projects/VMC/CmakeLists.txt current.

Build the Application

Here is how to use tools available in the Linux environment to create the VMC application. First up is communications. To your application, the communication loop looks like a file stream created like this:

```
int hNet = open("/dev/ttyUSB0", O_RDWR);
```

or /dev/ttyUSBwhatever, depending upon what else is going on in your system.

Now you can `read()` and `write()` hNet and the USB↔RS422 converter will connect you to the loop. Writing is no issue up to loop speed (in this case 57600/9 = 6400 messages/second), so that is your write (hNet,...) speed limit. Reading is a different deal as read(hNet,...) is a blocking operation. A process that makes that call remains stuck there until some data arrives. Thus, you want to make your read(hNet,...) calls from a process (thread) whose only task is to catch characters as they come in and then make them available in a buffer to other processes as they need them—most briefly, in abbreviated code:

```
//A thread to perform the read(hNet,...) function
class COMthread : public Qthread
{
Q_OBJECT    //Notice use of the Qt tools
protected:
     //Start point when myThread->start(); is called
void run()
{
while (1)
{
pthread_mutex_t mutex1\
         = PTHREAD_MUTEX_INITIALIZER;
//Lock out other processes while working
pthread_mutex_lock( &mutex1 );
          -manipulate shared variables here-
//unlock for other processes during read(hNet,...
pthread_mutex_unlock( &mutex1 );

//This is where this thread spends
read(hNet, Buf, BUF_SIZE);/////////
//99.99 (+/-) percent of its time

//Now lock while updating for new data
pthread_mutex_lock( &mutex1 );
         -buffer data and update pointers-
pthread_mutex_unlock( &mutex1 );
}
}
};
```

To activate that code, your statements in the VMC constructor are:

```
COMthread    *gCOMgo = new COMthread;
gCOMgo->start();
```

The complement to that loop data fetch is a character fetch routine running under some other process. That routine, using its own mutexes, extracts data from the buffer sourced by the thread above.

Now that you can send and receive data via the loop, let's look at how the application may interact with the hardware.

Figure 2 shows the Instrument Panel display as seen on a video display mounted in the driver's view.

The Tach and Speed display data are sourced from timer registers in an HIP that is timing the period between shaft rotations. The five indicators below are sourced by A/D registers in various HIPs. These seven data items are collected by sending seven nine-character data request messages to the loop and decoding the returned 63 characters (7X9). Below that is a representation of a partially populated map of a 4X4 keypad that is serviced by one of the HIPs. Each of the represented keys on that map issues a query for the HIP responsible for the physical keypad to see if its key was the last pressed. It gets back yes or no.

When you use KDevelop to create a

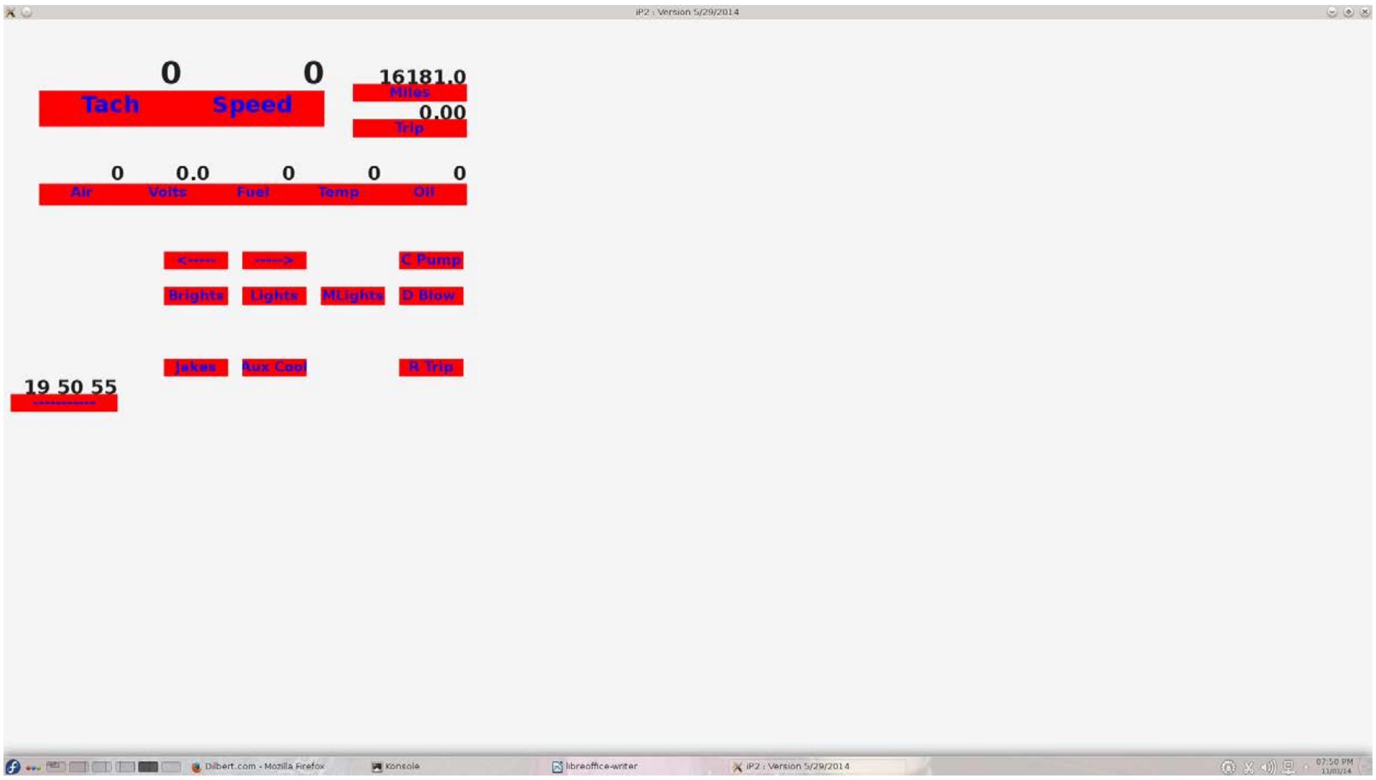


Figure 2. Instrument Panel Display

VMC project, some files of interest to you now were created. Look in directory ~/projects/VMC, and there you will find main.cpp and VMC.cpp. File main.cpp is fine as is. It simply declares and runs the application described by the code in VMC.cpp. None of the sample code in VMC.cpp within the curly braces is useful for you, so let's replace it with the constructor for the VMC application. As I mentioned previously, this application relies upon Qt, so an important resource for you is http://qt-project.org/doc/qt-4.8/classes.html.

Your VMC class will inherit from QmainWindow, so your constructor will be defined in VMC.cpp as shown here:

```
VMC::VMC() : QMainWindow()
 {
 //declare a central widget to host our screen:
 QWidget* gCentralWidget = new QWidget(this);
 setCentralWidget(gCentralWidget);
//Set fonts, colors, geometry, etc
       - - - -
//Declare an object to hold the screen features:
ScreenC cScreenLayout = new ScreenC();
//Lastly, breathe life into the application
cHeartBeat = new QTimer (this);
connect(cHeartBeat, SIGNAL(timeout()), this,\
                          SLOT(slotPaintScreen()));
cHeartBeat->setSingleShot(false);
cHeartBeat->start(50); //milliseconds/20Hz
}
```

It is here in the ScreenItemC class code where you can fancy it up.

That is an abridged view of the constructor, but the actual code isn't much longer. The connected routine `slotPaintScreen()` will be activated on a 50 millisecond interval by the timer overflow. It too is brief:

```
//Fetch loop characters gathered by COMthread
SensorLoopService();
//Update the display
cScreenLayout->Update();
//Redraw the screen
update();
```

(Again abridged because this is a story about how to do it rather than how to code it.)

The central portion of Figure 1 shows the cascade of object creation that will embody the VMC application. Notice the declaration of a ScreenC object by the VMC constructor and the update of that object at a 20Hz rate.

The ScreenC class constructor simply declares a ScreenItemC object for each entity that appears on the screen. A typical declaration is:

```
pF[i] = new ScreenItemC(xOff,yOff,xSiz,ySiz,\
                         MEAS_TACHOMETER, 0, "Tach");
```

Here you define the location and size and name the object type of each on-screen object. At update time "update" is simply relayed to its children like this:

```
//Update screen features
for (i=0; i<cNumberFeatures; i++)
{
 pF[i]->Update();
}
```

The ScreenItemC class constructor is responsible for the look of items on the screen. In this application, a ScreenItemC item consists of two QLabel objects placed one above the other so as to appear to be a single instrument. The form of a QLabel declaration is:

```
QLabel cReading = new QLabel(gCentralWidget, Qt::FramelessWindowHint);
```

The instrument displays of Figure 2 are pretty "plain Jane". It is here in the ScreenItemC class code where you can fancy it up. The ScreenItemC constructor also declares a MeasureC object. That object's update routine returns the data that the ScreenItemC object places on the screen:

```
MeasureC cMeasure = new
MeasureC(MEAS_TACHOMETER);
```

The MeasureC class is where the hardware interface is described. HIP address, register numbers and scale factors are defined. For example:

```
case MEAS_TACHOMETER:
{
fScale = 27648000.0;  //29.75Hz -> 1788rpm
      // RPM = fScale / binary from loop + fOffset
fOffset = 0;
rule = MEAS_RULE_RECIPROCAL_TACHO;
DeviceId = NODE_E;                  //Loop device id
DevicePort = P_IC_PERIOD_2;  //Sensor on device
//Create a sensor for the measurement
pSens = new SensorC(MEAS_TACHOMETER,\
    DeviceId, DevicePort, fScale, Offset, rule);
break;
}
```

Notice the declaration above of a SensorC object. At update time, that SensorC object will fetch its most recent raw reading from the loop buffer, scale that and return the result to its MeasureC parent, which will relay that back to its ScreenItemC parent, which will display that result on the screen. The MeasureC items that represent a keypad key will declare a ContrtolC object here. The ControlC object will use its own SensorC object to inquire of the loop if its key is the most recently pressed. ControlC objects also run device-specific code (like timing a blinker, for example). The ControlC object may place commands on the loop as necessary. The ControlC update routine will return 1 or 0 depending on whether its control target has changed state or not. That return flows back up the cascade to its grandparent ScreenItemC object and then is reflected on the display.

This cascade of object creation ends with SensorC objects that return the result of their previous request to the loop and issue a new data request at each update time. As ControlC objects may place commands on the loop at their whim, the loop will have a mixture of independent commands circulating that must be resolved back to their originator. When a command is issued to the loop, the issuer of that command also inserts into a class visible circular buffer a pointer to itself.

As mentioned above, slotPaintScreen() will call SensorLoopService() at each update time. SensorLoopService() extracts characters that have been placed into the loop receive buffer by the gCOMgo thread. Mutexes are used here to prevent interference by other

Some kinks that Linux throws in include the screen saver that defaults active, but is bad news in a monitoring application.

threads. SensorLoopService() parses the characters as it fetches data from the buffer, and when it has detected a complete valid message, it places the four data bytes into a location pointed to by the pointer mentioned above. This data will be returned up the cascade at the next update time.

Here it is in fewer words: the update event cascades down from the ScreenC object to multiple SensorC objects that bounce parameter states back up to ScreenItemC objects that paint those states on the screen. The left panel of Figure 1 depicts this.

Linux Environment Considerations

Some kinks that Linux throws in include the screen saver that defaults active, but is bad news in a monitoring application. To turn it off, go to System Settings→Power Management and disable all Screen Energy Saving options. Another issue is automatic software updates. It is my consideration that if something works, don't screw around with the operating environment, as software updates do. The safest way to suppress updating is by staying off the Internet while your application is active. Another way is to disable updates by software control. To do so, go to the Application Launcher (lower left on the desktop), start the System Settings from Favorites, go to Software Management and left-click the wrench icon at the upper-right edge. Select Settings from its menu. In the General Settings page, set the Check for updates menu to Never, and "Apply" that. Also, go to /etc/yum/pluginconf.d/refresh-packagekit.conf and set enabled to 0 (disable update). For me it was just too easy to switch off the Wi-Fi when I wanted a stable environment, so I can't give you other advice here.

To claim credit as being an "embedded" application, this system should come up with the power—that is, without login or any other user input required to make it go. To kill the login, go to /etc/kde/kdm/kdmrc and set `AutoLoginAgain=true` and `AutoLoginUser=YourUserName`. To bring up your application with system start up, go to ~/.kde/Autostart and place an executable script there like this:

```
#!/bin/bash
cd /home/YourUserName/projects/VMC/build
./VMC
```

Serendipitously, this will not bring up multiple instances of the application if it was active when you last powered down, and you have your system set to restore the previous session at power up.

With a man in the loop, the VMC application is not time-critical at all and may take its share of CPU time whenever it is offered. There is a lot of other stuff in a Linux system that also wants CPU time (look at `ps -A`). If your application is time-critical with predetermined response times at close tolerances between events, this scheme will not work for you. However, if you have a few milliseconds here and there to spare, Linux will host your monitoring and control applications with a reasonably small level of effort and good reliability.■

Rick Brown is a US Navy veteran, holds a BSEE granted in 1970 by the University of Florida, developed atmospheric research instruments for many years as a faculty member of the University of Nevada System, consulted in the private sector as a developer of electronic instruments and manufacturing test systems and now lives happily ever after on his little spread north of Reno, Nevada.

WEBCASTS

Learn the 5 Critical Success Factors to Accelerate IT Service Delivery in a Cloud-Enabled Data Center

Today's organizations face an unparalleled rate of change. Cloud-enabled data centers are increasingly seen as a way to accelerate IT service delivery and increase utilization of resources while reducing operating expenses. Building a cloud starts with virtualizing your IT environment, but an end-to-end cloud orchestration solution is key to optimizing the cloud to drive real productivity gains.

> **http://lnxjr.nl/IBM5factors**

Modernizing SAP Environments with Minimum Risk—a Path to Big Data

Sponsor: SAP | **Topic:** Big Data

Is the data explosion in today's world a liability or a competitive advantage for your business? Exploiting massive amounts of data to make sound business decisions is a business imperative for success and a high priority for many firms. With rapid advances in x86 processing power and storage, enterprise application and database workloads are increasingly being moved from UNIX to Linux as part of IT modernization efforts. Modernizing application environments has numerous TCO and ROI benefits but the transformation needs to be managed carefully and performed with minimal downtime. Join this webinar to hear from top IDC analyst, Richard Villars, about the path you can start taking now to enable your organization to get the benefits of turning data into actionable insights with exciting x86 technology.

> **http://lnxjr.nl/modsap**

WHITE PAPERS

White Paper: JBoss Enterprise Application Platform for OpenShift Enterprise

Sponsor: DLT Solutions

Red Hat's® JBoss Enterprise Application Platform for OpenShift Enterprise offering provides IT organizations with a simple and straightforward way to deploy and manage Java applications. This optional OpenShift Enterprise component further extends the developer and manageability benefits inherent in JBoss Enterprise Application Platform for on-premise cloud environments.

Unlike other multi-product offerings, this is not a bundling of two separate products. JBoss Enterprise Middleware has been hosted on the OpenShift public offering for more than 18 months. And many capabilities and features of JBoss Enterprise Application Platform 6 and JBoss Developer Studio 5 (which is also included in this offering) are based upon that experience.

This real-world understanding of how application servers operate and function in cloud environments is now available in this single on-premise offering, JBoss Enterprise Application Platform for OpenShift Enterprise, for enterprises looking for cloud benefits within their own datacenters.

> **http://lnxjr.nl/jbossapp**

WHITE PAPERS

Linux Management with Red Hat Satellite: Measuring Business Impact and ROI

Sponsor: Red Hat | Topic: Linux Management

Linux has become a key foundation for supporting today's rapidly growing IT environments. Linux is being used to deploy business applications and databases, trading on its reputation as a low-cost operating environment. For many IT organizations, Linux is a mainstay for deploying Web servers and has evolved from handling basic file, print, and utility workloads to running mission-critical applications and databases, physically, virtually, and in the cloud. As Linux grows in importance in terms of value to the business, managing Linux environments to high standards of service quality — availability, security, and performance — becomes an essential requirement for business success.

> **http://lnxjr.nl/RHS-ROI**

Standardized Operating Environments for IT Efficiency

Sponsor: Red Hat

The Red Hat® Standard Operating Environment SOE helps you define, deploy, and maintain Red Hat Enterprise Linux® and third-party applications as an SOE. The SOE is fully aligned with your requirements as an effective and managed process, and fully integrated with your IT environment and processes.

Benefits of an SOE:

SOE is a specification for a tested, standard selection of computer hardware, software, and their configuration for use on computers within an organization. The modular nature of the Red Hat SOE lets you select the most appropriate solutions to address your business' IT needs.

SOE leads to:

- Dramatically reduced deployment time.
- Software deployed and configured in a standardized manner.
- Simplified maintenance due to standardization.
- Increased stability and reduced support and management costs.
- There are many benefits to having an SOE within larger environments, such as:
  - Less total cost of ownership (TCO) for the IT environment.
  - More effective support.
  - Faster deployment times.
  - Standardization.

> **http://lnxjr.nl/RH-SOE**

Hacking a Safe with Bash

By combining simple tools, you can build an effective safe to store your sensitive data.

ADAM KOSMIN

Through the years, I have settled on maintaining my sensitive data in plain-text files that I then encrypt asymmetrically. Although I take care to harden my system and encrypt partitions with LUKS wherever possible, I want to secure my most important data using higher-level tools, thereby lessening dependence on the underlying system configuration. Many powerful tools and utilities exist in this space, but some introduce unacceptable levels of "bloat" in one way or another. Being a minimalist, I have little interest in dealing with GUI applications that slow down my work flow or application-specific solutions (such as browser password vaults) that are applicable only toward a subset of my

Asymmetric Encryption

Asymmetric encryption, or public-key cryptography, relies on the use of two keys: one of which is held private, while the other is published freely. This model offers greater security over the symmetric approach, which is based on a single key that must be shared between the sender and receiver. GnuPG is a free software implementation of the OpenPGP standard as defined by RFC4880. GnuPG supports both asymmetric and symmetric algorithms. Refer to https://gnupg.org for additional information.

GPG

This article makes extensive use of GPG to interact with files stored in your safe. Many tutorials and HOWTOs exist that will walk you through how to set up and manage your keys properly (https://www.gnupg.org/documentation/index.html). It is highly recommended to configure gpg-agent in order to avoid having to type your passphrase each time you interact with your private key. One popular approach used for this job is Keychain (http://www.funtoo.org/Keychain), because it also is capable of managing ssh-agent.

sensitive data. Working with text files affords greater flexibility over how my data is structured and provides the ability to leverage standard tools I can expect to find most anywhere.

Let's take the classic example of managing credentials. This is a necessary evil and while both pass (http://www.passwordstore.org) and KeePassC (http://raymontag.github.io/keepassc) look interesting, I am not yet convinced they would fit into my work flow. Also, I am definitely not lulled by any "copy to clipboard" feature. You've all seen the inevitable clipboard spills on IRC and such—no thanks! For the time being, let's fold this job into a "safe" concept by managing this data in a file. Each line in the file will conform to a simple format of:

```
resource:userid:password
```

Where "resource" is something mnemonic, such as an FQDN or even a hardware device like a router that is limited to providing telnet access. Both `userid` and `password` fields are represented as hints. This hinting approach works nicely given my conscious effort to limit the number of user IDs and passwords I routinely use. This means a hint is all that is needed for muscle memory to kick in. If a particular resource uses some exotic complexity rules, I quickly can understand the slight variation by modifying the hint accordingly. For example, a hint of "fo" might end up as "!fo" or "fO". Another example of achieving this balance between security and usability comes up when you need to use an especially long password. One practical solution would be to combine familiar passwords and document the hint accordingly. For example, a hint representing a combination of "fo" and "ba" could be represented as "fo..ba". Finally, the hinting approach provides reasonable fall-back protection since the limited information would be of little use to an intruder.

Despite the obscurity, leaving this data in the clear would be silly and irresponsible. Having GnuPG configured provides an opportunity to encrypt the data using your private key. After creating the file, my work flow was looking something like this:

```
$ gpg --ear <my key id> <file>
$ shred -u <file>
```

Updating the file would involve decrypting, editing and repeating the steps above. This was tolerable for a while since, practically speaking, I'm not establishing credentials on a daily basis. However, I knew the day would eventually come when the tedious routine would become too much of a burden. As expected, that day came when I found myself keeping insurance-related notes that I then considered encrypting using the same technique. Now, I am talking about managing multiple files—a clear sign that it is time to write a script to act as a wrapper. My requirements were simple:

1. Leverage common tools, such as GPG, shred and bash built-ins.

2. Reduce typing for common operations (encrypt, decrypt and so on).

3. Keep things clean and readable in order to accommodate future growth.

4. Accommodate plain-text files but avoid having to micro-manage them.

Interestingly, the vim-gnupg Vim plugin (https://github.com/jamessan/vim-gnupg) easily can handle these requirements, because it integrates seamlessly with files ending in .asc, .gpg or .pgp extensions. Despite its abilities, I wanted to avoid having to manage multiple encrypted files and instead work with a higher-level "vault" of sorts. With that goal in mind, the initial scaffolding was cobbled together:

```
#!/bin/bash

CONF=${HOME}/.saferc
[ -f $CONF ] && . $CONF
[ -z "$SOURCE_DIR" ] && SOURCE_
DIR=${HOME}/safe
SOURCE_BASE=$(basename $SOURCE_DIR)
TAR_ENC=$HOME/${SOURCE_BASE}.tar.gz.asc
TAR="tar -C $(dirname $SOURCE_DIR)"

usage() {
cat <<EOF
Usage: $(basename $0) OPTION
Options:
-c     create the safe
-x     extract contents
EOF
}

create_safe() {
}

extract_safe() {
}

[ $# -ge 1 ] || { usage; exit 1; }

while getopts "cx" opt; do
  case $opt in
    c)
      create_safe
      ;;
    x)
      extract_safe
      ;;
    *)
      usage
      exit 1
      ;;
  esac
done
```

This framework is simple enough to build from and establishes some ground rules. For starters, you're going to avoid micro-managing files by maintaining them in a single tar archive. The `$SOURCE_DIR` variable will fall back to $HOME/safe unless it is defined in ~/.saferc. Thinking ahead, this will allow people to collaborate on this project without clobbering the variable over and over. Either way, the value of `$SOURCE_DIR` is used as a base for the `$SOURCE_BASE`, `$TAR_ENC` and `$TAR` variables. If my ~/.saferc were to define `$SOURCE_DIR` as $HOME/foo, my safe will be maintained as $HOME/foo.tar.gz.asc. If I choose not to maintain a ~/.saferc file, my safe will reside in $HOME/safe.tar.gz.asc.

Back to this primitive script, let's limit the focus simply to being able to open and close the safe. Let's work on the `create_safe()` function first so you have something to extract later:

```
create_safe() {
  [ -d $SOURCE_DIR ] || { "Missing directory: $SOURCE_DIR"; exit 1; }
  $TAR -cz $SOURCE_BASE | gpg -ear $(whoami) --yes -o $TAR_ENC
  find $SOURCE_DIR -type f | xargs shred -u
  rm -fr $SOURCE_DIR
}
```

The `create_safe()` function is looking pretty good at this point, since it automates a number of tedious steps. First, you ensure that the archive's base directory exists. If so, you compress the directory into a tar archive and pipe the output straight into GPG in order to encrypt the end result. Notice how the result of `whoami` is used for GPG's `-r` option. This assumes the private GPG key can be referenced using the same ID that is logged in to the system. This is strictly a convenience, as I have taken care to keep these elements in sync, but it will need to be modified if your setup is different. In fact, I could see eventually supporting an override of sorts with the ~/.saferc approach. For now though, let's put that idea on the back burner. Finally, the function calls the shred binary on all files within the base directory. This solves the annoying "Do I have a plain-text version laying around?" dilemma by automating the cleanup.

Now you should be able to create the safe. Assuming no ~/.saferc exists and the $PATH environment variable contains the directory containing safe.sh, you can begin to test this script:

```
$ cd
$ mkdir safe
$ for i in $(seq 5); do echo "this is secret #$i" >
➥safe/file${i}.txt; done
$ safe.sh -c
```

You now should have a file named safe.tar.gz.asc in your home directory. This is an encrypted tarball containing the five files previously written to the ~/safe directory. You then cleaned things up by shredding each file and finally removing the ~/safe directory. This is probably a good time to recognize you are basing the design around an expectation to manage a single directory of files. For my purposes, this is acceptable. If subdirectories are needed, the code would need to be refactored accordingly.

Now that you are able to create your safe, let's focus on being able to open it. The following `extract_safe()` function will do the trick nicely:

```
extract_safe() {
  [ -f $TAR_ENC ] || { "Missing file: $TAR_ENC"; exit 1; }
  gpg --batch -q -d $TAR_ENC | $TAR -zx
}
```

Essentially, you are just using GPG and tar in the opposite order. After opening the safe by running the script with `-x`, you should see the ~/safe directory.

Things seem to be moving along, but you easily can see the need to list the contents of your safe, because you

Listing 1. safe.sh

```
#!/bin/bash
#
# safe.sh - wrapper to interact with my encrypted file archive

CONF=${HOME}/.saferc
[ -f $CONF ] && . $CONF
[ -z "$SOURCE_DIR" ] && SOURCE_DIR=${HOME}/safe
SOURCE_BASE=$(basename $SOURCE_DIR)
TAR_ENC=$HOME/${SOURCE_BASE}.tar.gz.asc
TAR="tar -C $(dirname $SOURCE_DIR)"

usage() {
cat <<EOF
Usage: $(basename $0) OPTION
Options:
-c        create the safe
-x        extract contents
-l        list contents
-b HOST   backup (scp) to HOST. Multiple -b options are supported
EOF
}

is_or_die() {
  if [ ! -d ${1:-$TAR_ENC} -a ! -f ${1:-$TAR_ENC} ]; then
    echo "Unknown or missing: ${1:-$TAR_ENC}"
    exit 1
  fi
}

shred_source_dir() {
  find $SOURCE_DIR -type f | xargs shred -u
  rm -fr $SOURCE_DIR
}

list_safe() {
  is_or_die
  gpg --batch -q -d $TAR_ENC | tar -zt
}

extract_safe() {
  is_or_die
  OPTS=" -zx"
  [ $# -eq 1 ] && OPTS+=" $SOURCE_BASE/$1 -O"
  gpg --batch -q -d $TAR_ENC | $TAR $OPTS
}

create_safe() {
  is_or_die $SOURCE_DIR
  $TAR -cz $SOURCE_BASE | gpg -ear $(whoami) --yes -o $TAR_ENC
  shred_source_dir
}

[ $# -ge 1 ] || { usage; exit 1; }

while getopts "lxcb:" opt; do
  case $opt in
    l)
      list_safe
      ;;
    x)
      extract_safe
      ;;
    c)
      create_safe
      ;;
    b)
      BACKUP_HOSTS+=("$OPTARG")
      is_or_die
      ;;
    *)
      usage
      exit 1
      ;;
  esac
done

for BACKUP_HOST in ${BACKUP_HOSTS[@]}; do
  echo -en "Copying to $BACKUP_HOST... "
  scp $TAR_ENC ${BACKUP_HOST}: &> /dev/null
  [ $? -eq 0 ] && echo OK || echo Failed
done
```

do not want to have to open it each time in order to know what is inside. Let's add a `list_safe()` function:

```
list_safe() {
  [ -f $TAR_ENC ] || { "Missing file: $TAR_ENC"; exit 1; }
  gpg --batch -q -d $TAR_ENC | tar -zt
}
```

No big deal there, as you are just using tar's ability to list contents rather than extract them. While you are here, you can start DRYing this up a bit by consolidating all the file and directory tests into a single function. You even can add a handy little backup feature to `scp` your archive to a remote host. Listing 1 is an updated version of the script up to this point.

The new `-b` option requires a hostname passed as an argument. When used, the archive will be `scp`'d accordingly. As a bonus, you can use the `-b` option multiple times in order to back up to multiple hosts. This means you have the option to configure a routine cron job to automate your backups while still being able to run a "one off" at any point. Of course, you will want to manage your SSH keys and configure ssh-agent if you plan to automate your backups. Recently, I have converted over to pam_ssh (https://wiki.archlinux.org/index.php/SSH_keys#pam_ssh) in order to fire up my ssh-agent, but that's a different discussion.

Back to the code, there is a small `is_or_die()` function that accepts an argument but falls back to the archive specified in `$TAR_ENC`. This will help keep the script lean and mean since, depending on the option(s) used, you know you are going to want to check for one or more files and/or directories before taking action.

For the remainder of this article, I'm going to avoid writing out the updated script in its entirety. Instead, I simply provide small snippets as new functionality is added.

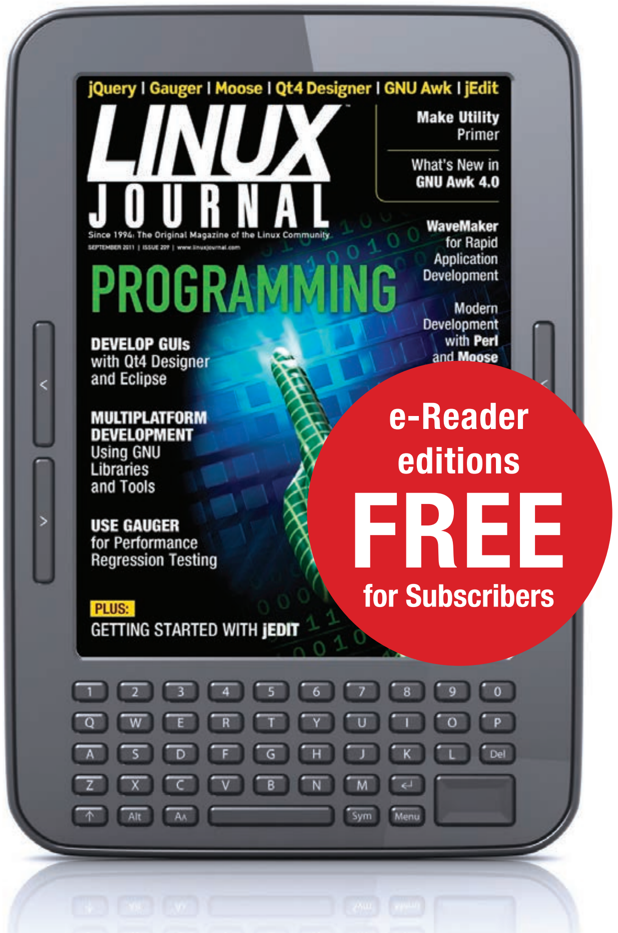
For starters, how about adding the ability to output the contents of a single file being stored in your safe? All you would need to do is check for the file's presence and modify your tar options appropriately. In fact, you have an opportunity to avoid re-inventing the wheel by simply refactoring your `extract_safe()` function. The updated function will operate on a single file if called accordingly. Otherwise, it will operate on the entire archive. Worth noting is the extra step to provide a bit of user-friendliness. Using the default `$SOURCE_DIR` of ~/safe, the user can pass either safe/my_file or just my_file to the -o option:

```
list_safe() {
  is_or_die
  gpg --batch -q -d $TAR_ENC | tar -zt | sort
}

search_safe() {
  is_or_die
  FILE=${1#*/}
  for f in $(list_safe); do
    ARCHIVE_FILE=${f#$SOURCE_BASE/}
    [ "$ARCHIVE_FILE" == "$FILE" ] && return
  done
  false
}

extract_safe() {
  is_or_die
  OPTS=" -zx"
  [ $# -eq 1 ] && OPTS+=" $SOURCE_BASE/${1#*/} -O"
  gpg --batch -q -d $TAR_ENC | $TAR $OPTS
}
```

The final version of safe.sh is maintained at https://github.com/windowsrefund/safe. It supports a few more use cases, such as the ability to add and remove files. When adding these features, I tried to avoid actually having to extract the archive to disk as a precursor to modifying its contents. I was unsuccessful due to GNU tar's refusal to read from STDIN when `-r` is used. A nice alternative to connecting GPG with tar via pipes might exist in GnuPG's gpg-zip binary. However, the Arch package maintainer appears to have included only the gpg-zip man page. In short, I prefer the "keep things as simple as possible; but no simpler" approach. If anyone is interested in improving the methods used to add and remove files, feel free to submit your pull requests. This also applies to the `edit_safe()` function, although I foresee refactoring that at some point given some recent activity with the vim-gnupg plugin (https://github.com/jamessan/vim-gnupg/issues/24).



In general, these credentials simply could be hard-coded in one or more configuration files but that would lead to shame, regret and terrible things.

Integrating with Mutt

My MUA of choice is mutt. Like many people, I have configured my mail client to interact with multiple IMAP accounts, each requiring authentication. In general, these credentials simply could be hard-coded in one or more configuration files but that would lead to shame, regret and terrible things. Instead, let's use a slight variation of Aaron Toponce's clever approach (https://pthree.org/2012/01/07/encrypted-mutt-imap-smtp-passwords) that empowers mutt with the ability to decrypt and source sensitive data:

```
$ echo "set my_pass_imap = l@mepassw0rd" > /tmp/pass_mail
$ safe.sh -a /tmp/pass_mail
```

Now that your safe contains the pass_mail file; you have mutt read it with this line in your ~/.muttrc:

```
source "safe.sh -o pass_mail |"
```

By reading the file, mutt initializes a variable you have named `my_pass_imap`. That variable can be used in other areas of mutt's configuration. For example, another area of your mutt configuration can use these lines:

```
set imap_user = "my_user_id"
set imap_pass = $my_pass_imap
set folder = "imaps://example.com"
set smtp_url = smtp://$imap_user:$imap_pass@example.com
```

By combining appropriately named variables with mutt's ability to support multiple accounts, it is possible to use this technique to manage all of your mail-related credentials securely while never needing to store plain-text copies on your hard drive.■

Adam Kosmin works as a Sr. Systems Engineer for Sailthru where he focuses on automation and configuration management. He has presented at PuppetConf on two occasions and is an avid supporter of the Free Software initiative. When not coding, tweaking or building something, he hangs out with his cat buddies: Evil and Handsome.

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

drupalize.me

Instant Access to Premium Online Drupal Training

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

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

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

```
 *
 * Usage: $(obj).insertAtCursor(text);
 *   obj = a textarea or textfield
 *   text = a string to insert
 */
$.fn.extend({
  insertAtCaret: function(myValue){
    return $(this).each(function(){
      // If target element is hidden, don't do it.
      if ($(this).is(':hidden') || $(this).parents(':hidde
        return;
      }
      if (document.selection) {
        this.fo
        sel = c
        sel.tex
        this.fc
      }
      else if (
        var sta
        var end
        var scr
        this.va
        this.fo
        // High
```

DOC SEARLS

An Easy Way to Pay for Journalism, Music and Everything Else We Like

All we need is to unbox business as usual.

Some of us work for money. Some of us work for love. Some of us work for both, or just because we feel compelled or obliged.

As a journalist, I work for all those reasons, except money—meaning I write as much as I ever did, but I rarely get paid for it. So I make up the difference with other kinds of work for which writing and talking are still marketable skills.

The same goes for nearly every journalist I knew back in the heyday of the trade, before everything got sucked into the Net.

Now, thanks to the Net, there is more news than ever, from more sources than ever, coming through more channels than ever. Could be there's more money than ever paying for it too. I don't know. I also don't know what percentage of the total flow is news or journalism.

We hate the term "content". It sounds like packing material, or container cargo. But hey, that's the world we live in now.

Now it's all just "content".

No journalist of any quality (other than bad) thinks what he or she produces is "content". Nor does any musician, or artist of any kind. We hate the term "content". It sounds like packing material, or container cargo. But hey, that's the world we live in now.

It's a good thing the Net we have today is only about 20 years old (1995 was when the NSFNET was decommissioned, opening the whole Net to commercial activity), and all businesses and business models are provisional anyway. They may do for now, but none of them foreclose future possibilities—just as no breed of commercial UNIX (launched by AT&T in 1973) foreclosed the possibility of free and open alternatives, such as Linux. (Historical note: *Linux Journal* was launched alongside Linux 1.0 in 1994. And we know what's happened to UNIX and Linux since then.)

So let's look at business models for free (as in beer) artistic goods as a blank slate.

I know: we can't ignore copyright laws, rights-clearing norms and industries (publishing, Hollywood) that call folks like us "pirates" and hold captive regulators by the short hairs. None of that will matter if we make all of them happy by finding a way to get them—and the artists they represent—*more* money than they've been getting by coercive means. If the answer is money, the only question is how.

Let's start with this thesis: *Information wants to be free, but value wants to be paid for.*

If that's true, the trick is to make it as easy as possible for anybody to pay for anything they like, wherever they find it, even if that thing is as small as a like, a tweet, a post, a graffito or a tune heard just one time—and even if all anybody wants to pay for that thing is a penny.

We do have proof that lots of people will pay for things they can get for free. Apple, for example, proved that people were willing to pay 99 cents for a song they could also get for $0 on Napster or Limewire. I've also conducted tests of value by asking audiences at my talks in the US if any of them listen to public radio. Usually most hands go up. Then I ask how many of those listeners pay for the privilege. About 10% of the hands stay up. Then I ask how many would pay if it was real easy. The number of hands doubles.

Here's what I suggest for an approach:

1. It needs be free-as-in-freedom as well as free-as-in-beer.

2. It should make it easy for anybody to pay any amount for anything they like. Even if it's not a thing (a song, for example).

3. It should be programmable. So, for example, one could set things up so it's as easy to throw a penny (or more) toward anything, anywhere, as it is to like something on Facebook.

4. It should be a capability that can be added to lots of different things, rather than a standalone thing, such as an app.

5. It should make use of APIs. That way, for example, one might be able to throw a penny (or more) at every tune one tags with the Shazam app.

Those are just off the top of my head. Add more of your own. It's easy if you think outside the boxes of business-as-usual and context-as-usual (such as Facebook, Twitter, Apple and other silo'd sites and services—or thinking "there needs to be an app for that").

It also helps to think inside the boxes that have produced millions of free and open-source code bases, and useful standards and protocols. Those are the boxes where the base imperative is making stuff as useful as possible for every purpose to which that stuff can possibly be put.

As it happens, I proposed one of these a few years back. It's called EmanciPay, currently described this way on the ProjectVRM wiki:

> EmanciPay is a payment framework for customers operating with full agency in the open marketplace. It operates on open protocols and standards, so it can be used by any buyer, seller or intermediary. Simply put, EmanciPay makes it easy for

anybody to pay (or offer to pay) —

- as much as they like
- however they like
- for whatever they like
- on their own terms

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

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

While EmanciPay was first conceived by ProjectVRM as a way to make live payments to nonprofits and online publishers, it is also positioned as a counterpart to sellers' subscription systems in what Zuora calls the "subscription economy" (https://www.zuora.com/what-is-zuora), which it says "is built

Advertiser Index

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

ADVERTISER	URL	PAGE #
Drupalcon Los Angeles	https://events.drupal.org/losangeles2015	19
Drupalize.me	http://drupalize.me	89
EmperorLinux	http://www.emperorlinux.com	17
NetGate	http://www.netgate.com	7
O'Reilly Velocity	http://velocityconf.com/web-mobile-business-conf-2015	15
O'Reilly Solid	http://solidcon.com/internet-of-things-2015	37
Peer 1 Hosting	http://go.peer1.com/linux	21
SUSE	http://suse.com	3
WITI SUMMIT	http://www.witi.com/summit	27

ATTENTION ADVERTISERS

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

on ever changing relationships with your customers". Since relationships are two-way by nature, EmanciPay is one way that customers can manage their end, while sell-side systems such as Zuora's manage the other.

In "EmanciPay: A Content Monetization Plan for Newspapers", I say:

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

I think EmanciPay can work as what Bruce Sterling calls a design fiction:

A formal definition exists: "Design fiction is the deliberate use of diegetic prototypes to suspend disbelief about change."

There's heavy freight in that sentence, but most can be disposed of promptly. "Deliberate use" means that design fiction is something that people do with a purpose.

"Diegetic" is from film and theatre studies. A movie has a story, but it also has all the commentary, scene-setting, props, sets and gizmos to support that story. Design fiction doesn't tell stories—instead, it designs prototypes that imply a changed world.

It's nearly impossible for any threatened businesses to imagine what will disrupt them—especially when they are busy fighting disruptions full-time already. This is why publishing, entertainment and other content-pumping businesses can't grok the likes of EmanciPay—at least not until they see a prototype such as might be produced in a design fiction exercise.

But geeks who have already changed the world are in a much better position to imagine out the possibilities, and actually assemble the code required for it. That's why I'm talking about it here.■

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.

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

Resources

EmanciPay: **http://cyber.law.harvard.edu/projectvrm/EmanciPay**

Project VRM Wiki: **http://cyber.law.harvard.edu/projectvrm/Main_Page**

Zuora: **https://www.zuora.com**

Subscription Economy: **https://www.zuora.com/what-is-zuora**

"EmanciPay: A Content Monetization Plan for Newspapers": **http://blogs.law.harvard.edu/vrm/2009/05/28/emancipay-a-content-monetization-plan-for-newspapers**

CRM: **http://en.wikipedia.org/w/index.php?title=Customer_relationship_management**

Bruce Sterling: **http://en.wikipedia.org/wiki/Bruce_Sterling**

"Patently untrue: fleshy defibrillators and synchronized baseball are changing the future": **http://www.wired.co.uk/magazine/archive/2013/10/play/patently-untrue**

NFSNET:
http://en.wikipedia.org/wiki/National_Science_Foundation_Network#Commercial_traffic

History of the Internet: **http://en.wikipedia.org/wiki/History_of_the_Internet**