LINUX JOURNAL™

Since 1994: The Original Magazine of the Linux Community

NETWORK SECURITY and SSH What You Need to Know

OCTOBER 2014 | ISSUE 246 | www.linuxjournal.com

EMBEDDED

HOW-TO: Sump Pump Monitor with Raspberry Pi and Python

AN INDEPTH LOOK at the U-Boot Environment Anatomy

+

Interview with the Creators of the "Hello World Program"

Advanced Vim Macro Techniques

Use the BeagleBone Black TO COMMAND YOUR ELECTRONIC GEAR presented by IT-oLogy™

ALL THINGS OPEN™

OCT 22-23, 2014 | RALEIGH CONVENTION CENTER

In Raleigh and the Research Triangle: A Global Epicenter of Technology and Open Source Innovation

A CONFERENCE EXPLORING OPEN SOURCE, OPEN TECH AND THE OPEN WEB IN THE ENTERPRISE.

LEADING TECHNOLOGY AND BUSINESS LEADERS WILL PARTICIPATE

SEAN MARTELL
Mozilla

DOUG CUTTING
Cloudera

JAMES TURNBULL
Docker

KAREN SANDLER
Software Freedom Conservancy

DWIGHT MERRIMAN
MongoDB

YEHUDA KATZ
Tilde

ELIZABETH JOSEPH
Hewlett Packard

STEVEN VUGHAN-NICHOLS
ComputerWorld & CBS/ZDNet

Two days of keynotes, talks, tutorials, workshops and networking opportunities in Raleigh and the Research Triangle area

REGISTER NOW!

ALLTHINGSOPEN.ORG

SILICON MECHANICS
SERVERS • STORAGE • HPC

Are you tired of dealing with proprietary storage?

zStax®
ZFS Unified Storage
Powered by NexentaStor

zStax® StorCore ZFS Unified Storage from Silicon Mechanics is truly software defined storage.

From modest data storage needs to a multi-tiered production storage environment, the **zStax StorCore** ZFS unified storage appliances have the right mix of performance, capacity, and reliability to fit your needs.

October
Featured
Download

Achieving Enterprise Class Storage in the Cloud with Software Defined Storage

zStax StorCore 64

zStax StorCore 104

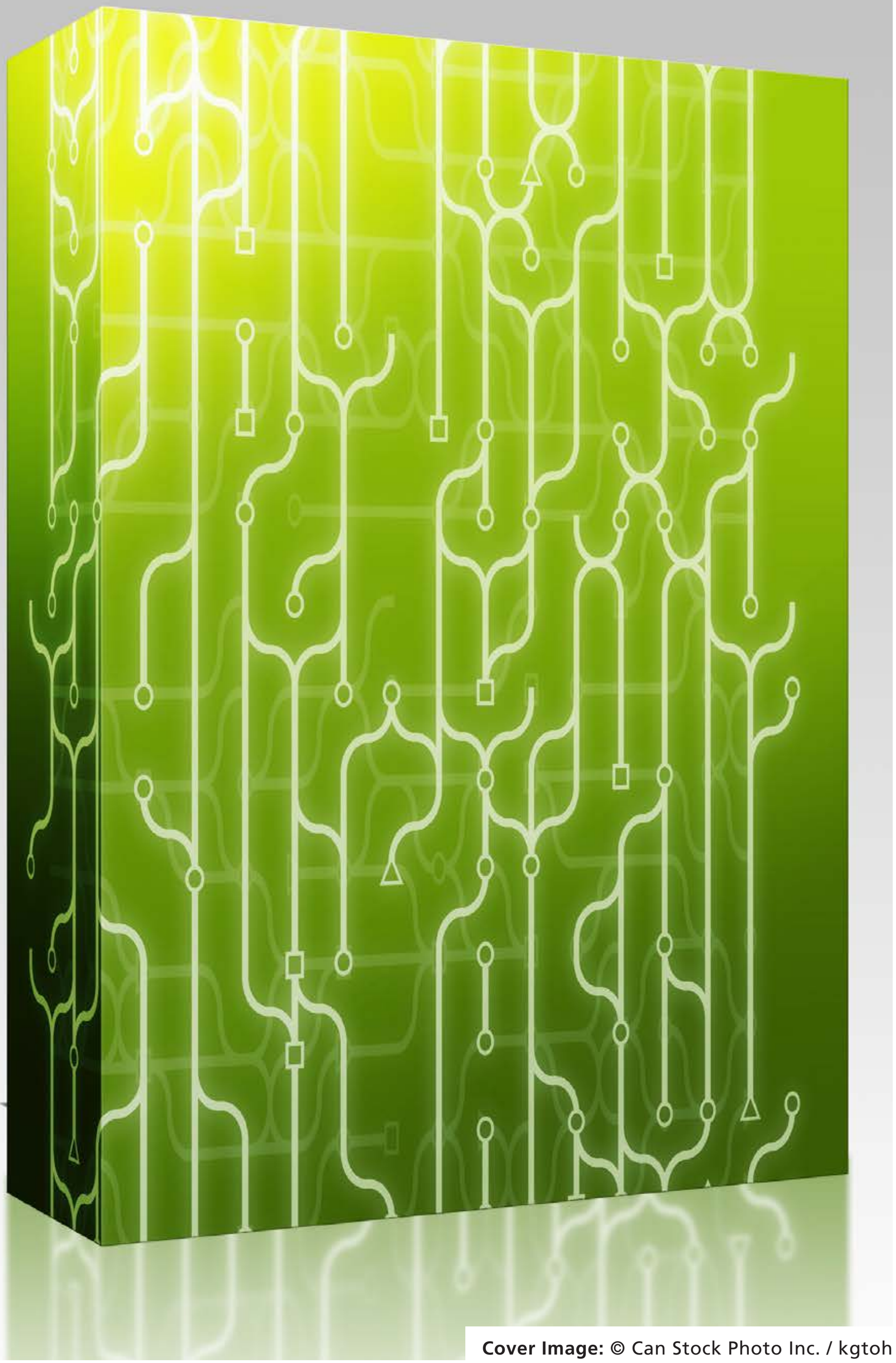
Talk with an expert today: 866-352-1173 | www.siliconmechanics.com/zstax

CONTENTS

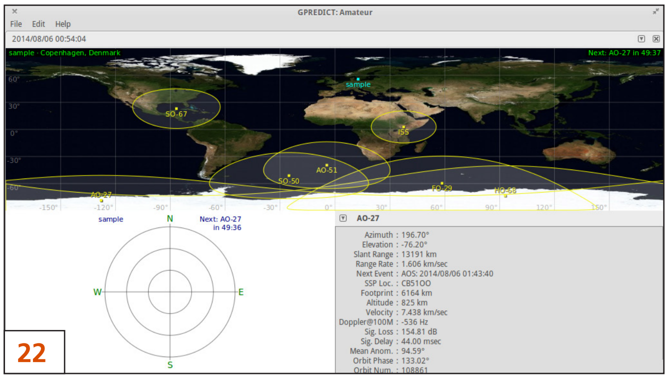
OCTOBER 2014
ISSUE 246

EMBEDDED

FEATURES

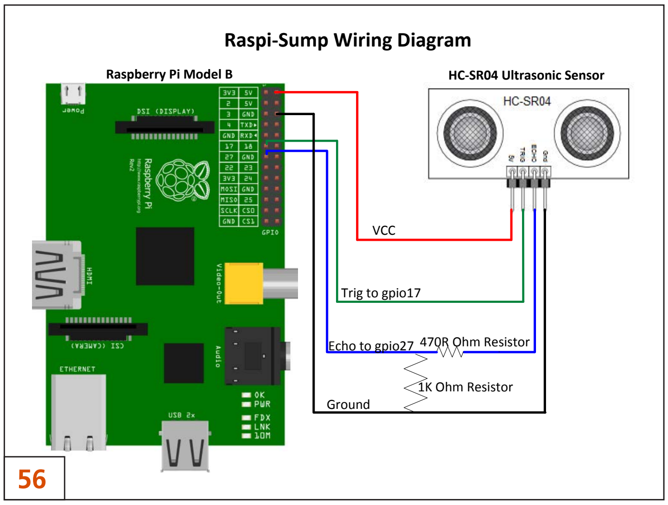


Cover Image: © Can Stock Photo Inc. / kgtoh

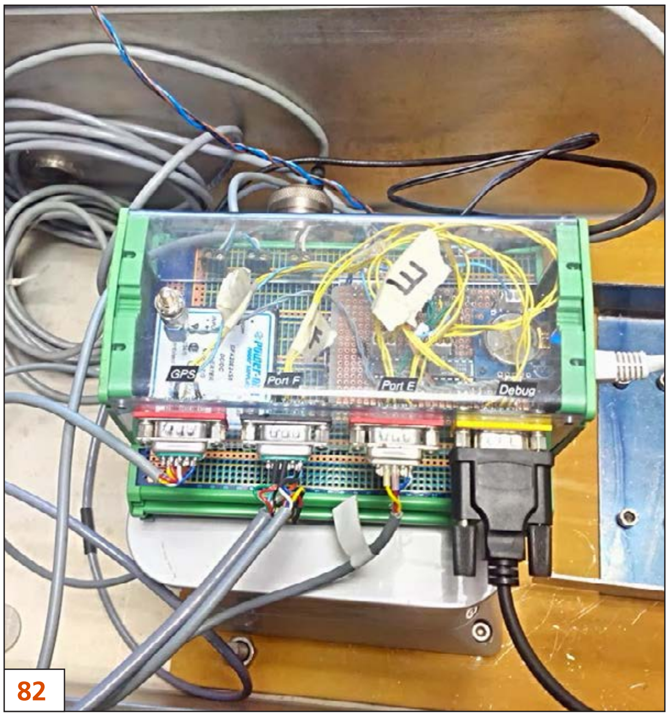


```
AO-27
      Azimuth : 196.70°
    Elevation : -76.20°
  Slant Range : 13191 km
   Range Rate : 1.606 km/sec
   Next Event : AOS: 2014/08/06 01:43:40
     SSP Loc. : CB51OO
    Footprint : 6164 km
     Altitude : 825 km
     Velocity : 7.438 km/sec
Doppler@100M : -536 Hz
    Sig. Loss : 154.81 dB
   Sig. Delay : 44.00 msec
  Mean Anom. : 94.59°
  Orbit Phase : 133.02°
   Orbit Num. : 108861
```

22



56



82

ON THE COVER
- Network Security and SSH—What You Need to Know, p. 46
- How-To: Sump Pump Monitor with Raspberry Pi and Python, p. 56
- An Indepth Look at the U-Boot Environment Anatomy, p. 68
- Interview with the Creators of the "Hello World Program", p. 110
- Advanced Vim Macro Techniques, p. 42
- Use the BeagleBone Black to Command Your Electronic Gear, p. 82

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.

LINUX JOURNAL™

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

ENJOY:

Timely delivery

Off-line reading

Easy navigation

Phrase search and highlighting

Ability to save, clip and share articles

Embedded videos

Android & iOS apps, desktop and e-Reader versions

SUBSCRIBE TODAY!

LINUX JOURNAL

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

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

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

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

Editorial Advisory Panel
Brad Abram Baillio • Nick Baronian • Hari Boukis • Steve Case
Kalyana Krishna Chadalavada • Brian Conner • Caleb S. Cullen
Keir Davis • Michael Eager • Nick Faltys • Dennis Franklin Frey
Victor Gregorio • Philip Jacob • Jay Kruizenga • David A. Lane
Steve Marquez • Dave McAllister • Carson McDonald • Craig Oda
Jeffrey D. Parent • Charnell Pugsley • Thomas Quinlan • Mike Roberts
Kristin Shoemaker • Chris D. Stark • Patrick Swartz • James Walker

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

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

LINUX is a registered trademark of Linus Torvalds.

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.fo
      }
      else if (
        var sta
        var end
        var scr
        this.va
        this.fo
        // High
```

SHAWN POWERS

Linux Inside! Linux Inside!

Linux inside, Linux inside, every single one of us has Linux inside! (Sung to the tune of “Devil Inside” by INXS—you’re welcome for the ear worm, I can’t get it out of my head either.)

In the world of embedded systems, systems-on-chip and single-purpose hardware solutions, it’s easier to list the products that don’t have Linux as their operating system! This month, we focus on putting Linux into tiny places, and that means everything from the tiniest “Android Wear” watch to the ubiquitous Raspberry Pi. As Linux users, we’ve been sneaking open source into server rooms for decades. Now we get to sneak it in everywhere!

This issue starts with Reuven M. Lerner discussing the books he’s *embedded* into his bookshelf this year. Month after month, Reuven gives us incredible tips, techniques and training for programmers and neophytes like myself. Now we get a glimpse at some of the books he uses for learning himself. Whether you want advice on language books, database books or even some information on freelancing, Reuven shares his stash. If you’re looking for your next book to read (after this issue of *Linux Journal*, of course), check out his column.

VIDEO:
Shawn Powers runs through the latest issue.

Dave Taylor begins a new adventure, this time attempting to parse the written word and grammar with scripts. It’s always fascinating to see tasks that are simple for humans attempted by programs and logic. Dave’s column is more than just instructive, it will force you to think as well! Speaking of thinking, the title of Kyle Rankin’s column this month will make you think hard. “Return of the Mac” seems like the exact opposite of what Kyle would write about. And, of course, he doesn’t talk about an Apple product. I won’t give it away, but don’t worry, Kyle hasn’t given up the command line for the GUI world of Apple—in fact, quite the opposite.

I put on my grey hat this month and talk a little bit about SSH. "Hacking" is a word that is used to describe far too many things, most of which I think are more accurately described as, "Using Things Like They Were Made To Be Used", but that's not as catchy for a movie title. Firewalls are important security tools for any network, but it's important to realize that firewalls can't be your only defense against access. This month I show you why. Do I "hack" the firewall? I don't think so, but for someone unfamiliar with SSH tunneling, it sure looks like it.

Al Audet shows how to embed Linux into a hole in the ground this month. More specifically, he explains how he uses a Raspberry Pi device to monitor the sump hole in his basement to avoid flooding. Sump pumps are amazing devices—until they're not. Al shares not only his method, but also his scripts for controlling a sensor that warns him if the water level rises too much. It's the perfect example of solving difficult problems with Linux. Sachin Verma goes even deeper (pun intended) and shows how to deal with the bootloader on embedded Linux systems. U-Boot is commonly used in such environments, and along with booting the kernel, you can pass environment variables as well. Sachin walks through the boot process and teaches how to customize the information passed on at boot.

Finally, Samuel Bucquet gives a crash course in accessing the BeagleBone Black's I/O ports via Python. Although the Raspberry Pi and BeagleBone Black are both excellent platforms for embedded projects, the BBB's sheer number of I/O options make it incredible for projects with a large number of sensor needs. With Samuel's help, you can access those ports with the friendly and familiar Python language.

The Embedded issue of *Linux Journal* always makes me want to build something. Now that BirdCam is off-line (the battery in my old Android phone blew up—it was ugly), maybe it's time to start over from scratch. I'm sure there are some great weatherproof IP cameras out there I could use to get better shots of the winter birds. Either way, this is a great issue, and we had a great time putting together for you. Whether you want tech tips, programming ideas or just my silly insight on the best new apps for your phone, the October issue of *Linux Journal* aims to please! ■

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
AN IN-DEPTH LOOK AT ZFS vs. BTRFS
HOW-TOs
Ease the Pain of Provisioning X.509 Certificates
Synchronize Your Life with ownCloud
Check Your Exchange InBox from the Command Line
+ 12-Factor, Scalable, Maintainable WEB APPS
pi-web-agent A DESKTOP ENVIRONMENT FOR THE RASPBERRY PI
DNSMasq A HERO FOR SERVERS
OpenAxiom A COMPUTER SYSTEM FOR ALGEBRA

Two-Column Format

I love being able to read *Linux Journal* on my laptop, but I can't stand the two-column format that is used in the PDF version. I need to magnify the text due to poor eyesight, and the two-column format forces me to scroll down and then up constantly in order to read all the text on a page. A one-column format for the PDF would make for much smoother reading and navigation.

Please don't suggest I read the EPUB version. EPUB is a terrible format.
—Scott Randby

It's tough to come up with a perfect format. The PDF version is designed to look like the traditional magazine version, and it looks best (I think) on a large format color tablet. If you look back, the older PDF versions were even more difficult to read on a computer screen, as the layout mirrored the paper version exactly.

*I know you said you don't like the EPUB version, but I'm not sure whether you've tried reading the EPUB on a computer screen or only on eReaders. Using something like Readium (*http://www.readium.org*), you can change the font size and read the text in a single column. Apart from that, I'm not sure what else to suggest. I'm not sure what you dislike about the EPUB version, so I don't know how to help there if Readium doesn't work for you.—Shawn Powers*

Subscription Question

I carry two tablets, one iOS and the other Android. If I subscribe on my iPad to *Linux Journal*, can I also read it on my Nexus 7?
—Greg

If you subscribe directly through Linux Journal*, we will send you monthly notices with download links for .epub, .pdf, .mobi and browser versions of the magazine. You also will get access* *through our Android or iOS apps.*

We want our subscribers to be able to access the issues in any format that is available.

If you order through Google Play or iTunes, you will receive only those versions. Both have subscription options, but they are pretty much a closed environment.

We want you to be able to access the magazine any way you choose. If you subscribed through Google Play or iTunes, please send me your transaction ID, e-mail address and full mailing address (we use that to encrypt the files) to gm@linuxjournal.com. Once I have that, I can add you to our subscription database as well.

The best deal I have for Linux Journal *directly is* https://www.pubservice.com/Subnew2page.aspx?PC=LJ&PK=M48RENN.

I hope that helps. Sorry that it's more complicated than we would like, but once iTunes and Android closed their subscription environments, it became impossible to extract those orders and automatically add them to our database so readers can receive every format we have.—Mark Irgang, Publisher

Leap Years in Bash—or Anywhere

Regarding the detection of leap years in Dave Taylor's recent articles, there is a *very simple* way to determine whether a year is a leap year: if it's divisible by 4 and not 100. So a simple modulus function will suffice:

```
YEAR=$( date "+%Y" )
if [ $(( ${YEAR} % 4 )) -eq 0 ] && [ $(( ${YEAR} % 100 )) -ne 0 ]
then
echo 'Leap'
else
echo 'No Leap';
fi
```

For the real geek, during a century, an integer binary number can be ANDed with 0x03, and the result is represented by the modulus of 4. If 0, the leap year.
—Jacques Amar

Dave Taylor replies: *There are lots of ways to calculate it, for sure. I like my latest: just type* `date -d 12/31/YEAR +%j`*, and see if it's 365 or 366.*

Counting Days

Dave Taylor's Work the Shell column is one of my favorites in *Linux Journal*. However, while reading the August 2014 issue, I thought "don't re-invent the wheel." In astronomy, getting the time interval (in days and fractions thereof) between any dates is a *daily bread and butter* issue. The solution is the Julian Date, a consecutive count of days from centuries ago into the future. It is a relatively simple algorithm, although not intuitive. See, for example, **http://aa.usno.navy.mil/faq/docs/JD_Formula.php**.

Yes, the algorithm is originally coded up in FORTRAN (what else was there in the science world some decades ago), and we still use it. A conversion to a shell script or any other programming language should be easy.
—Norbert Zacharias

Dave Taylor responds: *Holy cow, that's awesome stuff, Norbert! I haven't dabbled in FORTRAN since the job I had while I was an undergrad at UCSD back in the early 1980s. As it happens, I really enjoyed FORTRAN and enjoyed reading the code too.*

How that formula works, however, is a bit puzzling to me, as there are some rather mysterious constants that don't seem like they should work. Still, it'd be easy enough to recode as a shell script—in fact, the FORTRAN code is almost workable as is with just a few tweaks. My only comment: the lack of error checking makes me get a bit anxious with such complicated formulae in the mix.

Thanks for pointing that out. Good stuff!

Norbert responds: Thanks for responding to my earlier e-mail. Those constants are determined to "make it work", accounting for the fact that different months have a different number of days (even going over centuries) and so on. It is hard to understand the algorithm; however, the result is a fast, accurate algorithm. Testing was done on sample data, and after verification, no further tests are needed as part of the algorithm. Of course, one needs to be very careful when cutting/pasting lines in order not to screw it up.

PS. We still code in FORTRAN partly because we inherited a large body of code in our area of research and partly because the code is really easy to write and efficient—it pays off when dealing with billions of star positions. It even still can be done on a modern laptop!

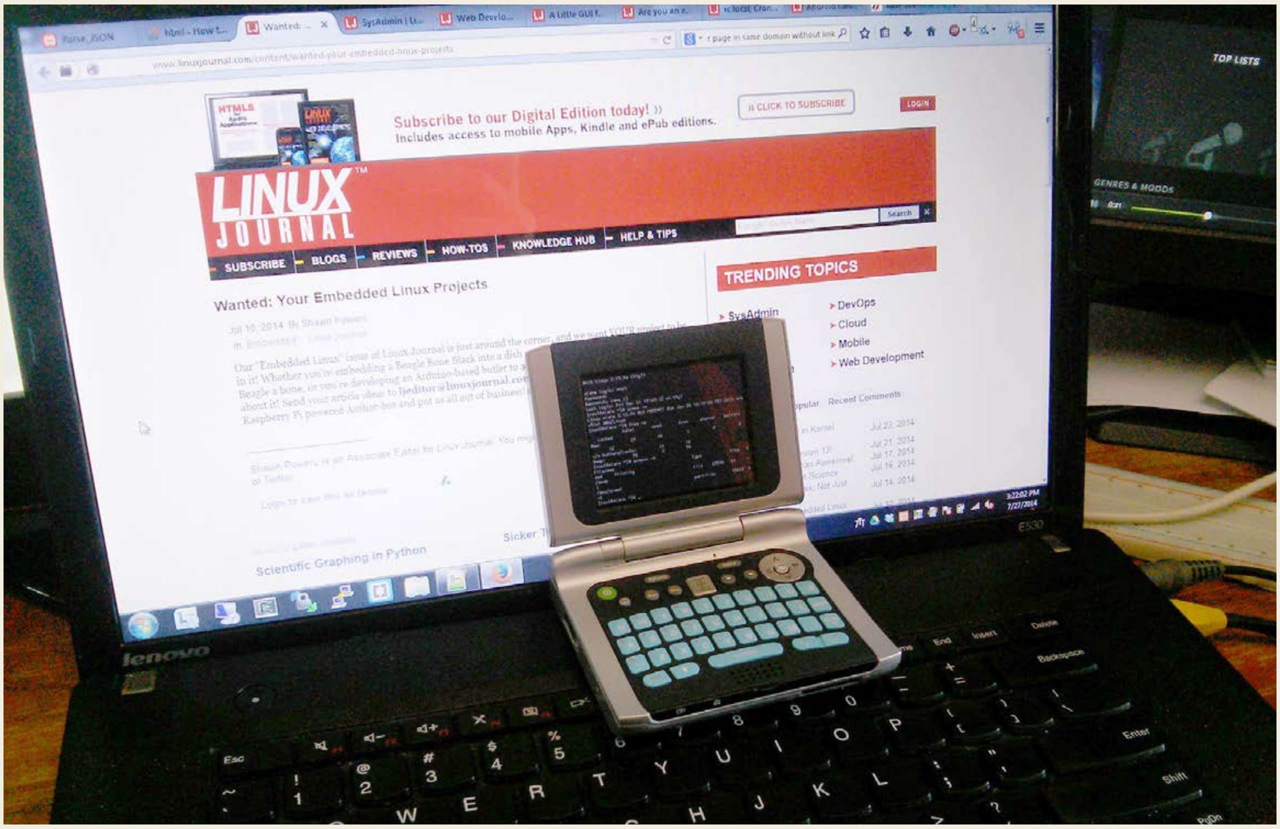
Suggestion

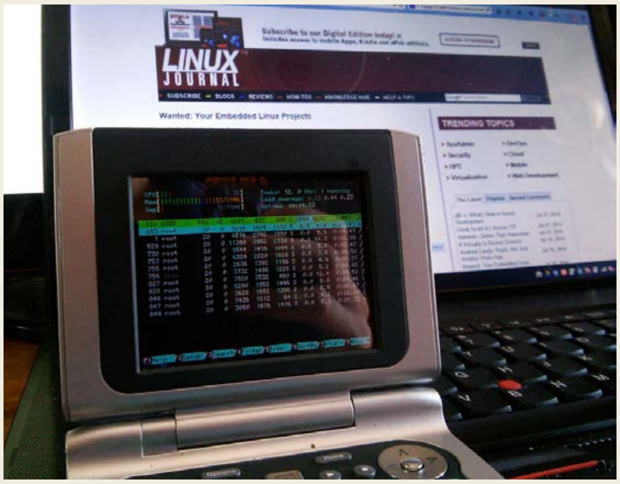
I am working for a telecom vendor on OSS products. Lately, I've been working on NFV-related meetings and workgroups. There is a lot going with NFV all around telco operators, and almost all will run

PHOTO OF THE MONTH

I'm writing to share this magnificent Linux laptop, which I believe is one of the smallest around. It's a Zipit Z2, and I'm a big fan of it. Thanks to Johannes from Wejp.k.vu, I'm able to host my Web site on this beauty.

—Matias





on Linux/KVM/OpenStack. There is and will be a lot of work done and money spent on this issue. Operators are pushing vendors for open source. There is OpenDaylight as well for SDN. The vendors are big ones, such as Ericsson, Huawei, ALU, Oracle, Cisco, Nokia, Microsoft and HP. Some other vendors are AT&T, Orange and TIM. Those are the biggest players around.

You are *the* Linux magazine, yet I see almost nothing on these issues. This disappoints me. Personally, I would like to see not only NASA's open-source tools, but also such big enterprise issues on open source and Linux. Linux is a serious thing that runs the Internet and many serious applications on the enterprise level. I would like to see these improvements in your magazine.
—Umit Kaan Sonal

We look for content that interests our users, along with new technology we think deserves to be noted. Getting feedback like yours is the best way for us to know what readers want, so we'll keep our eyes open. Thanks!—Shawn Powers

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.

diff -u
WHAT'S NEW IN KERNEL DEVELOPMENT

Kernel configuration has become more and more complex through the years with the proliferation of new drivers, new hardware and specific behaviors that might be needed for particular uses. It has reached about 3,000 **config options**, and that number will only increase.

Jean Delvare recently pointed out that a lot of those config options were relevant only to particular hardware, and yet the config system presented them to users who didn't have that hardware. This seemed like a bug to him, and he suggested that maintainers begin requiring proper hardware dependencies for all config options.

He acknowledged this would be a big task—especially for existing drivers. But creating the requirement for new drivers would at least get the ball rolling, and older drivers could follow along more gradually.

Josh Boyer agreed with all this. From his work on the **Fedora** project, he had to deal with the config system intensively, and he found it difficult. He said, "I've gotten to the point where I can somewhat guess based on the driver name which arch it's for (lately the majority are for ARM), but that isn't really a great way to handle things."

There was some resistance to the idea. **Greg Kroah-Hartman**, in particular, suggested that there were existing alternatives. For example, he said users simply could compile everything as modules. Then, they'd be loaded into the system only as needed.

But, neither Jean nor Josh liked that suggestion. Jean said that in the old days it was fine to build everything as a module, but nowadays there were just too many modules, and that "Saying 'm' to everything increases build times beyond reason. You also hit build failures you shouldn't have to care about, depmod takes forever, updates are slow as hell. This is the problem I am trying to solve."

Greg didn't see how build times could be a problem. Building the kernel on his laptop, he said, took about 20 minutes—with all 3,000 modules compiled in. If that wasn't good enough, he suggested upgrading the hardware to get a faster build time.

But, Jean said this wasn't a practical

solution for some cases. He said, "We have 34 kernel flavors for openSUSE 13.1, for example. And, every commit potentially triggers a rebuild of the whole set, to catch regressions as fast as possible. So every module we build, for no good reason, gets built a hundred times each day." He added that it would cost a lot of money to upgrade the hardware underneath that build system.

Greg said he understood the issue, but that fixing the config system was just a hard problem to solve. It boiled down to enforcing better habits on everyone producing patches. He said, "If you see new drivers show up that you don't know where they work on, ask the developers and make up patches." He added, "Perhaps a few developers could be auditing the new **Kconfig** items of every kernel around -rc3 time frame to ensure that they don't do stuff like this."

Jean said that -rc3 would be too late, because "all kernel developers and distributions have already moved to the new kernel so they have already answered the n/m/y question for all new entries." He added, "It's the reviewer's job to refuse new drivers with bad Kconfig descriptions in the first place. This must happen as early as possible in the chain."

No clear decision came out of the discussion, but it does seem as though there's a vast mountain of configuration options that are becoming more and more difficult to deal with. Eventually, I think some form of clean hardware dependencies will end up being implemented, along the lines of Jean's suggestion.

With all the new devices coming out on the market, there's a big desire to get Linux running properly on all of them. Things like **Intel's Quark** system use only a few MB of RAM and have other tight hardware requirements. Shrinking Linux down to the right size poses a challenge.

Andi Kleen recently pointed out, "One problem on these small systems is the size of the network stack. Currently enabling IPv4 costs about 400k." He wanted to give users the option to prune down the Linux networking stack to only the bare essentials and get it down to 100K per application—competitive with **Adam Dunkel's LwIP** (Lightweight IP) project.

Andi posted some patches to create three available options for the networking stack: a full system with all current features, a partial system that supported regular users but not servers or a minimal networking system for the special userland on deeply embedded systems. The minimal system, he said, would "remove rtnetlink (ioctl only), remove ethtool, raw sockets".

This seemed extreme to **Richard Weinberger**, who said that on such a minimal system, even the ps command wouldn't work. **Tom Zanussi** on the other hand, said that the **microYocto** Linux distribution ran okay with Andi's patches and had decent workarounds to keep ps working properly. But, he added that microYocto was "very much a work-in-progress with a lot of rough edges, but it is a fully functional system on real hardware".

Alexei Starovoitov felt that trying to support such a minimal system would only create config options that led to a hacky, work-in-progress, rough-edged system. He said that if the goal was to create a half-functional system that was just very very small, the same thing could be accomplished with linker hacks. Simply "compile kernel as-is. Most functions have a stub for mcount() already. Use it to track whether the kernel function was called or not. Collect this data in userspace (as perf already does), add a few more functions that had a 'notrace' attribute on them, and feed this into a special linker that unpacks existing vmlinux, throws away cold functions, relocates the rest and here you have tiny vmlinux without recompilation."

But, Andi pointed out that for networking code, this wouldn't really work. He objected, "How would you know you exercised all the corner cases in the TCP stack? And you wouldn't want a remotely exploitable system because some important error handler is missing."

The discussion ended inconclusively. But, it does seem as though real patches, and not linker hacks, will be used to support all new hardware—even the tiny hardware that's been coming out lately.—**ZACK BROWN**

They Said It

Do not hire a man who does your work for money, but him who does it for love of it.
—Henry David Thoreau

When in doubt, tell the truth.
—Mark Twain

Experience is a good teacher, but she sends in terrific bills.
—Minna Thomas Antrim

Just the knowledge that a good book is awaiting one at the end of a long day makes that day happier.
—Kathleen Norris

Always be a little kinder than necessary.
—James M. Barrie

EdgeRouter Lite

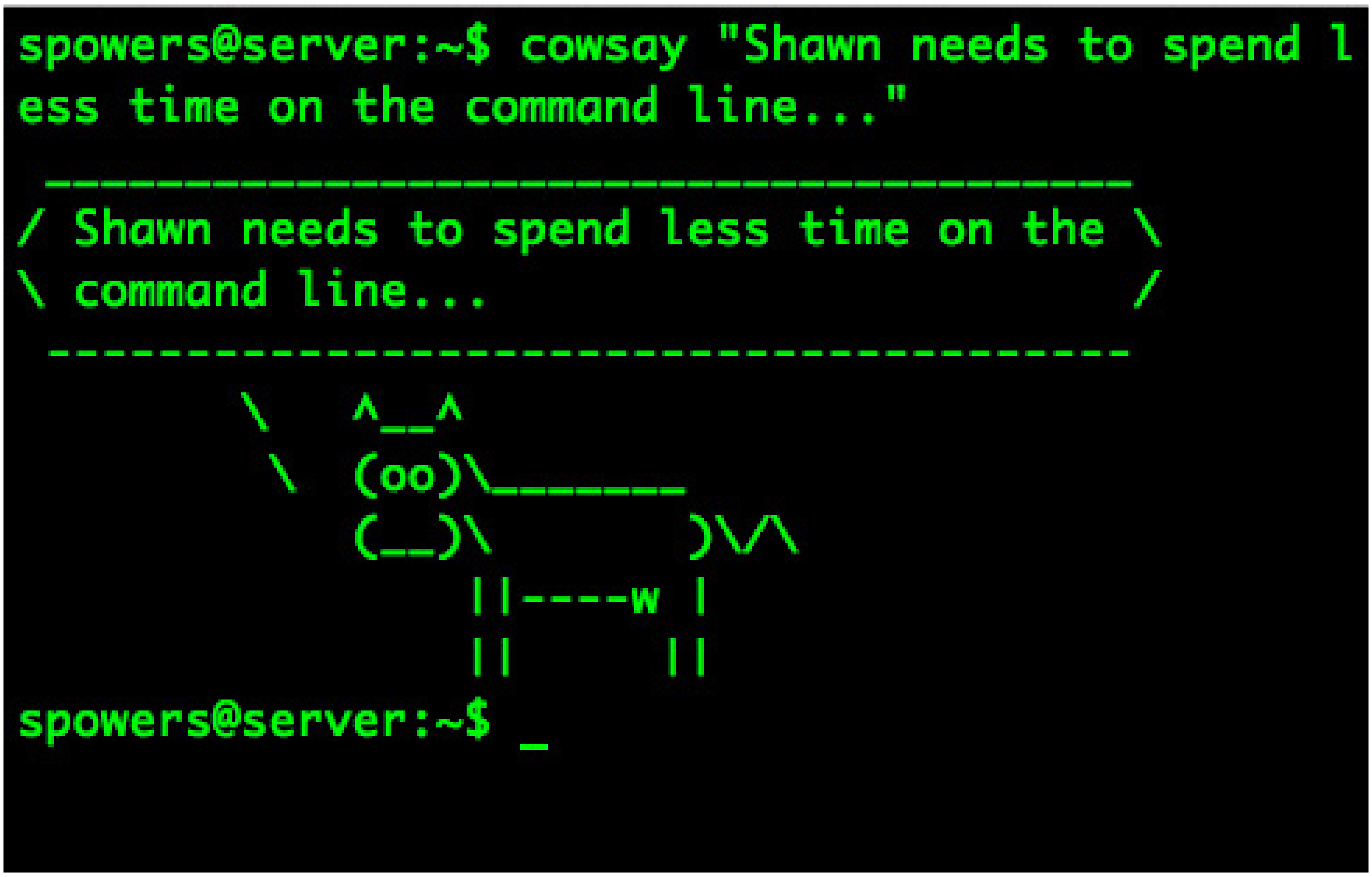
In the September 2014 issue, I mentioned my new router, and I got a lot of e-mail messages asking about how well it works. I can say without hesitation it's the nicest router I've ever owned. And, it was less than $100!

The EdgeRouter Lite is a business-class router based on the open-source Vyatta system. It has been forked, and as it matures, it will become less and less like the original Vyatta code, but for the present, it works much the same. I purchased the EdgeRouter because my old ATOM-based ClearOS router/firewall couldn't keep up with the traffic from my home network. My favorite features include:

- Three GigE ports, each routable separately.
- A claimed one million packets per second throughput.
- Wire-speed routing.
- Advanced configuration possible.
- Price!

I'll admit, setting up the EdgeRouter Lite was a pain in the rear end. The basics can be done via the Web interface, but if you want any QOS for your connection, it will be a learning experience trying to figure out the Vyatta command and code (and concepts!) for doing so. It took me three or four hours to get the setup that I'm happy with, and since then, I haven't touched it—at all. It works so fast, I never notice it, and it's been rock-solid since day one. If you're looking for an advanced router, but don't want to break the bank, the EdgeRouter Lite might be exactly what you're need. Be warned, however, its setup is not for the faint of heart.—**SHAWN POWERS**

The Cow Says, Have Fun!



```
spowers@server:~$ cowsay "Shawn needs to spend l
ess time on the command line..."
 ________________________________________
/ Shawn needs to spend less time on the \
\ command line...                        /
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
spowers@server:~$ _
```

Sometimes, when the clock hits 3:00am, and you've been in the server room since 9 o'clock the previous day, you start to get a little batty. That's the only explanation I have for programs like cowsay in Linux. Still, I'm glad they're there, because life wouldn't be nearly as fun without them. Here's a quick list of silly Linux programs off the top of my head. I'd love to hear about more, so please, send me your crazy Linux Easter eggs, and I'll follow up on the Web site showing off the best ones.

- cowsay: install this program, and the cow will say whatever you ask it to. (Bonus: there's a GUI version of cowsay too, called xcowsay!)

- sl: this is a program I like to install, because it makes fun of you when you accidentally type `sl` instead of `ls`—a steam locomotive chugs across the screen. (It also shows up if you press caps lock, and type `LS`!)

- cmatrix: blue pill or red pill, this little program will suck you in to the Matrix either way! (Leaving *this* matrix just requires a Ctrl-C, thankfully.)
- libaa-bin: install this package, and then type `aafire` to stoke up the ASCII flame. Grab your digital marshmallows!
- Star Wars: open a terminal and type `telnet towel.blinkenlights.nl`, and grab some popcorn. Or maybe roast some of those digital marshmallows, because you can watch the entire *Star Wars* movie in a terminal window.

Most of these silly things have been around for years and years, but every so often, I learn of one I never knew about before. Send me your favorites, and I'll be sure you get credit for slacking off at work—er, I mean for discovering awesomeness! E-mail shawn@linuxjournal.com (put something like "FUN" in the the subject line so I know what it's about).—**SHAWN POWERS**

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.

What's Happening above Your Head?

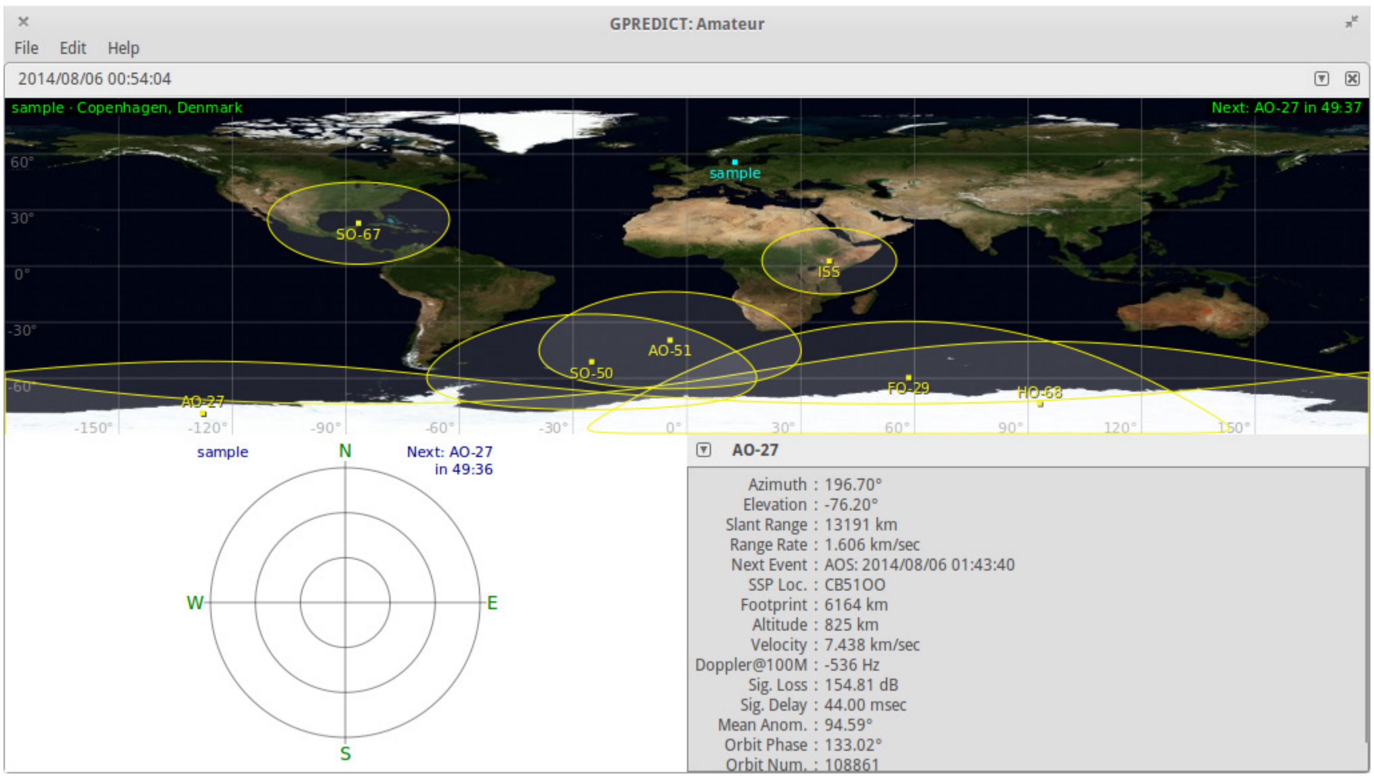
In the past, I've covered various astronomy packages that help you explore the universe of deep space. But, space starts a lot closer to home. It actually begins a few hundred miles above your head. There are lots of things in orbit right above you. In this article, I look at one of the tools available to help you track the satellites that are whizzing around the Earth: Gpredict (**http://gpredict.oz9aec.net/index.php**).

A package should be available in most Linux distributions. In Debian-based ones, you can install Gpredict with:

```
sudo apt-get install gpredict
```

This is usually a version or two behind the latest, so if you want to have the newest options, you always can download and build from source.

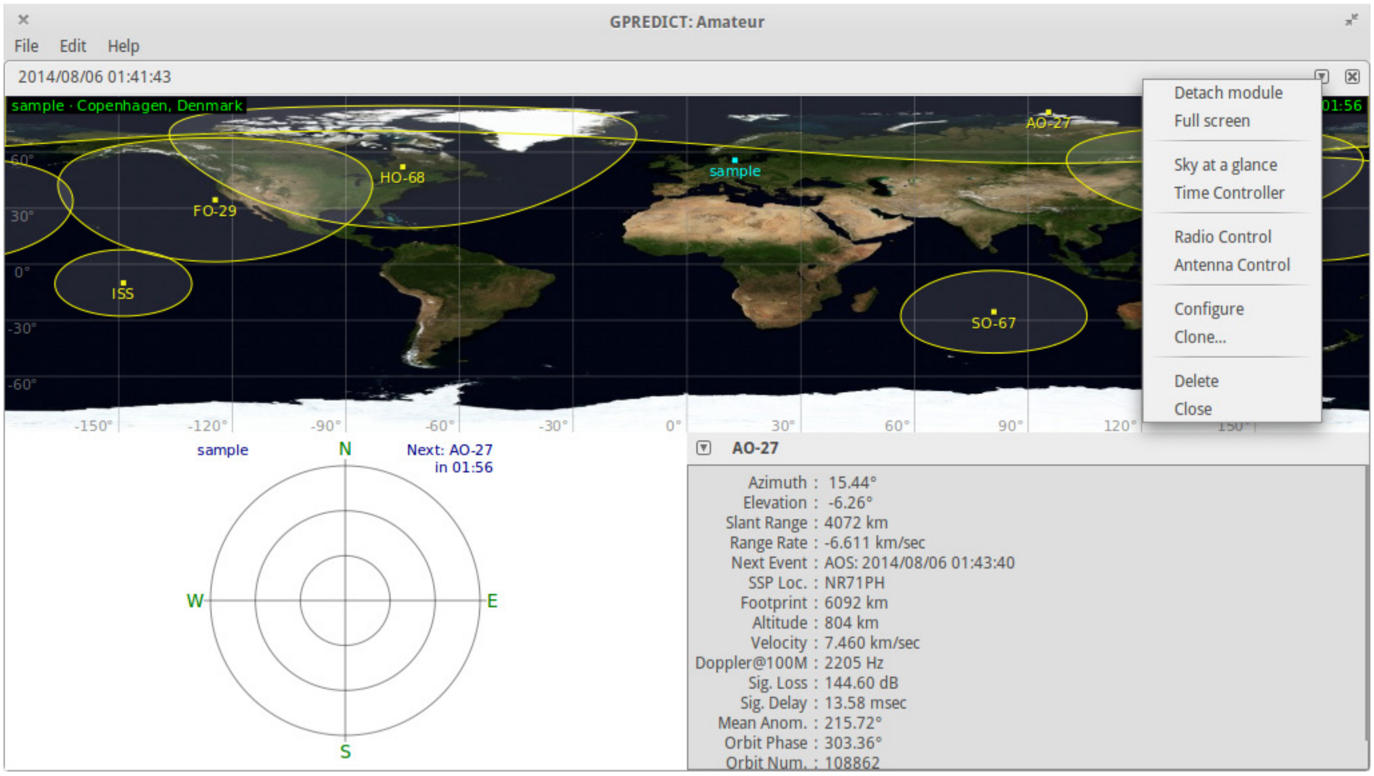
Once you have it installed, you can start it with the `gpredict`



```
2014/08/06 00:54:04
sample · Copenhagen, Denmark                Next: AO-27 in 49:37
```

```
     Azimuth : 196.70°
   Elevation : -76.20°
 Slant Range : 13191 km
  Range Rate : 1.606 km/sec
  Next Event : AOS: 2014/08/06 01:43:40
    SSP Loc. : CB51OO
   Footprint : 6164 km
    Altitude : 825 km
    Velocity : 7.438 km/sec
Doppler@100M : -536 Hz
   Sig. Loss : 154.81 dB
  Sig. Delay : 44.00 msec
 Mean Anom. : 94.59°
 Orbit Phase : 133.02°
  Orbit Num. : 108861
```

Figure 1. When Gpredict first starts, you get an initial module called Amateur.



```
      Azimuth : 15.44°
    Elevation : -6.26°
  Slant Range : 4072 km
   Range Rate : -6.611 km/sec
   Next Event : AOS: 2014/08/06 01:43:40
     SSP Loc. : NR71PH
    Footprint : 6092 km
     Altitude : 804 km
     Velocity : 7.460 km/sec
Doppler@100M : 2205 Hz
    Sig. Loss : 144.60 dB
   Sig. Delay : 13.58 msec
  Mean Anom. : 215.72°
  Orbit Phase : 303.36°
   Orbit Num. : 108862
```

Figure 2. Some views have drop-down lists of options.

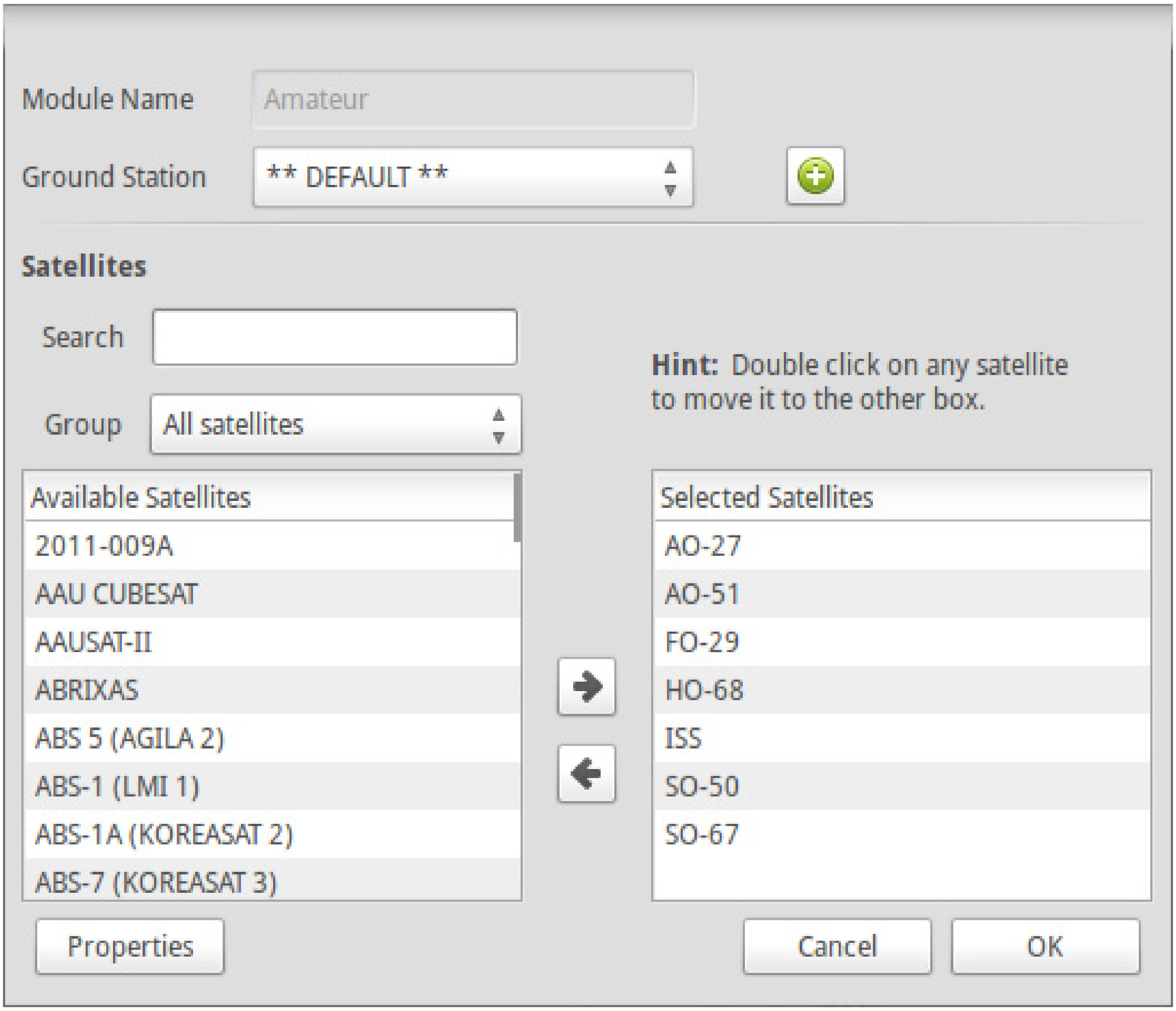
command. When it opens, you should see the main window, with a sample layout given by the module named "Amateur" (Figure 1).

In the rest of this article, I take a look at all the various possible layouts and show just how much information is available to you.

The core concept in Gpredict is that of the module. You can think of modules as documents in a word processor. They are containers you can use to hold a number of other layout objects that display satellite information in a number of different ways.

When you first start Gpredict, you get the default Amateur module, which contains a map view, a polar view and a single satellite view. For some of these views, you may notice a small down-arrow in one of the top corners. Clicking this icon gives you an appropriate drop-down list of options. For example, clicking the down-arrow in the map view gives you a list of items, such as detaching the module or configuring it (Figure 2).

The map view offers a map of the Earth, with a series of satellites and their footprints on the Earth. When you hover over one of the satellites, you will see an information box with the satellite's detailed location. The single satellite view provides


Module Name: Amateur

Ground Station: ** DEFAULT **

Satellites

Search:

Group: All satellites

Hint: Double click on any satellite to move it to the other box.

Available Satellites
2011-009A
AAU CUBESAT
AAUSAT-II
ABRIXAS
ABS 5 (AGILA 2)
ABS-1 (LMI 1)
ABS-1A (KOREASAT 2)
ABS-7 (KOREASAT 3)

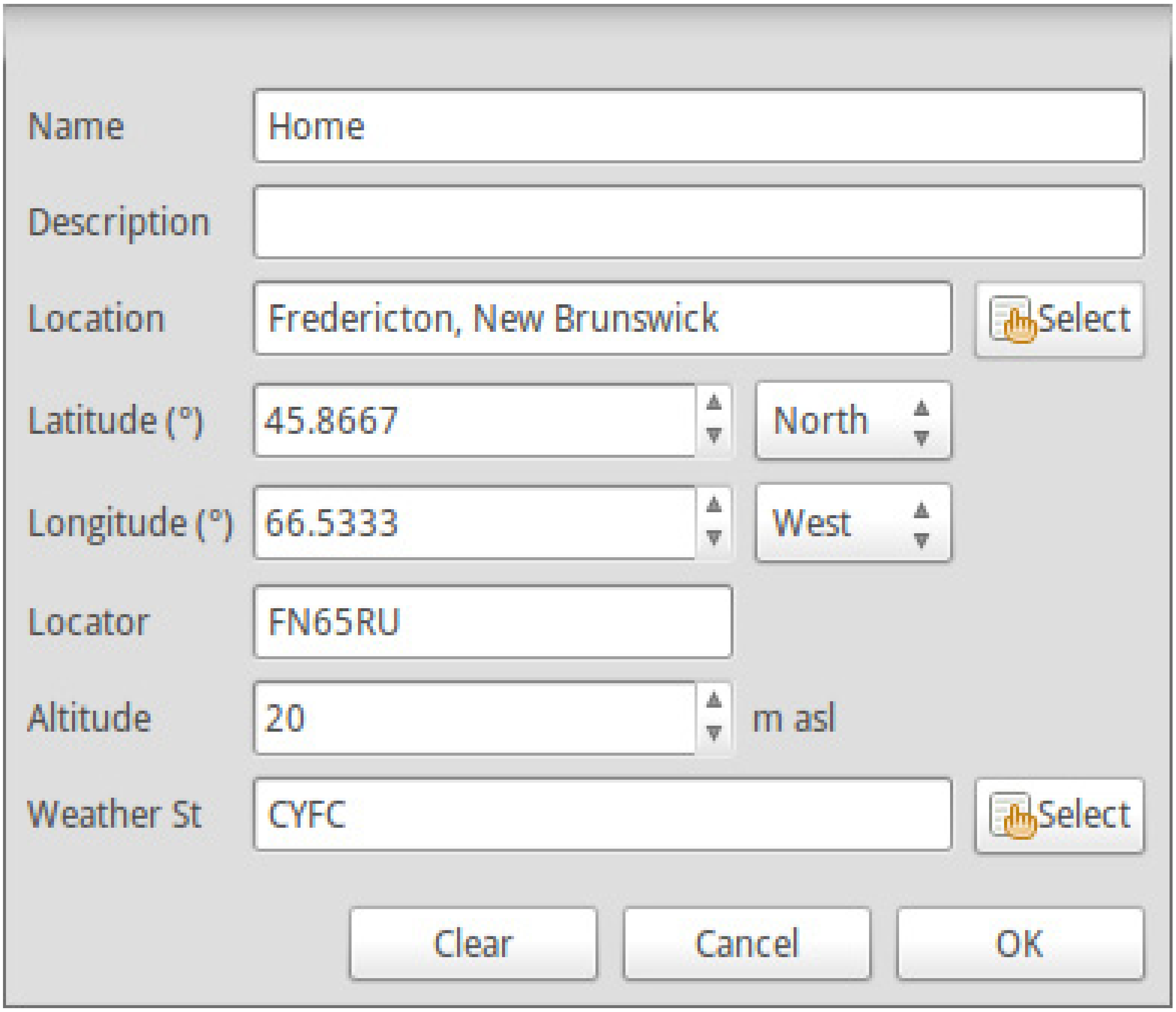
Selected Satellites
AO-27
AO-51
FO-29
HO-68
ISS
SO-50
SO-67

Figure 3. You can set the configuration details for the map view here.

even more detail about one specific satellite. You can select which satellite is being displayed by clicking on the down-arrow in the view. The polar view provides an overhead look, located at the ground station.

In the Amateur module, you get one ground station called "sample". You can add more ground stations by clicking the down-arrow and selecting configuration (Figure 3).

You can add another ground station by clicking the plus sign. This will pop up a details window where you can enter a name and the location data for your ground station (Figure 4). For the location, you can enter the latitude and longitude



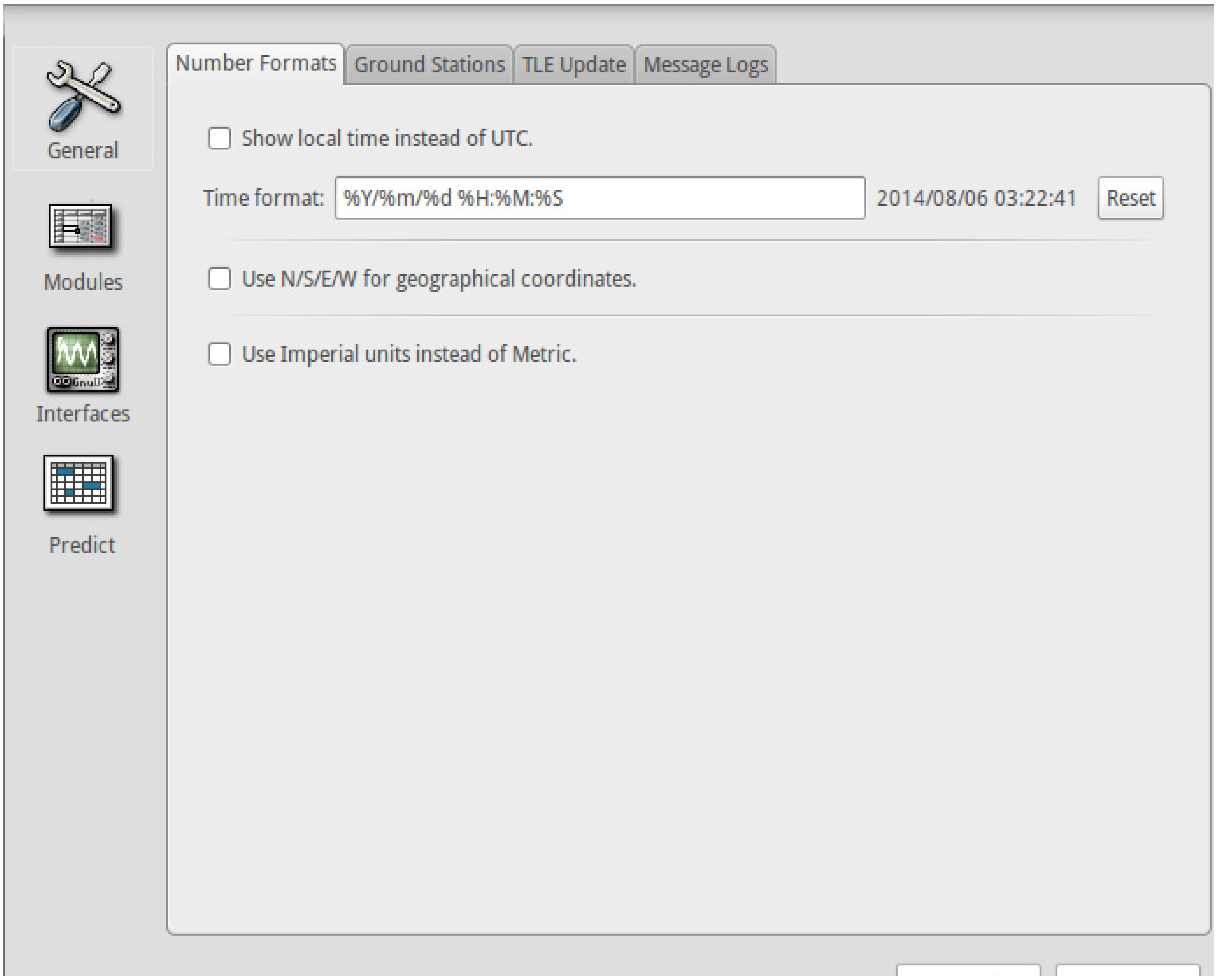
Name	Home
Description	
Location	Fredericton, New Brunswick
Latitude (°)	45.8667 North
Longitude (°)	66.5333 West
Locator	FN65RU
Altitude	20 m asl
Weather St	CYFC

Figure 4. You can add a ground station with details here.

manually, or if you live in a major city, you can select it from a global list of locations.

One other item you will notice when you have the configuration window open is that you can select which satellites are displayed. This list is rather large, but there is nothing stopping you from adding all of them to your module. It might make the display a tad crowded, but it still should work.

I should take a step back at this point and describe some other configuration options available. The first option to look at is the menu item Edit→Update TLE. This option lets you update the Keplerian elements for the satellites. You can



☐ Show local time instead of UTC.

Time format: %Y/%m/%d %H:%M:%S 2014/08/06 03:22:41

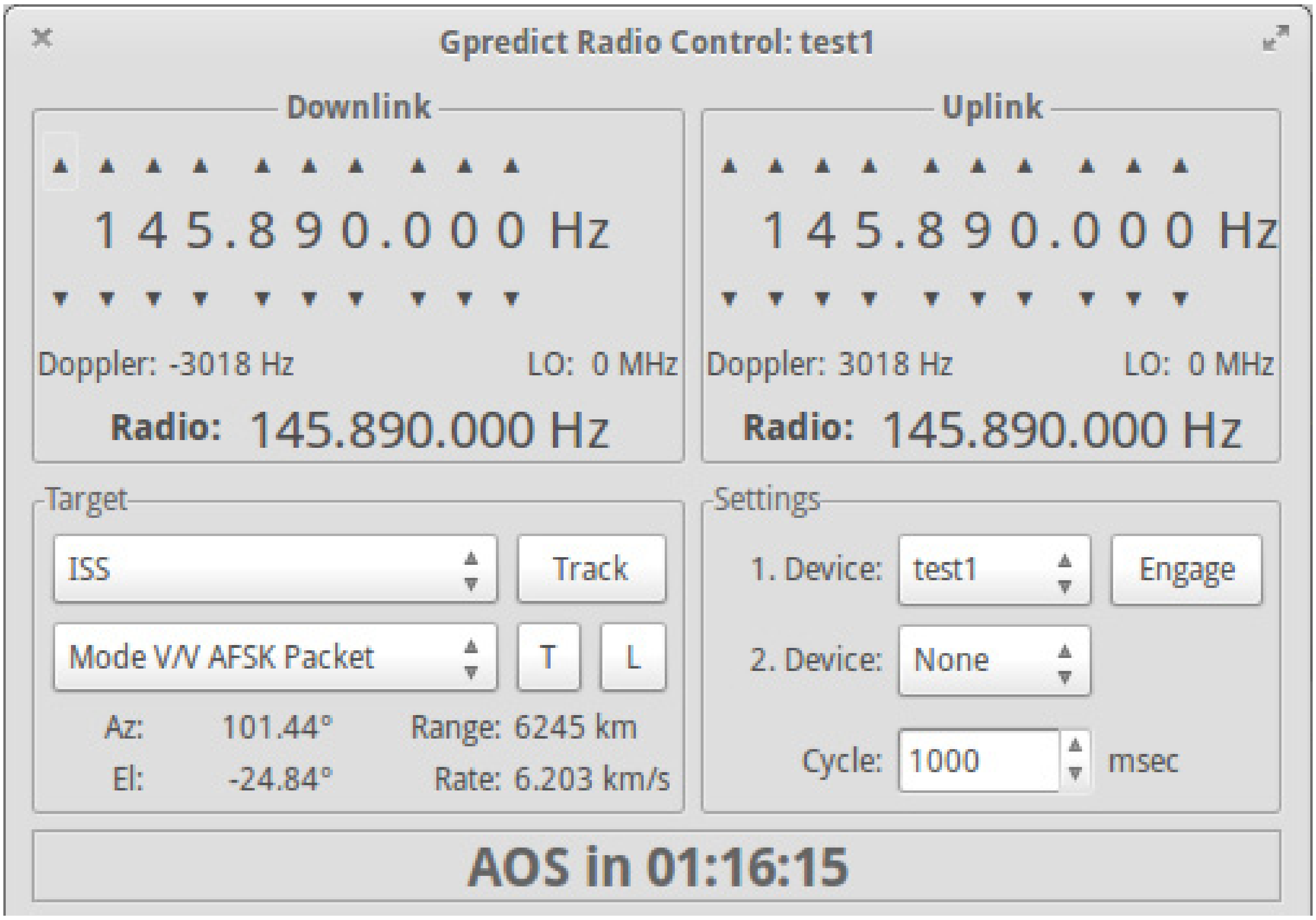
☐ Use N/S/E/W for geographical coordinates.

☐ Use Imperial units instead of Metric.

Figure 5. The preferences window lets you change all sorts of options.

update them either from the network or from a local file. The default configuration is set up to remind you when the TLE data is likely out of date. You then can go ahead and update this data. For an update over the network, the default configuration is to download NORAD data from **http://www.celestrak.com**. For a tutorial on the format for TLE data, visit **http://www.amsat.org/amsat-new/tools/keps_tutorial.php**.

All the other configuration options are available under the menu item Edit→Preferences (Figure 5). Here, you can change options like the number formats used or the geographical coordinates. There also is a tab for ground stations, where you can edit or add ground stations.



Downlink: 145.890.000 Hz
Doppler: -3018 Hz LO: 0 MHz
Radio: 145.890.000 Hz

Uplink: 145.890.000 Hz
Doppler: 3018 Hz LO: 0 MHz
Radio: 145.890.000 Hz

Target: ISS
Mode V/V AFSK Packet
Az: 101.44° Range: 6245 km
El: -24.84° Rate: 6.203 km/s

1. Device: test1
2. Device: None
Cycle: 1000 msec

AOS in 01:16:15

Figure 6. You can control radios and antenna rotators that are defined in Gpredict.

The modules section lets you change configuration options used in the display of the Gpredict modules. You can change things like the refresh rate for the displays or what map to use as the background for the map view. You also can select what type of layout you want for a particular module. When you select a new layout, you will see a preview of what it will look like in the preferences window. There are nine different pre-generated layouts available, or you can create a custom layout. When you select the custom layout, you define what it will look like by entering the layout code. See the user manual for details on how to define a code to create the layout you want.

Gpredict also has the ability to control radios and rotators. The key to this is the hamlib library. By using this library, Gpredict can handle Doppler tuning of radios and tracking of antenna rotators. When you want to connect hardware to your computer, you should verify that hamlib can talk to it successfully. Once you have made sure everything is working correctly, you can set up Gpredict to talk to your hardware. This is handled in the interfaces section of the preferences window. There are two tabs, one for radios and one for rotators. Since hamlib communicates over network protocols, the radio or rotator doesn't even need to be connected to the same machine. You can define one of these pieces of hardware with a hostname, a port and the communication details. Once you have the hardware configured, you can control it by pulling up the radio control window, which you access by clicking the down-arrow in the map view and selecting Radio Control (Figure 6). You can see the details of the downlink and uplink, as well as information about targets.

Now that you know how to get satellite information for what is moving above your head, you should be able to go outside and do some actual observations and see all of the man-made objects travelling around. It can be inspiring to see how much we already do in space, and how much more we could be doing.—**JOEY BERNARD**

Non-Linux FOSS: Remember Burning ISOs?

Image courtesy of http://www.visualpharm.com.

I was chatting with a Windows-using friend recently, and he wanted to try Linux on one of his older computers. I always like those sorts of conversations, and so I kept chatting, walking him through setting up Unetbootin to create a USB installer and so on and so on. Unfortunately, he wasn't able to get the USB drive to boot. Since we were half a country apart, I couldn't troubleshoot locally, so we moved on to burning a CD/DVD.

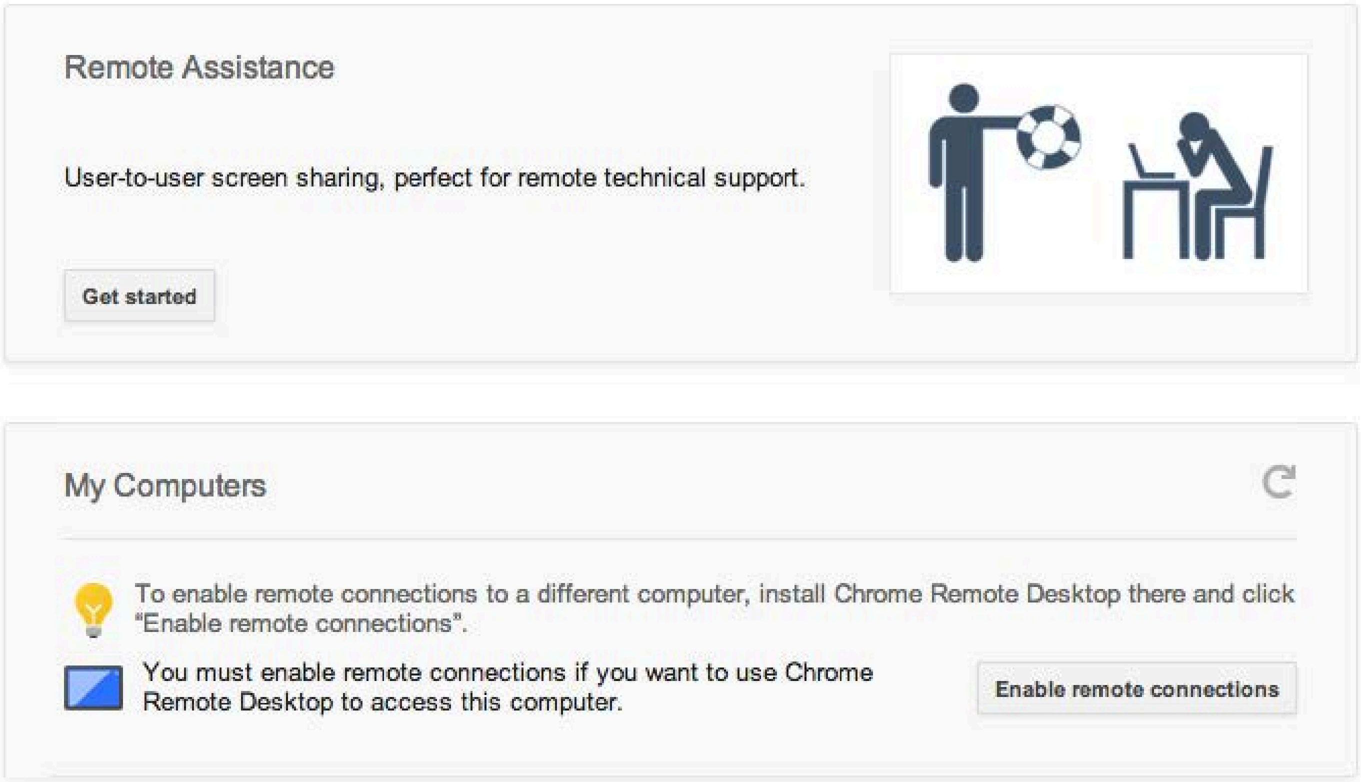
My first instinct was to have him download the incredible InfraRecorder. Unfortunately, there seems to be a malware-infected version of InfraRecorder going around, and of course, that's the download link he found. So, be sure to send folks directly to **http://infrarecorder.org** to download it.

Alternatively, I'm a big fan of the free-but-not-free program ImgBurn as well. It's not open source, but it is freeware, and it has a very simple interface. Either way you go, be sure to warn potential Linux converts about the malware masquerading as open-source software. Remember to send people directly to the Web site rather than having them "google" for it. The open-source InfraRecorder is at **http://infrarecorder.org**, and the freeware ImgBurn is at **http://www.imgburn.com**. Once they switch to Linux, everything they need will be an `apt-get` or `yum` away!—**SHAWN POWERS**

Android Candy: Goodbye RDP, Hello Chrome Remote Desktop!

Controlling a remote computer is something you're all familiar with. Whether that means RDP to your corporate Windows Server (we don't judge), Apple Remote Desktop (which is really VNC) to your OS X machine or VNC/X11/etc. into your GUI Linux machine, it's always a pain in the rear.

The folks at Google are trying to make it a little easier with Chrome Remote Desktop. It's a combination of Chrome browser app and native binary that runs on your



Remote Assistance

User-to-user screen sharing, perfect for remote technical support.

My Computers

To enable remote connections to a different computer, install Chrome Remote Desktop there and click "Enable remote connections".

You must enable remote connections if you want to use Chrome Remote Desktop to access this computer.

client machines. Once the dæmon is installed, you can access the computer remotely from anywhere, including a really cool Android app. The combination of available platforms is pretty impressive too:

Server platform (what can be controlled):

- Windows
- OS X
- Linux (Beta, Ubuntu/Debian for now)

Client platforms (what can be used to control the systems above):

- Windows
- OS X
- Linux
- Android
- iOS (coming soon, supposedly)

Permissions on the dæmon can be customized for controlling your own computer remotely (no local permission required) or for allowing other people in to assist you. The latter is preferred to avoid anyone barging in on your work session. The dæmon is available for Windows and OS X, and recently they released a beta version of the dæmon for Ubuntu/Debian Linux! Thanks to its wonderful cross-platform approach and smooth functionality in our testing, Chrome Remote Desktop earns this month's Editors' Choice award! Check it out today at **https://chrome.google.com/remotedesktop**.

—SHAWN POWERS

2014 Book Roundup

REUVEN M. LERNER

Reuven gives his annual rundown of great books on programming languages, databases and analytics, freelancing and more.

As I write these words, the end of summer is approaching, and with it, so is the time for my annual book roundup. As in past years, in this article, I describe books that were new to me during the past 12 months, which means that I might well mention some new ones or ignore others that simply didn't come to my attention. My interests (professionally, and for this column) generally are relevant to people involved in Web development using open-source technologies, but I admit to veering into other subjects occasionally. And as I have done for the last few years of this annual roundup, I have expanded my list of books also to include blogs, podcasts and other resources that may be of interest to others in the Open Source community.

Programming Languages

Python continues to grow in popularity and for good reason—it's an elegant language that is both powerful and easy to learn. I have been teaching a growing number of Python courses during the past few years to a variety of audiences—aspiring Web developers, companies interested in moving their automation and testing systems into an open-source high-level language, and also people interested in learning basic programming techniques and ideas.

The best-known book about Python, *Programming Python* by veteran instructor Mark Lutz, recently was released in its 4th edition by O'Reilly (ISBN 9780596158101). The book has been expanded and improved in many ways, most importantly with additional information about Python 3.x. Truth be told, I have yet to encounter a company that is using Python 3; all of my consulting and training clients still use Python 2.7. However, there's no doubt that Python 3.x is the future, and Lutz has done us all a good service by providing such information.

This is not a book about learning Python, but rather it's meant to be a reference and in-depth description of how the language works. And at that, it succeeds. At the same time, I feel like the book is far too large (at more than 1,600 pages), 400 pages of which are spent describing TkInter, a GUI toolkit that is included with Python, is easy to use and is cross-platform. But, it's also extremely ugly and not particularly popular, in my experience. If you're a Python programmer though, you'll likely want to have this book on your shelf.

When I'm teaching Python programming classes, my students often ask me how they can boost the performance of their code. True, I think that Python, like many other high-level languages, stresses programmer efficiency over language efficiency. And of course, we know that worrying about performance too early in a project is asking for trouble. At the same time, once our program is working, we definitely want to increase its speed. *Python High-Performance Programming*, written by Gabriele Lanaro and published by Packt Press (ISBN 978-1-78328-845-8), aims to solve some of these issues. The book concentrates on a small number of areas that can be used to increase performance: benchmarking and profiling (to understand where the problem is); NumPy's arrays (which are faster, if more limited, than Python's built-in lists); Cython (which translates Python code into C); and finally, multiprocessing. The book is relatively short (at 108 pages), but it does provide some of the most common techniques for increasing program speed. At the same time, I think that some more basic techniques, such as not using += to iteratively build strings, would have added something to the book.

The hottest language continues to be JavaScript, if for no other reason than the very large number of people who have access to browsers, be it on their computers or their mobile devices. JavaScript's huge community has been pushing forward with a variety of techniques and libraries that make development easier. Moreover, JavaScript is now popular beyond the browser, in such places as node.js, a server-side framework that uses asynchronous JavaScript.

It is this first topic that *Async JavaScript*, written by Trevor Burnham and published by the Pragmatic Programmers (ISBN 978-1-937785-27-7), addresses. Part of the difficulty that people have when working with JavaScript is its asynchronous nature. You often set up a function to be executed later on, when a particular even happens, and it's frequently tough to remember just how that worked. This book introduces techniques that will help you build asynchronous programs that are easier to understand and debug.

Functional programming is a technique that was pushed aside for many years by object-oriented programming, but it's making a comeback. Even if you're not using a purely functional programming language, functional techniques are increasingly important and useful. *Functional JavaScript*, written by Michael Fogus and published by O'Reilly (ISBN 9781449360726), introduces a wide variety of functional ideas and techniques. I'd say that this book is good for two types of audiences: those who already know functional programming and want to see how these ideas can be applied in JavaScript, as well as the opposite, people who know JavaScript and are interested in learning functional programming.

If you are new to JavaScript, or if you have been using it only for Web development and haven't yet learned how it works as a more general-purpose programming language, you may well want to read *Eloquent JavaScript*. The book, written by Marijn Haverbeke and available on-line at **http://eloquentjavascript.net** (and soon to be published in paper form by No Starch Press), reviews the basics of JavaScript as a language, ignoring the Web-related aspects until halfway through the book. So you learn about JavaScript objects, higher-order functions and even regular expressions, with clear and interesting examples.

HTML5 is not quite a language, but it certainly represents a combination of APIs and tools—most specifically, HTML, JavaScript and CSS—that are the standard way to create modern Web applications. It's quite amazing to see the breadth and depth of HTML5, as well as the applications that people are creating with it. If you're reading this column, you're probably not the right audience for *Head First HTML5 Programming*, which aims to teach the basics of HTML, JavaScript and CSS to people who have little or no familiarity with those technologies. But if you know of someone who wants to understand how these things work, it's worth recommending it.

Another non-language is Git, which has become one of my favorite and most indispensable tools. It's hard to believe that I used to be content with CVS and SVN, given the advantages that Git brings to the table. I must admit that many years ago, a friend of mine said that I eventually would switch to a distributed version-control system, which would make life so much easier. He was right, although he was recommending a system that preceded Git known as Arch, which I don't believe is used any longer.

I frequently teach courses in the use of Git, and I've found that the problem for most users is not the commands, but rather the understanding of what Git does (and doesn't do) and how the various objects work together. In particular, an understanding of what a commit is, and how branches, tags, blobs and trees fit into this commit-centric view of the world, is the biggest obstacle to working with Git.

So, I was happy to see the *Git Pocket Guide*, written by Richard E. Silverman and published by O'Reilly (ISBN 9781449325862). Yes, this book will help you use Git better, and it will remind you of many of the commands. But more significant, the *Git Pocket Guide* introduces you to the objects Git uses and describes how they work together to provide a robust and efficient version-control system.

Databases and Analysis

The growth of the Web has led not only to a growing need for people who can create Web applications, but also for those who can store, retrieve and analyze the data generated by those applications. The term "big data" increasingly is applied to such programs, and a number of books have been written in the past few years that attempt to help newcomers to the field.

In some cases, you don't need to collect or analyze the data yourself. Rather, you can get by just using the APIs that various systems have provided. This is particularly the case with social networks, which collect enormous quantities of information about their members and the connections between them. *Mining the Social Web*, by Matthew A. Russell, was released in its second edition (ISBN 9781449367619), and it describes many ways in which you can access and analyze different social networks. I've played with such APIs in the past, so I wasn't new to their use in my applications, but this book described a number of uses I didn't think of, with lots of clear example code that illuminated the points.

For those who want to analyze large data sets more directly, *Doing Data Science*, by Cathy O'Neil and Rachel Schutt and published by O'Reilly (ISBN 9781449358655), is a dense, but interesting, introduction to the techniques that are collectively known as "Data Science". The authors give a whirlwind tour of the math and algorithms used in such analysis and also the types of conclusions you can draw from its use. Some of this book was a bit heavy for my needs and interests, but it gave me a much better sense of what the Data Science people are doing, and how it is already becoming useful. I should note that Cathy O'Neil, one of the co-authors of this book, is a participant in the new "Slate Money" podcast. If you want to hear financial analysis driven by big data, you might want to tune in to that weekly recording.

Freelancing and Business

I've been a freelance consultant since 1995, working with clients around the world, and I really enjoy my work. I even participate in a weekly podcast, "The Freelancers Show", in which panelists discuss different aspects of what it's like to be self-employed.

Two of my co-panelists from the show have come out with eBooks during the past year. Curtis McHale wrote *Don't Be an Idiot: Learn to Run a Viable Freelance Business* (**http://curtismchale.ca/products/run-viable-freelance-business**), which gives practical advice for starting and running a successful freelance business. Along the same lines, Eric Davis has a practical, day-by-day plan for you to start on the freelancing track, in his *30 Days to Become a Freelancer* (**http://theadmin.org/30-days-become-freelancer**).

Other Books

Of course, I don't read only technical books, although my friends and family might not quite believe that claim. I have read a number of excellent books in the past year that I think you all might enjoy.

Tim Harford is a well-known economics writer and has published numerous books on the subject. His most recent, *The Undercover Economist Strikes Back*, describes how macroeconomics and government policies work, and the various theories that economists have used to try to understand these policies and their effects. I have long had an interest in economics, but still very much enjoyed reading this book, which shed new light on the differences between various schools of economics.

Another favorite writer of mine, Tom Standage, came out with a new book this year: *Writing on the Wall: Social Media—The First 2,000 Years*. Standage's argument is that just about every type of social media and technology that we use nowadays has precedents in previous cultures. His funny and interesting book is sure to give you some perspective on Facebook and LinkedIn, among other things.

Another great book that came out this year is *How Not to Be Wrong: The Power of Mathematical Thinking*, by Jordan Ellenberg. Ellenberg is a mathematics professor, and he tries to show in this book how mathematical thinking can help you make useful decisions. He starts the book by pointing out that "mathematics" isn't what you learned in elementary and high school, and it succeeds in having very few equations in the book. Rather, his point is to show how it's a perspective on life that can help you make decisions or persuade others. To his credit, the author also includes a number of examples of where mathematics cannot really provide a clear-cut answer, such as (most surprisingly) in deciding election results.

As readers of this column know, I have traveled to China several times in the past few years to teach Ruby and Python programming classes. And, I've become something of an aficionado of China-related books written by reporters who were there. Evan Osnos, who was the *New Yorker*'s correspondent in China for several years, published a book this year titled *Age of Ambition: Chasing Fortune, Truth, and Faith in the New China* that tries to make sense of the political and social changes happening in China. Along the same lines, I'm enjoying Howard French's book, *China's Second Continent: How a Million Migrants Are Building a New Empire in Africa*, which describes the experiences and interactions of Chinese expatriates in various African countries.

Finally, as someone who recently finished a PhD, I can recommend a funny (if rather cynical) book: *Surviving Your Stupid, Stupid Decision to Go to Grad School*, by Adam Ruben. Perhaps it's easier to laugh at his jokes now that I've completed the degree, but I found it not only funny but also therapeutic to know that others had experienced similar problems and challenges to mine during their grad-school careers.

I hope these recommendations will provide you with many hours of insightful and interesting reading! I'm sure I have missed many good books that came out in the past year; I'm always open to hearing about them and welcome your suggestions.■

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.

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

DAVE TAYLOR

Picking Out the Nouns

Dreamer—a dream interpretation program, as a shell script... well, sort of.

A reader wrote a letter to me (oh happy day!), and although I'm still not entirely sure what she's trying to accomplish, it's an interesting puzzle to try to tackle anyway. Here's what she asked:

> I do not know how to code, but I have a project in mind that is something like *Mad Libs*, but is for dream interpretation. I would like for people to be able to type a dream, and then the computer program would pick out the nouns and ask the participants to freely associate anything that comes to mind if they were that object or person. Then, the computer would replace the typed responses back into the typed text for the surreal interpretation. Do you think this would be difficult to create?

Mad Libs for dreams? That's certainly a curious idea, particularly given how seemingly random and disconnected the elements of a dream often seem. Dreams have been seen as both visions from the gods and the playground of our subconscious and its need to resolve our daily experiences. And then there's Freud, who is pretty sure that if you aren't literally dreaming of cigars, it's because you're envious of people with cigars or because you're fixated on cigars but suppressing your interests.

OOohhhhkay then. No cigars, okay? And no Lewinsky jokes either.

What we need to accomplish this task is a script that parses input, identifies and creates a list of nouns, prompts users for their free-association synonyms for each of the nouns, then pushes out the original text again, replacing each original noun with a substitute as suggested by the user. To start, how do you identify nouns?

First, We'll Kill All the Nouns

I was going to grab the comprehensive dictionary from Princeton University's

Wordnet program, but closer examination reveals that it has more than 85,000 words and has all sorts of obscure alternative uses and so forth. The end result is that although it's comprehensive, it generates too many false hits. So instead, Desi Quintans has a simple word-only list you can grab for our purpose here: **http://www.desiquintans.com/downloads/nounlist.txt**.

It's in exactly the format needed too:

```
$ head nounlist.txt
aardvark
abyssinian
accelerator
accordion
account
accountant
acknowledgment
acoustic
acrylic
act
```

It seems like that would be the most difficult step, but in fact, it's surprisingly easy given the almost infinite data store of the Internet.

Identifying Nouns in Prose

The next step is rather easy: given some prose, break it down into individual words, then test each word to identify which are nouns. This is really the bulk of the program, now that we have a noun list:

```
for word in $( sed 's/[[:punct:]]//g' $dream |
➥tr '[A-Z]' '[a-z]' | tr ' ' '\n')
do
  # is the word a noun? Let's look!
  if [ ! -z "$(grep -E "^${word}$" $nounlist)" ] ; then
    nouns="$nouns $word"
  fi
done
```

The for loop is a bit complicated, but it's removing all punctuation from the input, translating uppercase to lowercase, and then converting each space into a carriage return. The result can be shown most easily by example. Let's say that we had this as input:

```
I've never seen a blue chipmunk!
```

Running it through the `sed | tr | tr` filter produces this:

```
ive
never
seen
a
blue
chipmunk
```

That's easy enough, and now that we can separate out each word from the input, it's easy to search the noun list to see if any match. Again, it's a bit complex, because we need to ensure that we aren't getting embedded matches (for example, matching the noun "acoustic" for the slang word "stic").

That's done by *rooting* the search as a regular expression: ^ is at the beginning of the line, and $ is the end of the line—hence the regular expression `^${word}$` where the use of the optional `{}` notation just delimits exactly what the variable name is to the shell.

With some debugging code included, here's our first draft of this entire script:

```
#!/bin/sh

# dreamer - script to help interpret dreams. does this
#    by asking users to describe their most recent
#    dream, then prompts them to free associate
#    words for each of the nouns in their original description.

nounlist="nounlist.txt"
dream="/tmp/dreamer.$$"

input=""; nouns=""

trap "/bin/rm -f $dream" 0      # no tempfile left behind

echo "Welcome to Dreamer. To start, please describe in a
➥few sentences the dream"
echo "you'd like to explore. End with "DONE" in all caps
➥on its own line."
until [ "$input" = "DONE" -o "$input" = "done" ]
do
  echo "$input" >> $dream
  read input    # let's read another line from the user...
done

echo ""
echo "Okay. To confirm, your dream was about:"

cat $dream

echo "=============="

for word in $( sed 's/[[:punct:]]//g' $dream | tr '[A-Z]'
➥'[a-z]' | tr ' ' '\n')
do
  # is the word a noun? Let's look!
  if [ ! -z "$(grep -E "^${word}$" $nounlist)" ] ; then
    nouns="$nouns $word"
  fi
done

echo "Hmm.... okay. I have identified the following
➥words as nouns:"
echo "$nouns"

echo "Are you ready to do some free association? Let's begin..."

for word in $nouns
do
  echo "What comes to mind when I say $word?"
done

exit 0
```

As is immediately obvious, the free association section at the end and the subsequent reassembly of the prose with the new free association words or phrases is still to come.

It's really broken into simple functional blocks: first prompting users to share their dream, then breaking down the prose into individual words and comparing them to the noun list and finally (albeit not yet in its final form), prompting for the free association of each identified noun.

Let's run it to see what I mean:

```
$ sh dreamer.sh
Welcome to Dreamer. To start, please describe in a few
sentences the dream you'd like to explore. End with DONE
in all caps on its own line.
I was sitting in a tree house in the middle of an ancient
forest and an owl was staring at me. It asked "who?" and
I woke up in a cold sweat.
DONE

Okay. To confirm, your dream was about:

I was sitting in a tree house in the middle of an ancient
forest and an owl was staring at me. It asked "who?" and
I woke up in a cold sweat.
==============
Hmm.... okay. I have identified the following words as nouns:
 tree house middle forest owl cold
Are you ready to do some free association? Let's begin...
What comes to mind when I say tree?
What comes to mind when I say house?
What comes to mind when I say middle?
What comes to mind when I say forest?
What comes to mind when I say owl?
What comes to mind when I say cold?
```

As is immediately obvious, the free association section at the end and the subsequent reassembly of the prose with the new free association words or phrases is still to come.

But that's a project for next month. Meanwhile, keep a dream journal and soon you'll be ready to interpret it thanks to the Linux shell—or something like that!■

Dave Taylor has been hacking shell scripts for more than 30 years. Really. He's the author of the popular *Wicked Cool Shell Scripts* and 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.

KYLE RANKIN

Return of the Mac

Save yourself keystrokes by building a library of vim macros to automate mundane text-editing tasks.

In my July 2014 column, I talked about vim macro basics. In that article, I described how to record a custom macro, assign it to a key and then use it to make automated edits to a BIND zone. I also teased that I would cover more advanced uses of macros, like nested macros, in a future issue. I took a brief detour to cover a few different topics, but now I'm back on topic, and in this article, I discuss more complicated uses for macros.

I like using BIND zone files for macro examples, because it's the file I most often use macros in myself. Because multiple people often edit zone files, they may not all have the same formatting. Plus, the top of a zone file generally has a different multi-line format compared to the rest of the file. In my July 2014 article, I talked about how to add 50 sequential A records in a zone file using a single macro, but when I have to perform more complicated edits, or if I have to perform edits selectively in some files but not others, I've found it useful to record a few different simple macros under different keys, then record a new macro that just calls those other macros in a particular order. Among other things, this lets me change some of the shorter macros if I need to, without having to re-record everything.

For the first example, let's look at a more complete version of the BIND zone file I used last time:

```
;
; BIND data file for example.com
;
$TTL 15m
@       IN      SOA     example.com. root.example.com. (
                        2014081500
                        10800
                        1200
                        7200
                        7200 )
;
```

```
example.com.   IN      NS      ns1.example.com.
example.com.   IN      NS      ns2.example.com.
;
ns1      IN A  10.9.0.5
ns2      IN A  10.9.0.6
;
worker1  IN A  10.9.0.15
worker2  IN A  10.9.0.16
worker3  IN A  10.9.0.17
. . .
worker50 IN A  10.9.0.64
```

In this example, let's say I have 15 zone files for different zones, but I want to make the same set of edits to all of them. I want to change the TTL in the file to be 10 minutes, and I need to change the contact info for the domain from root@domain to dnsadmin@domain. I also need to increment the serial number (2014081500) in the zone file, and I need to change the name server IPs all to point to a new set of name servers at 10.1.0.250 and 10.1.0.251, and finally, I want to add 50 more workers to each zone file.

Although I imagine I could build all of this into a single big macro, for me, it makes sense to split up the steps into at least four macros that I already have pre-assigned letters to:

- Macro t will change the TTL.
- Macro s will change the contact information and increment the serial.
- Macro n will update the name server records.
- Macro w will add one more worker.

Because I'm going to chain these macros together, it's even more important than in the past that I make sure my cursor is anchored in a known location first. For the first macro, this means starting with `gg` to move to the very top of the file, while for the last macro, I will want to type `G` to move to the bottom of the file. At each phase, it's incredibly important that you test each macro, then undo the changes and confirm that your macros work exactly how you expect. Let's break down each macro.

For macro t, I first type `qt` to enter macro mode and assign the macro to the t key. Then I type `gg` to make sure I'm at the top of the file. Next I type `/TTL` <Enter> to move the cursor to the TTL line. Then I type `w` to move forward a word to the actual TTL value I want to change, and then I type `cw10m` to change the following word from 15m to 10m. Finally, I press Esc to exit insert mode, and `q` to exit the macro. The complete set of macro keystrokes then

Although this seems like a lot of text, it will save a ton of work when you have to repeat the steps on multiple files.

would be `qtgg/TTL` <Enter> `wcw10m` <Esc> `q`. Once I record the macro, I type `u` to undo my changes, and then test the macro by typing `@t`.

For macro s, I type `qs` to enter macro mode and assign the macro to the s key. Then I type `gg` again to anchor to the top of the file. Next I type `/SOA` <Enter> to move to the SOA line. Then I type `/root` <Enter> to move to the beginning of root.example.com, and then type `cwdnsadmin` <Esc> to change that word to dnsadmin and exit insert mode. Next I type `^j` to move the cursor to the beginning of the following line. Finally, I type `w` <Ctrl-a> to move forward to the serial number and increment it, and then `q` to exit macro mode. The complete set of macro keystrokes becomes `qsgg/SOA` <Enter> `/root` <Enter> `cwdnsadmin` <Esc> `^jw` <Ctrl-a> `q`. And again, I save the macro and use `u` to undo the change and replay it with `@s` to make sure it does what I expect.

For macro n, I type `qn` to enter macro mode and assign the macro to the n key. Then I type `gg` to anchor to the top of the file. Next I type `/^ns1` <Enter> to move to the line that configures the name servers. At this point, there are a few ways I could edit these lines. My preference is to type `/IN A` <Enter> `2w`, which will move my cursor to the beginning of the IP. Then I type `c$10.1.0.250` <Esc> to edit to the end of the line and exit insert mode.

Since ns2 is so similar to ns1, I can just type `yyp` <Ctrl-a> `$` <Ctrl-a> to copy and paste ns1, increment ns1 to be ns2, then move to the end of the line and increment the IP. Next I need to find the existing ns2 line and delete it with `/^ns2` <Enter> `dd`. Finally, I can type `q` to save the macro. The complete macro is `qngg/^ns1` <Enter> `/IN A` <Enter> `2wc$10.1.0.250` <Esc> `yyp` <Ctrl-a> `$` <Ctrl-a> `/^ns2` <Enter> `ddq`. Although this seems like a lot of text, it will save a ton of work when you have to repeat the steps on multiple files.

For the final macro w, I type `qw` to enter macro mode assigned to the w key, and then type `G` to move to the *bottom* of the file this time.

Then I type `/^worker` <Enter> `N` to search for the next worker (which will wrap around to the top, then `N` will move back to the last worker in the file). Finally, I type `yyp` to copy and paste the line, then <Ctrl-a> `$` <Ctrl-a> to increment both the worker hostname and the IP. Finally, I type `q` to exit macro mode. The complete macro is then `qwG/^worker` <Enter> `Nyyp` <Ctrl-a> `$` <Ctrl-a> q. Like the others, I test it out with `@w` a couple times, and use `u` to undo all the changes in between until I am satisfied that it works.

Now that I have all of these macros recorded, I could just open each file and type `@t` to update the TTL, `@s` to edit the contact information and serial, `@n` to update the name servers, and type `50@w` to add 50 more workers.

Because I'm going to perform these same steps on a number of files, I might as well capture all those commands in a new macro I'll assign to c. To do that, I just type `qc` to enter macro mode assigned to the c key, then type `@t@s@n50@w` to perform all of the previously recorded macros. Finally, I type `q` to exit macro mode. The complete set of keystrokes is `qc@t@s@n50@wq` to assign all of the above sets of keystrokes to a single macro. Now when I open the next file, I can just type `@c` to perform the complete list of steps.

Now besides saving a few keystrokes, there are other good reasons to nest macros in this way. Because I saved each logical step as its own macro, I can tweak or modify any of the above macros independently, save the new macro to the same key, and all of the other macros, including the final combo macro can stay the same.

Let's say that after I recorded all of these macros, I realized I got the IP address for the name servers wrong. All I would have to do is record a new macro assigned to the same n key, and once I was done, I still could use `@c` to make the complete set of changes to a file.

I hope you find these examples useful, and that the next time you have to perform a series of mundane edits to many text files, you'll save some keystrokes with vim macros.■

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.

SHAWN POWERS

This Is Why We Can't Have Nice Things: SSH

Firewalls are great, but don't rely on them as your only network security!

I've written about SSH before— often even. But for the Linux user, SSH is one of those tools that is so incredibly flexible, everyone should know and understand it inside and out. For this article, I decided to put on my black hat and demonstrate how convenient, and terrifying, SSH can be.

Phone Home

Most articles on SSH start with forwarding X11 traffic or demonstrating the SOCKS proxy feature. I want to start with something a little more creepy and a whole lot more awesome. The premise is this: you have a computer inside a firewalled, NAT'd network, and you want to access it remotely. If you're the sysadmin, you can just forward a port on the firewall into the computer. If you're not, however, you can use SSH to open a tunnel for you automatically. This does require a few things:

- You need a server running SSH with a public IP address. I use my co-located Raspberry Pi in Austria, but you can use your home connection as long as you set up a Dynamic DNS entry in case your IP changes.

- The firewalled network has to allow outgoing SSH connections. Most do, but if you can't SSH out from inside the firewall, you might have to do something creative like run SSH on port 443 on your server. (That port is usually open for HTTPS traffic, and since it's encrypted, it's tougher to sniff out your naughty deeds.)

- You need key pairs set up for password-less logins from the computer inside the firewall to your server with the public IP. (Here's a demo I did back in 2009 on how to set up SSH key pairs: **https://www.youtube.com/watch?v=R65HTJeObkI**.)

Once you have the prerequisites in place, the process is a simple one-liner. First, the command, then the explanation:

```
ssh -N -R 0.0.0.0:2222:127.0.0.1:22 user@remotehost
```

The `-N` flag tells SSH that you don't want an interactive shell; you just want to establish the connection. That means in order to take the tunnel down, you simply press Ctrl-C. It can become confusing if you have an open terminal connected interactively to a remote server. The tunneling still will work, but if you inadvertently type `exit`, it logs you out and kills the tunnel.

The `-R 0.0.0.0:2222:127.0.0.1:22` portion of the command is where the magic happens. What you're doing is creating a reverse tunnel, which allows anyone who can access your public IP server to `ssh` in to the server behind the firewall. In English, the command is saying, "Hi remote server, I'm stuck behind a firewall. Will you listen on port 2222 for anyone trying to connect, and if they do, please forward the traffic to me on my port 22."

The command can get creepier too. In my example, I just forwarded traffic to the public port 2222 to the internal port 22 behind the firewall. But if you were to change the command like this:

```
ssh -N -R 0.0.0.0:3389:192.168.1.5:3389 user@remotehost
```

You've now created a public-access tunnel directly to the Windows RDP server on 192.168.1.5 behind the firewall. Anyone on the Internet who connects their RDP client to the remote host's IP address will get the RDP login screen from 192.168.1.5. Freaked out yet?

One of the problems with this setup is the relative instability of the Internet. If the SSH connection is severed, the tunnel collapses, and you no longer can reach the computer inside the firewall. Thankfully, there's a really great tool to help with that too: AutoSSH.

AutoSSH

Having a diabolical tunnel to an internal network across the globe is only awesome until it stops working.

SSH is a finicky protocol, so the smallest blip over the Internet can cause the connection to fail. If you launch the SSH command with AutoSSH, however, it will monitor your connection and restart it if things go wrong. AutoSSH will keep trying too, so even if the network is down for an extended time, when it comes back up, the SSH connection will be re-established.

AutoSSH is available for just about every major distribution, but it has to be installed on the computer *inside* the firewall, because that's where the connection has to initiate from. How I do it is basically put something like this in my crontab to run on @reboot:

```
autossh -M 41000 -f -N -R 0.0.0.0:2222:127.0.0.1:22 user@remotehost
```

Only two of the flags are for AutoSSH; the others get handed off to SSH itself. Basically, the `-M 41000` establishes a monitor port for AutoSSH to use. It's possible to use SSH's built-in ability to monitor a connection, but I've had very bad luck with it. Using AutoSSH's `-M` flag seems to work the best. It doesn't matter what port you select, as long as it's not currently in use on either computer. The next flag, `-f`, just tells AutoSSH to run in the background and monitor the connection. The rest of the line is similar to what I showed typed above.

I used to write complex bash scripts to check for connectivity, and kill off then restart SSH, but using AutoSSH is much more efficient and reliable. I've been using it for years, and it is rock-solid. The most I've ever had to do is kill off defunct SSH connections on the public server if for some reason I can't connect. AutoSSH then happily creates a new connection, and I'm back in business.

What Good Is It?

I personally use this sort of setup to access internal servers that aren't accessible directly from the Internet. I don't like to open ports directly into internal servers if possible, so if I can grant myself access to multiple internal servers by SSH'ing to a remote, unrelated IP on a random port, I feel a little better about it.

I also use this sort of setup to expose multiple Web servers from inside a network to the outside. Figure 1 shows the basic premise. The command looks similar to the example above, but with multiple `-R` flags.

```
autossh -M 41000 -f -N -R 0.0.0.0:8001:192.168.1.10:80 \
-R 0.0.0.0:8002:192.168.1.20:80 user@remotehost
```

To note, that's all on one line, I

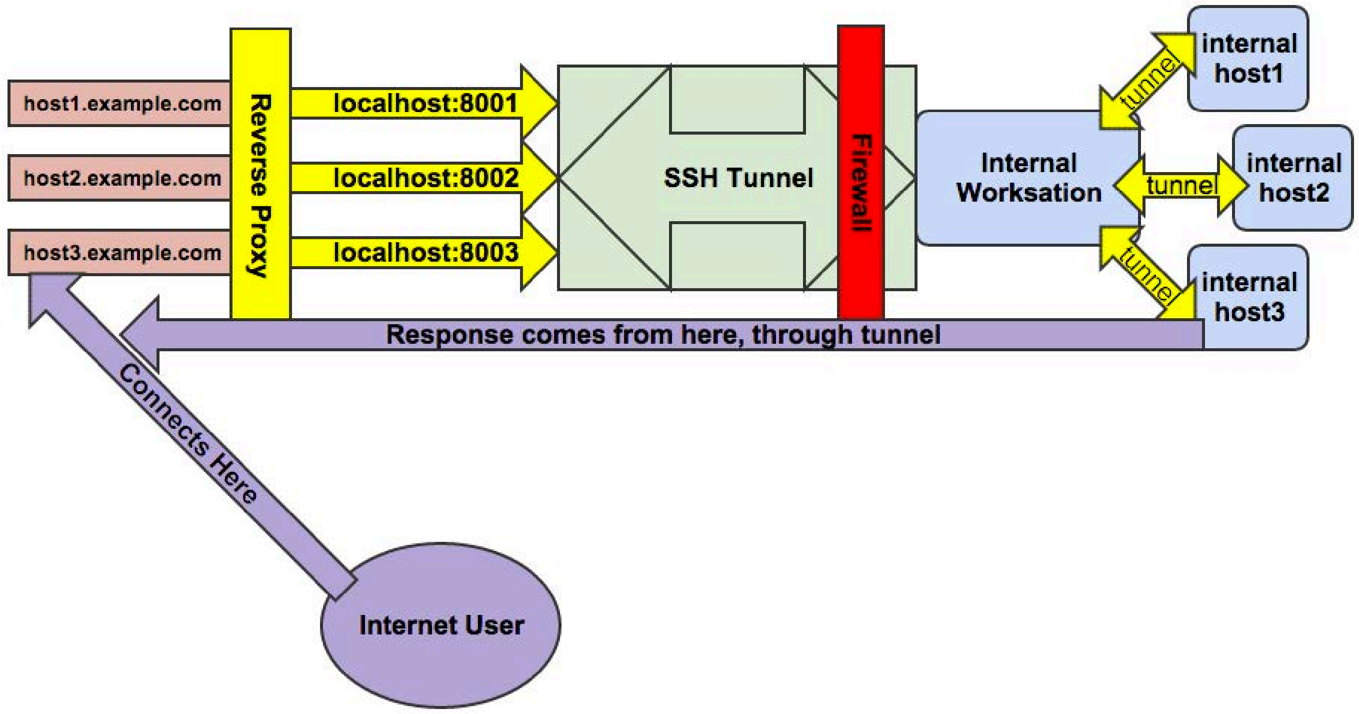


Figure 1. Although there's a lot going on behind the scenes, an end user accessing the virtual hosts on the left just sees a Web server.

just used the backslash so it formats better. Basically, by using multiple SSH reverse tunnels (the `-R` flags), internal Web servers are accessible by pointing the browser to http://remotehost:8001, http://remotehost:8002 and so on. I then use a reverse proxy (see my column in the August 2013 issue for details) to connect to those strange URLs with standard virtual hostnames. SSH is such a powerful, flexible tool, that its uses are seemingly unlimited! With that great power comes great responsibility though, because SSH allows you to do some pretty creepy things.

Even Scarier!

You may have noticed that using an SSH tunnel provides you only with specific access to specific ports spelled out with the tunnel directives. It is possible to pass multiple `-R` flags, but it's tough to do that on the fly, because the command is performed on the computer inside the firewall. If you need to access the entire network behind the firewall, that's where sshuttle comes into play.

I've mentioned sshuttle in past issues of *Linux Journal*, but when used in conjunction with the tunnels I just described how to create, it

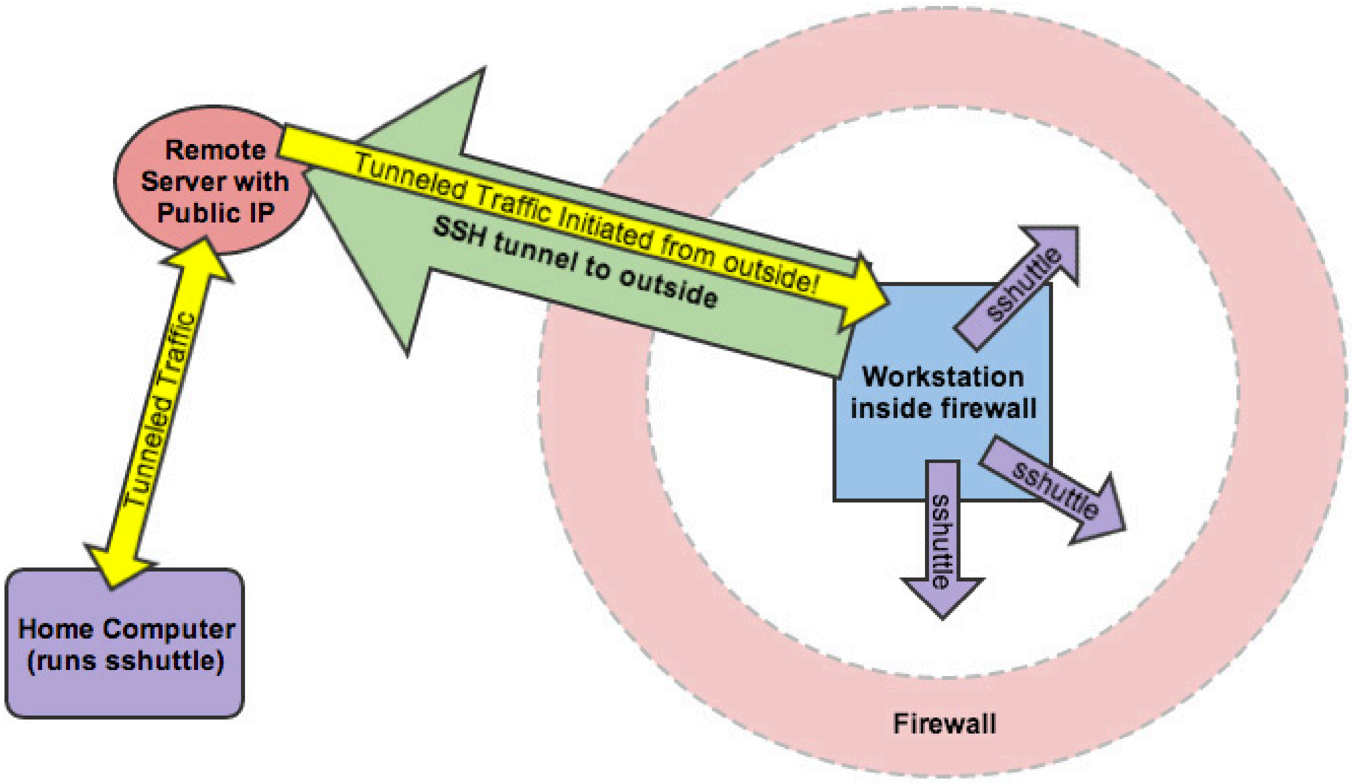

Remote Server with Public IP
Tunneled Traffic Initiated from outside!
SSH tunnel to outside
Tunneled Traffic
Workstation inside firewall
sshuttle
sshuttle
sshuttle
Home Computer (runs sshuttle)
Firewall

Figure 2. Using SSH tunnels and sshuttle together can provide an incredibly scary level of network access from outside the firewall.

turns into something that should be impossible, but isn't. Once you have the reverse tunnel established, using the processes above, the command to get you access to the entire firewalled network is another one-liner:

```
sshuttle -D -r user@remotehost:2222 192.168.1.0/24
```

As a reminder, this runs on your home workstation, not on the computer behind the corporate firewall. (See Figure 2 for a complete picture.) The `-D` flag tells sshuttle to run as a dæmon in the background. The `-r` flag tells it what remote server to connect through—in this case, user@remotehost on port 2222. Then the last part describes the internal network behind the firewall. This is something you'll need to know or figure out from your internal workstation. It will ask you for your sudo password, and then establish a network route through the SSH tunnel.

Basically, you've now not only accessed the workstation inside the firewall, but you have full access to anything that workstation has access to as well—from anywhere on the Internet.

We're All Doomed!

If you are feeling a bit like the nerdy hero in a modern espionage film, well yeah, I get it. There are some legitimate reasons to create tunnels like this, although admittedly an actual VPN is usually the "proper" way to go about it. It's important for those of us in charge of networks to realize how easy it is to gain access to internal systems, however. It's possible to block access like this at the firewall level, but honestly, there's always ways around the firewall if you're able to initiate internally. Plus, using draconian blocking methods will just inconvenience your users to the point of making them revolt. So what's a network admin to do?

Obviously, learning about network security is crucial. The reasons VLANs and NAC (network access control) systems exist is to prevent undesired access to various systems. When you're designing or redesigning your network, don't assume an external firewall will protect you from computers outside your network. Disgruntled employees, malware victims or nerdy employees like me will find a way to access systems from the outside. Make sure their point of entry doesn't give them access to systems they shouldn't have access to in the first place.

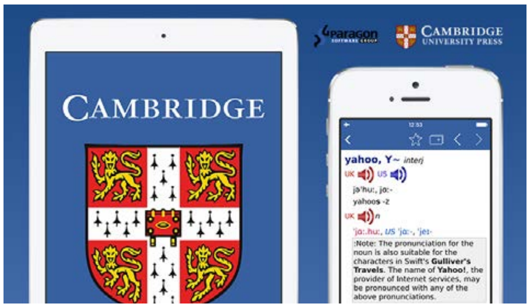
Today's little tutorial isn't really hacking. We're not doing anything the protocols aren't designed to do. Heck, all the tools are available pre-packaged in your distribution! I don't want anyone to spend too much effort trying to block my "attack", because it's not an attack at all. It's just using the tools available in exactly the way they're supposed to be used.

SSH is my favorite command-line utility. It can do so many things, from transferring files to tunneling X11 traffic. As I described here, you also can reroute traffic over tunnels giving you access to systems that shouldn't easily be accessible. Ultimately, I hope learning about SSH will get you interested in network security, because until you understand the danger, there's not much motivation to learning and implementing such systems. Until next time, happy tunneling! ■

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.

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

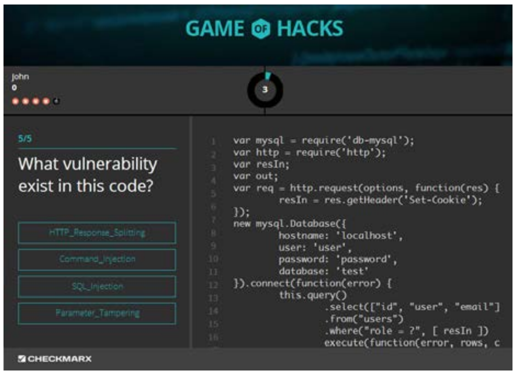
Paragon Software Group and Cambridge University Press' *Cambridge English Pronouncing Dictionary*


Paragon Software Group
Cambridge University Press
CAMBRIDGE
yahoo, Y~ interj
UK US
Note: The pronunciation for the noun is also suitable for the characters in Swift's Gulliver's Travels. The name of Yahoo!, the provider of Internet services, may be pronounced with any of the above pronunciations.

English-language learners—and American Anglophiles practicing their imitation of James Bond—will delight in the news that the *Cambridge English Pronouncing Dictionary* is now in app form. This latest version of the dictionary, "the classic guide to accurate, contemporary pronunciation for British and American English", is published jointly by Cambridge University Press, the world's oldest publisher, and Paragon Software Group. This dictionary app for Android, iOS and Mac OS X users is based on the classic version of Daniel Jones' guide to contemporary pronunciation for British and American English and is designed to meet the needs of English language learners striving to perfect their pronunciation of even the most challenging words, such as "though" and "hono(u)r", personal names, company names, technology and science definitions, and more. The app version contains a variety of features that aid learning (such as a Flash Card Quiz), search and navigation (such as full-text search) and convenience (no Internet connection required). http://www.cambridge.org and http://www.slovoed.com

Imagineer Systems Ltd.'s mocha Pro

Visual effects solutions developer Imagineer Systems Ltd. says that it has made its mark on such films as *The Hobbit*, *Black Swan*, *Transformers* and the *Harry Potter* series. The latest release in Imagineer's creativity toolkit is mocha Pro 4, the company's updated standalone software utility optimized for visual effects and post-production challenges. Key existing mocha Pro features include a rock-solid planar motion tracking engine, advanced roto tools, 3-D camera solver, stereoscopic 3-D support, calibration of lens distortion, stabilization module and insert module. New features found in version 4 include new advanced tools for stereoscopic 3-D, customizable keyboard shortcuts, Adobe Premiere support, Python scripting, improved format support and more. http://www.imagineersystems.com



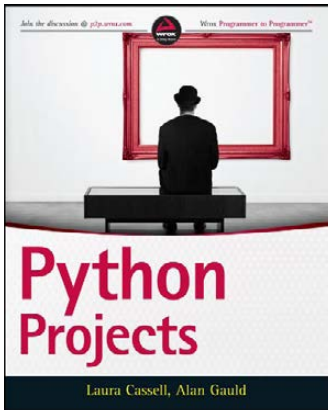
What vulnerability exist in this code?

```
var mysql = require('db-mysql');
var http = require('http');
var resIn;
var out;
var req = http.request(options, function(res) {
        resIn = res.getHeader('Set-Cookie');
});
new mysql.Database({
        hostname: 'localhost',
        user: 'user',
        password: 'password',
        database: 'test'
}).connect(function(error) {
        this.query()
                .select(["id", "user", "email"]
                .from("users")
                .where("role = ?", [ resIn ])
                execute(function(error, rows, c
```

Checkmarx *Game of Hacks*

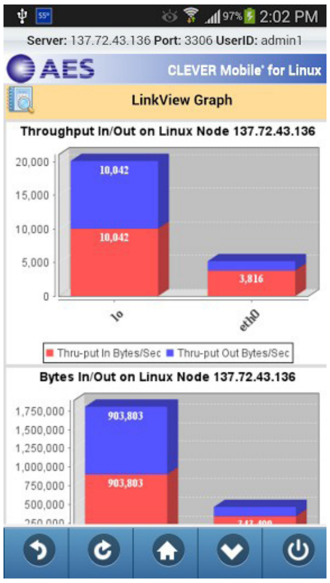
In its new on-line game *Game of Hacks*, Checkmarx allows developers to have fun while honing their security skills and understanding of the hacker's perspective. In the game, developers and security professionals test their application hacking skills, improve their code security know-how and facilitate better security practices in hope of reducing the amount of vulnerabilities in their applications. Available for desktop, tablet and mobile, *Game of Hacks* presents developers with vulnerable pieces of code and challenges them to identify the application layer vulnerability as quickly as possible. It even has a two-player mode allowing head-to-head competition. Players analyze vulnerabilities including SQL injection, XSS, log forgery, path traversal, parameter tampering and others in myriad programming languages. Additionally, developers can add their own questions and vulnerable code to the game in any programming language highlighting any vulnerabilities, growing the game's scope as more users join.
http://www.gameofhacks.com

Laura Cassell and Alan Gauld's *Python Projects* (Wrox)

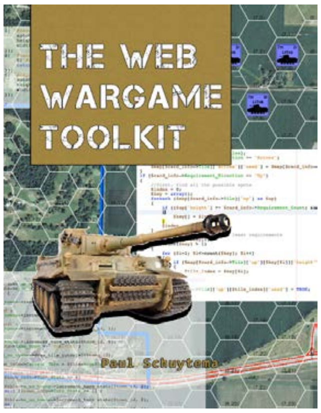


Python trainers Laura Cassell and Alan Gauld targeted their new book *Python Projects* at the Python programmer with basic skills who is ready to start building real projects. Programmers who know the Python syntax and lay of the land, but who still may be intimidated by larger, more complex projects, will find this book useful. *Python Projects* provides a walk-through of the basic setup for an application and the building and packaging for a library, and explains in detail the functionalities related to the projects. Topics include maximizing the power of the standard library modules; finding and utilizing third-party libraries; creating, packaging and reusing libraries within and across projects; building multilayered functionality, including networks, data and user interfaces; and using development environments like virtualenv, pip and more. Publisher Wrox says that Python developers looking to apply their skills to real-world challenges will find a goldmine of information and expert insight in *Python Projects*.
http://www.wrox.com

AES' CleverView for TCP/IP on Linux and CLEVER Mobile for Linux

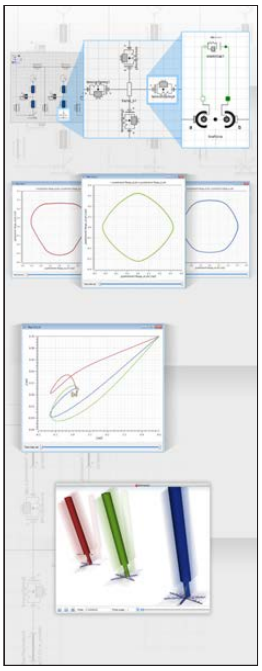

2:02 PM
Server: 137.72.43.136 Port: 3306 UserID: admin1
AES
CLEVER Mobile' for Linux
LinkView Graph
Throughput In/Out on Linux Node 137.72.43.136
10,042
10,042
3,816
lo
eth0
Thru-put In Bytes/Sec
Thru-put Out Bytes/Sec
Bytes In/Out on Linux Node 137.72.43.136
903,803
903,803

The CleverView for TCP/IP on Linux solution from AES addresses the challenge in cloud-based data centers that an ever-greater amount of traffic is between servers, preferably Linux-based ones, of course. CleverView is a performance and availability monitoring solution for Linux, UNIX and System z that has added new features in its latest v2.3 release. Concurrently, AES released an updated CLEVER Mobile for Linux so that enterprise knowledge workers can access server performance and availability details from their mobile devices. The most noteworthy highlight of CleverView v2.3 is the SMF Host Utility that integrates Linux/UNIX-monitored metrics with z/OS SMF records, expanding enterprise insight into performance and availability. Also noteworthy is the real-time notification of problems leading to increased Linux and UNIX service availability, regardless of the hardware platform on which Linux or UNIX resides.
http://www.aesclever.com



Paul Schuytema's *The Web Wargame Toolkit* (Mercury Learning & Information)

Paul Schuytema's new book *The Web Wargame Toolkit* walks readers through the process of crafting an old-school, turn-based wargame in PHP utilizing the CodeIgniter application framework. More generally, this is a soup-to-nuts how-to book for those interested in crafting Web game applications utilizing a basic LAMP system. The wargame project serves as an exciting context for PHP programmers looking to gain more in-depth application coding experience. Those with an interest in game development will learn the coding skills to support design decisions as they create the game project. The book assumes at least a beginning level of PHP coding experience, though no previous CodeIgniter or game development experience is required. Other features of the book-DVD set include full database design for game data structures, a flexible game map system, a complete overview of a simple CRUD system and complete, modifiable source code for the game.
http://www.merclearning.com



Wolfram SystemModeler

The updated Wolfram SystemModeler 4 is an easy-to-use, next-generation modeling and simulation environment for cyber-physical systems. SystemModeler enables users to draw on a large selection of built-in and expandable modeling libraries to build industrial-strength, multidomain models of a complete system. Wolfram is also well known for the complementary Mathematica application, which provides a fully integrated environment for analyzing, understanding and quickly iterating system designs. The new SystemModeler 4 vastly expands support for modeling libraries, adds standardized deployment of models to other simulation tools and deepens integration with Mathematica. Other key new features include a library store with verified model libraries, improved modeling features, model creation support from Mathematica, as well as improved workflow in the integration with Mathematica, a new documentation center and support for our beloved Linux OS.

http://www.wolfram.com

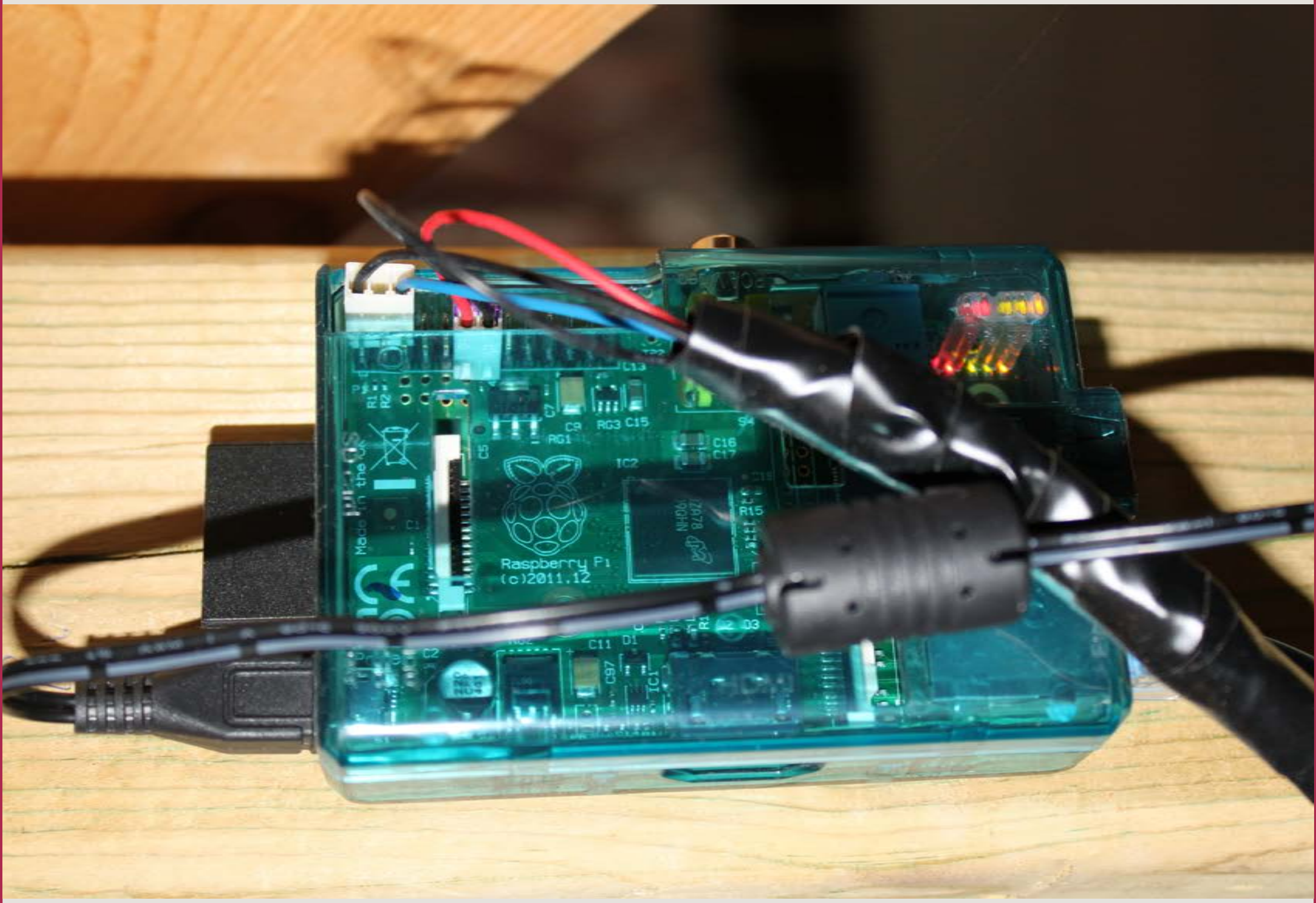
CacheGuard Technologies Ltd.'s CacheGuard OS

CacheGuard Technologies Ltd.'s motto is "Web security made affordable". The company puts its motto into action with products like CacheGuard OS, an integrated security solution designed to manage Web traffic that is based on a custom-hardened version of Linux. The latest "next-generation" release is CacheGuard OS NG v1. The company calls CacheGuard OS a powerful, turn-key solution that allows companies to protect and optimize Web traffic traversing their Web infrastructure. Customers can utilize CacheGuard OS as an appliance or turn their own hardware into a powerful Web Gateway Appliance. CacheGuard OS integrates numerous Web security and optimization technologies in a single functional network device. Technologies like proxy, IP firewall, bandwidth shaping, caching, HTTP compression, URL filtering, Web application firewall and Web malware filtering are all integrated into a unique operating system.

http://www.cacheguard.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.

RASPI-SUMP



How to set up a sump pump monitor with an ultrasonic sound sensor, Raspberry Pi and Python.

AL AUDET

In June 2013, we had the unfortunate luck of a basement flood, caused by a tripped electrical breaker connected to our sump pump. There are so many things that can go wrong with a sump pump. You always are on guard for power outages, blown breakers, sump pump failures, clogged pipes and all manner of issues that can arise, which ultimately can end with a flooded basement. I needed a way to alert me of issues when I was not at home. Audible alarms are fairly cheap and are great when you are physically in the house. They fail miserably when you are ten miles away at work. I had a Raspberry Pi that I had tinkered with periodically but for which I never had a real purpose. I decided to try to put the Pi to work as a dedicated sump pit monitoring device. Hopefully, the Pi could send me SMS alerts if a problem arose while I was away.

Since I did not have a programming background, I started to look for an existing project I could install on the Pi that could act as a sump pit monitor. There are other projects that can monitor sump pump activity; however, it seemed that everything I came across looked overly complicated or didn't have the features I required. I needed something simple that monitored the water level in the sump pit at regular intervals and sent me a text if there was a problem. If it also could display pretty graphs of sump pit activity that I could access easily, that would be a bonus.

Although I had written many scripts through the years at work, I never learned object-oriented programming. I made the decision to learn Python, and a few months later, set myself to work on a monitoring system. I chose Python because it has an active community, and many Raspberry Pi enthusiasts use it as their main scripting language. The Raspberry Pi uses Raspbian Linux, which is based on Debian, so that already was familiar ground. With these tools in hand and in true Linux and Raspberry Pi spirit, I decided to build my own and called it Raspi-Sump.

Raspi-Sump is a sump pit water-level monitoring system written in Python. It uses a Raspberry Pi and an HC-SR04 ultrasonic sensor to monitor the water level in a sump pit, log the readings and send SMS e-mail alerts if the water rises above a predefined level.

In this article, I show the methodology I used to create Raspi-Sump. I also describe the physical setup of the monitor and the scripts that make it work. If you choose to do something similar, the source code and install instructions are available on GitHub.

With the help of a Python script, the sensor, which is mounted inside the sump pit facing the water, sends a sound pulse that reflects off the water and back to the sensor.

It is free to use and modify as you wish (see Resources).

I determined that the features I required in a monitor included the following:

- Regular one-minute-interval readings of the water depth in my sump pit.
- Logging of readings to a comma-delimited file for processing graphs and historical pump activity.
- Automated SMS e-mail alerts if the water exceeds a predefined level.
- Off-site graphical reports of the current water level to a Web site.
- Web-based historical information on sump pump activity.
- Automatic restart of the raspisump.py process after an unexpected failure.

The Physical Setup

The complete list of components for Raspi-Sump includes:

- Raspberry Pi Model B and case.
- Raspbian Linux.
- HC-SR04 ultrasonic sensor.
- Five feet of Cat5 wire (four 24AWG strands needed).
- Two resistors (one 470R Ohm and one 1K Ohm).
- Heat-shrink tubing to protect soldered connections.
- Plastic bracket to hold the sensor.
- One two-foot piece of wood strapping to mount the plastic bracket in the pit.
- One floppy drive four-pin power connector salvaged from an old PC.
- Two case-fan power connectors, also salvaged from the same PC.

Total cost for materials, including a couple spare sensors, was $80.

The ultrasonic sensor I chose is the HC-SR04, which has four connections that are wired to the GPIO pins of the Raspberry Pi. With the help of a Python script, the sensor, which is mounted inside the sump pit facing the water, sends a sound pulse that reflects off the water and back to the sensor. The script monitors the amount of time it takes for the sound pulse to bounce back to the sensor. It calculates the distance by measuring the time required for the pulse to return at the speed of sound. This gives you a reading of the distance between the sensor and the water. The distance is used to calculate the water depth and log a time-stamped result to a CSV file.

Figure 1 shows a closer look at the connections.

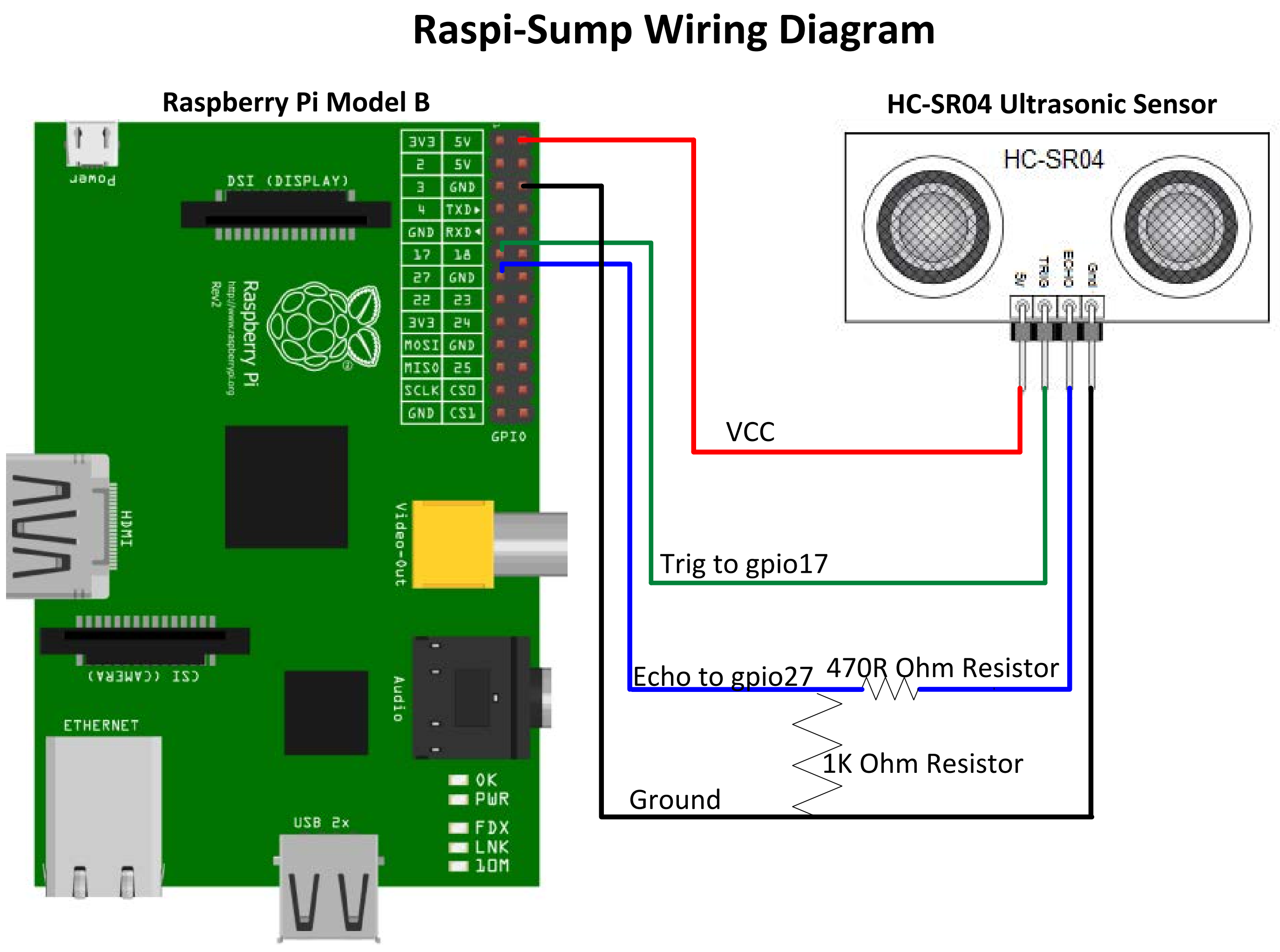


Figure 1. Wiring Diagram

The four pins on the sensor are wired to the Raspberry Pi as follows:

- Pin 1 VCC connects to the 5V pin 2.
- Pin 2 Trig connects to GPIO17 pin 11.
- Pin 3 Echo connects to GPIO27 pin 13.
- Pin 4 Ground connects to pin 6 Ground.

I chose GPIO17 and 27, but you can use any available GPIO pins on the Pi as long as they are identified properly in the Python script.

Pin 1 provides 5V of power to the HC-SR04 sensor. A command is initiated on GPIO17 (Trig) that sets the value of the pin to True for 10 micro seconds. This causes the sensor to initiate a series of sound pulses toward the water for that short amount of time. The Echo pin connected to GPIO27 listens for a return pulse. The difference between the send and the return of the pulse gives a time measurement. The measurement is used to calculate the distance of the water.

This causes a small problem as Raspberry Pi GPIO pins are rated only for 3.3V. The sensor sends a 5V current back toward GPIO27. A way is needed to throttle the current to 3.3V, which won't damage the Pi. To protect the Pi from damage, simply insert a voltage divider on the Echo line between the sensor and the Pi.

Voltage Divider

The purpose of a voltage divider is to reduce the amount of current sent from one component to another. As shown in Figure 1, I soldered a 470R Ohm resistor on the Echo wire and bridged a 1K Ohm resistor between the Echo and Ground wires. This prevents blasting 5V to a pin that is rated only for 3.3V. With these resistors, voltage is actually a touch higher at 3.4V, which is within a tolerable level. All soldered connections are covered with heat-shrinking tube to avoid electrical shorts.

Calculating resistor types required is beyond the scope of this article, but there are many handy Web-based voltage divider calculators available to determine your requirements. In this example, a 1K and 2K Ohm resistor would reduce the current to 3.333V.

Wiring

The Raspberry Pi is connected to the sensor with a five-foot length of

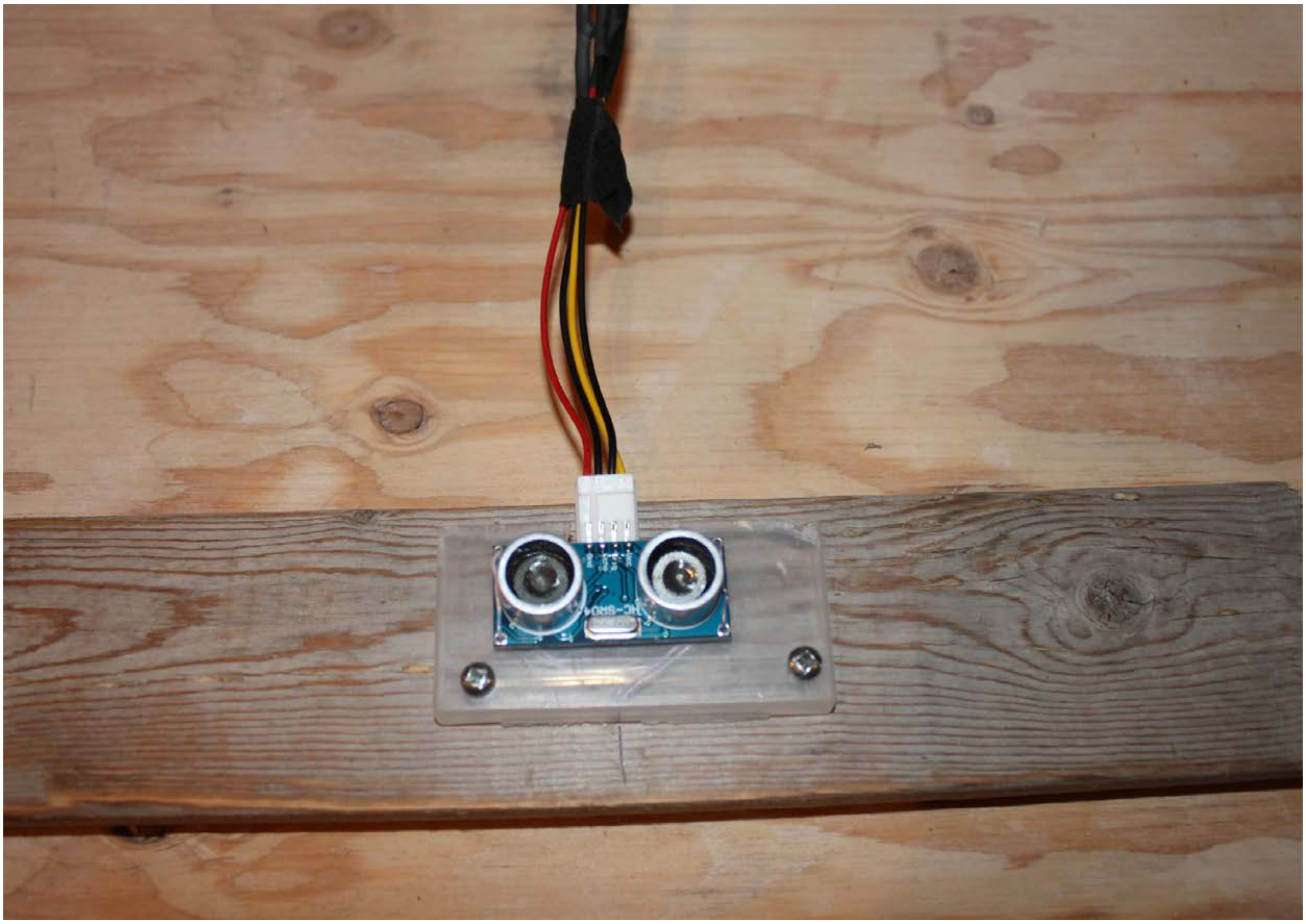


Figure 2. Floppy Connector

CAT5 cable. Because there are four connections, only four of the eight twisted wires are used. On each end of the selected wires, I soldered connectors that were compatible with the sensor pins and the pins on the Pi. An old 3.5" floppy drive power connector works great for the sensor connection (Figure 2). I used a couple two-pin PC case-fan connectors, salvaged from an old PC, for the connections on the Pi's pins. These connectors are available on-line, but anything you can salvage from an old PC works great.

Mounting

The HC-SR04 is attached to a plastic case and screwed onto a piece of wood strapping. The wood strapping is inserted into the sump pit facing downward and is easily adjustable and removable if needed. The Cat5 wire is securely taped to the sump pump's ABS pipe and an open wall stud to prevent tangling and

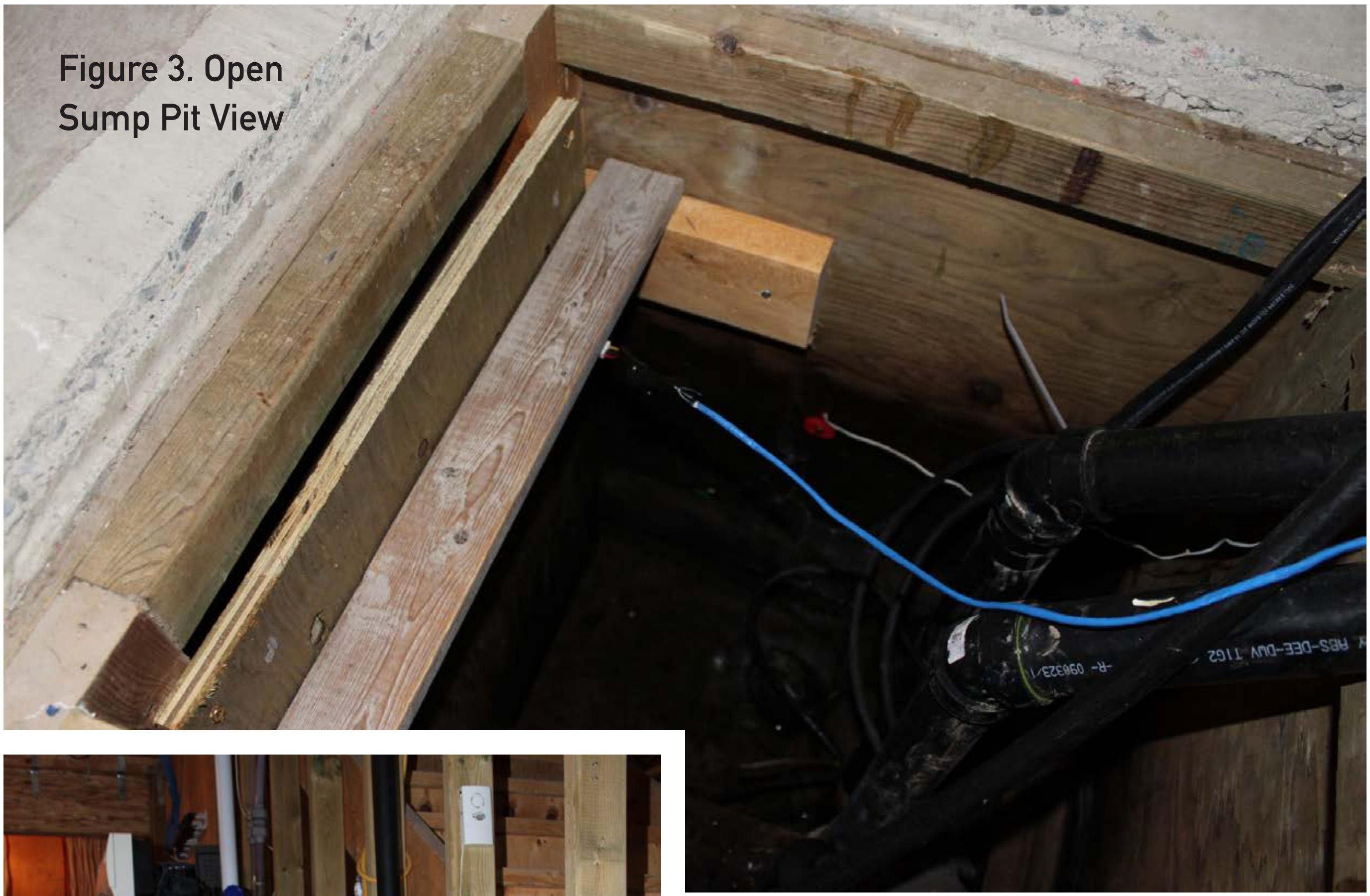


Figure 3. Open Sump Pit View

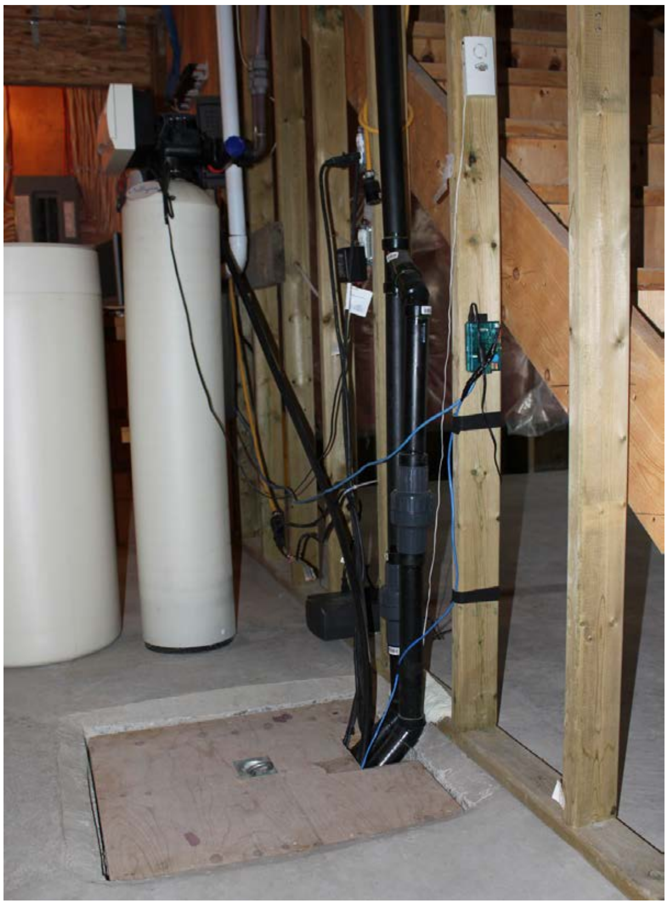


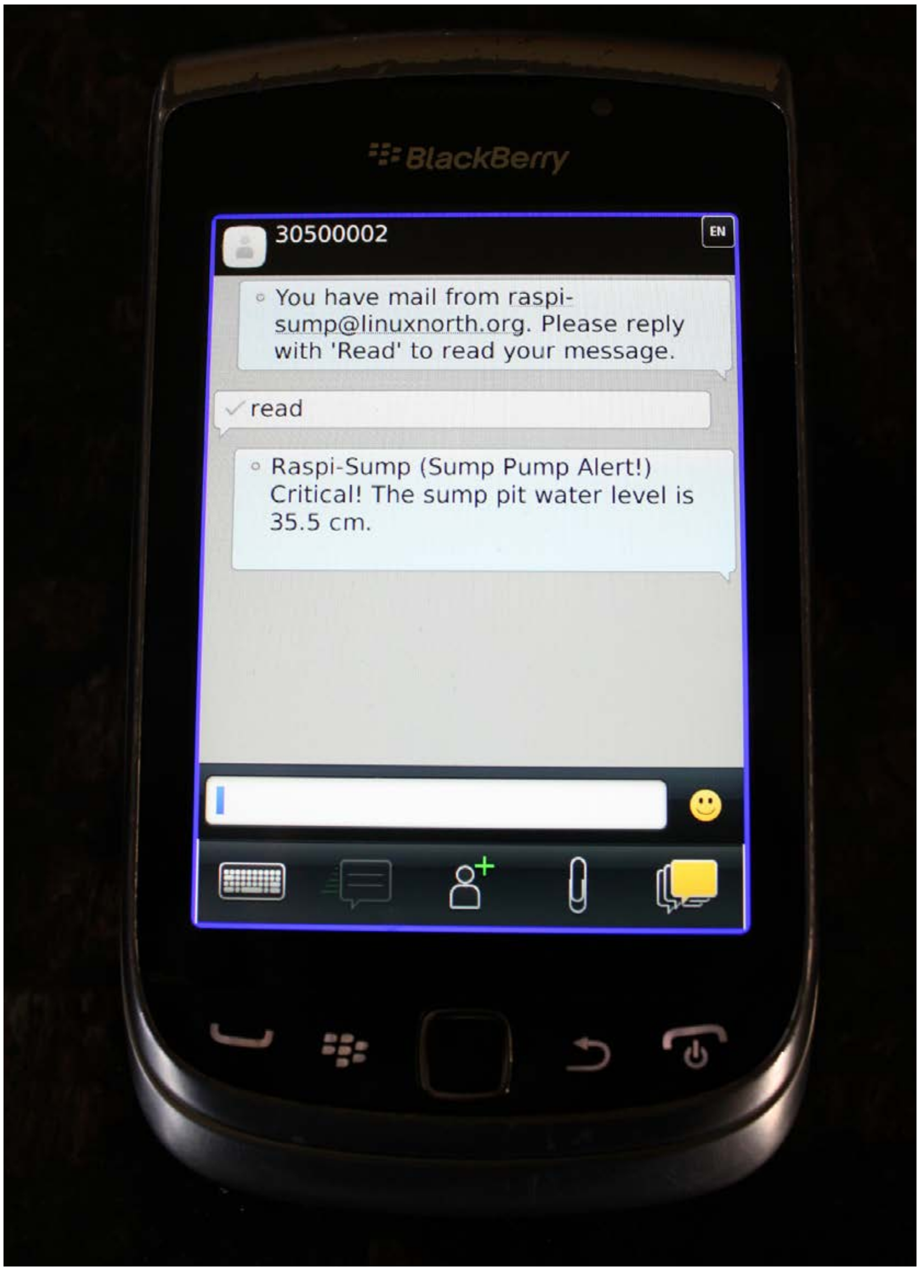
Figure 4. Finished Pit View

disconnection of the wire when removing the sump pit lid.

Finally, the Raspberry Pi is mounted on a wall stud and plugged in to a UPS unit. Figure 4 shows the finished view.

Raspi-Sump

The Raspi-Sump program currently consists of three Python scripts. The main script is raspisump.py. The script is very simple and is only about 100 lines of code. The first thing it does is set the variables of the sump pit, like depth (72cm), critical water level (35cm) and GPIO pin assignments as mentioned earlier. The script then takes a sample of 11 water-level readings every minute and uses the median sample as the best reading (more on this later). Once the reading is established, the script determines if the water is at a safe or critical level. Safe levels are logged to a CSV file, and the script waits for another minute to take the next reading. Critical levels are passed to a function that logs the level to the same CSV file and initiates an SMS e-mail to my cell phone (Figure 5). I use the Python smtplib module to handle e-mail alerts. You can configure any e-mail server to handle the alerts, including a localhost mail server on the Pi, if your ISP allows port 25 traffic. You also can use your ISP's SMTP server or Google's Gmail SMTP server if you are using a Gmail account.



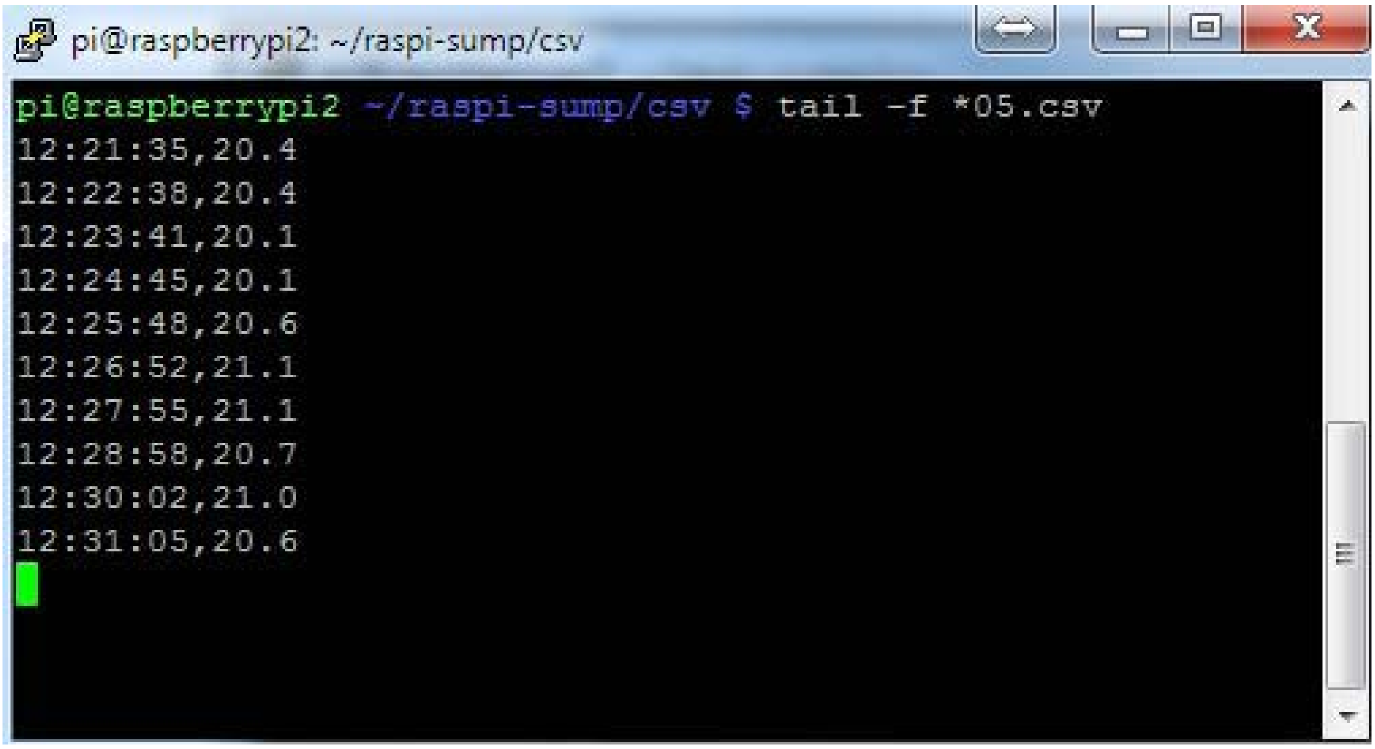
You have mail from raspi-sump@linuxnorth.org. Please reply with 'Read' to read your message.

read

Raspi-Sump (Sump Pump Alert!) Critical! The sump pit water level is 35.5 cm.

Figure 5. SMS Alert

The key Python module used to communicate between the Pi and the sensor is called RPi.GPIO. This module can be used to control so many different types of equipment with your Pi. Without delving into the "nuts and bolts" of RPi.GPIO, the module helps you take control of the pins by turning them on and off. This allows you to



```
pi@raspberrypi2 ~/raspi-sump/csv $ tail -f *05.csv
12:21:35,20.4
12:22:38,20.4
12:23:41,20.1
12:24:45,20.1
12:25:48,20.6
12:26:52,21.1
12:27:55,21.1
12:28:58,20.7
12:30:02,21.0
12:31:05,20.6
```

Figure 6. CSV File Being Updated in Real Time by raspisump.py

control all sorts of equipment, like sensors and LEDs, for example.

You can view the GPIO code in the raspisump.py script within the water_level() function. Similar code is used by many other projects that communicate with the Pi's GPIO pins. Adam Lappin's Byte Creation Blog has a good example that helped me learn how to use the RPi.GPIO module in this project (see Resources).

Raspi-Sump is started automatically on bootup of the Raspberry Pi by adding this line to /etc/rc.local right before the last line `exit 0`:

```
/home/pi/raspi-sump/raspisump.py &
```

The ampersand (&) starts the script as a background process.

Access to GPIO pins requires elevated privileges on the Pi. To start the script manually, issue the command:

```
sudo /home/pi/raspisump/raspisump.py &
```

Figure 6 shows using the `tail` command to demonstrate the CSV file being updated in real time by raspisump.py.

What is displayed in Figure 6 is rather strange. The water depth is bouncing around. You would expect the water to be consistently higher with each reading. The reason for

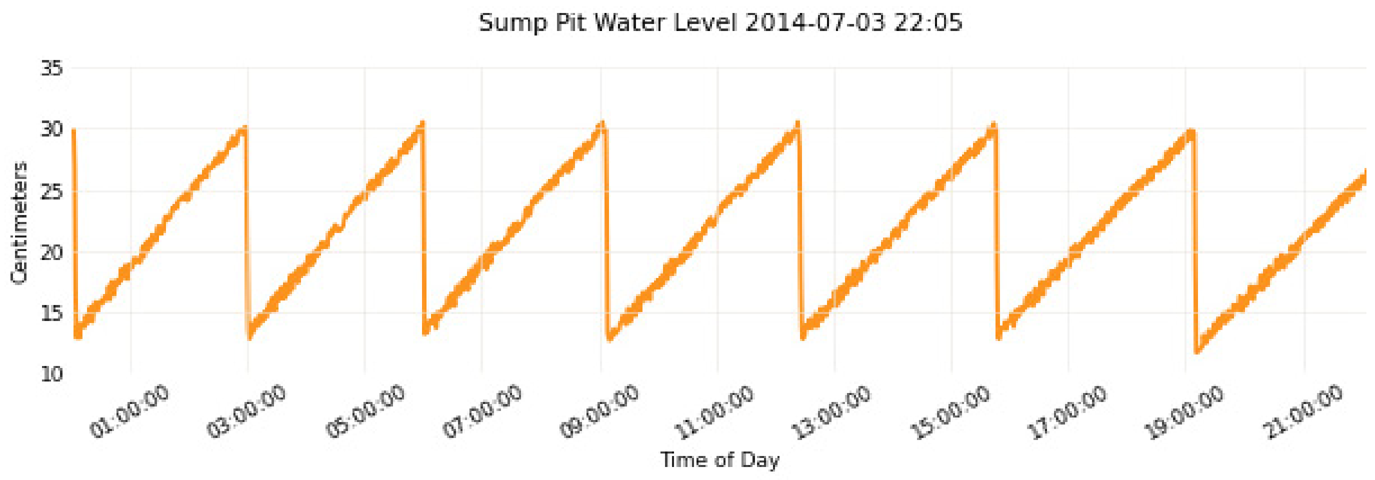


Figure 7. Graphs Generated by todaychart.py

this is that there is a one-centimeter variance in each reading. Linux is a multitasking OS and not a real-time one. It is not optimal for real-time applications like communicating with sensors and returning precise results. The best reason I can come up with is that the OS is busy doing other tasks and allows raspisump.py to record the reading when it is finished dealing with those other processes.

This brings me back to the reason I use the median reading of a sorted sample. Every once in a while, the script gives an invalid reading that can be way off. This can trigger a false warning SMS alert even if the water is below my critical level. However, these readings are rare. By using a sorted sample, I can remove those fringe readings at the high and low end if they occur. The median reading is always accurate within one centimeter of the actual water level. For a residential system, I am not concerned with millimeter accuracy. A small variance in readings still provides safe reporting of the water level. This also helps explain the jagged line in the graphs that are generated and sent to a Web server at regular intervals.

The second script I use is todaychart.py. This script generates graphs, as shown in Figure 7, of water level activity from my CSV log files. It uses the Python matplotlib and NumPy modules to generate the graphs. rsync over SSH copies the graphs and CSV log files hourly to my Web server via a cron script. I chose to generate graphs on the Pi instead of the Web server, because different Linux distributions package different versions of matplotlib and NumPy. I prefer using the packaged versions for simplicity. Always using the Raspberry

Pi renders more consistent graphs, no matter which distro you use for your off-site component.

The third and final script is checkpid.py. Its purpose is to monitor the health of the raspisump.py process and restart it if it is stopped. Cron runs the script at regular intervals and looks for one of three outcomes. If the script returns 0, this indicates a failed process. checkpid.py then initiates a restart command. If the script returns 1, the process is fine, and the script exits cleanly. If the number is greater than 1, this indicates more than one raspisump.py process. In this instance, a `killall 09 raspisump.py` directive is initiated, and the process is restarted.

Other Issues with Raspi-Sump

The HC-SR04 sensor has a fairly wide sonar field. The user manual states that it works best with a 30° angle. My sump pit is a busy place. It has a backup pump that sits higher than the main pump on a 2x6 stud. Each pump has a float ball that bounces around in the pit. This results in false readings when the sensor picks up an object that enters its field. This problem can be mitigated by strategically placing the sensor further away from these objects. If that is not possible, you can vertically insert a 3" or greater piece of PVC pipe in the sump pit and force the sensor to take its reading down the empty pipe. This will focus the pulse and hide the objects in the pit that are causing problems.

Conclusion

Raspi-Sump is still in the early stages of development. There are other features I would like to add, such as:

- A manual power button to start and shut down the Raspberry Pi gracefully without logging in.
- A small LCD display to show the current water level without opening the lid.
- A Web-based reporting system using a Python Web framework.
- A Web-based management interface for Raspi-Sump on the Pi (like a home router).
- A GSM module component to use the cellular network for alerts instead of the Internet.
- A configuration file to store variables as opposed to within the script.
- Package management for installation of Raspi-Sump.

A sump pit monitor is just one tool you can use to help avoid a flooded basement. It's not a replacement for a complete strategy that includes a backup pump on a separate electrical breaker. A gas-powered electrical generator is also essential for extended power outages. Also, I kept my cheap Home Depot audible alarm. A text alert at two in the morning is useless if I am sound asleep. I want a "full-out" screech to wake me up.

I welcome all feedback on this project. I am not a professional programmer, and I am sure that I can substantially improve the code or add useful features that I have not even considered.

Although it's not perfect, I now have a system that works and gives me extra peace of mind while I am away. If you are looking for a similar solution, I hope you can use, modify and improve Raspi-Sump to suit your needs. If you do, I would love to hear from you.

Acknowledgement

Special thanks to Ron Hiller (GitHub user @rhiller) for tirelessly answering my questions about voltage dividers and his own sump pump monitor called pi-distance: **https://github.com/rhiller/pi-distance**.■

Al Audet lives in Timmins, Ontario, with his wife Lucie and five-year-old son Nicholas. He works as a Team Leader and Technical Advisor in Platform Virtualization Services for Shared Services Canada and has been a Linux enthusiast and advocate since 1997. He loves all technology whose purpose is to increase quality of life. He is user @Al_Audet on Twitter.

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

Resources

Raspi-Sump Web Site: **http://www.linuxnorth.org/raspi-sump**

Source Code: **https://github.com/alaudet/raspi-sump**

Quick Start Guide: **https://github.com/alaudet/raspi-sump/tree/master/docs**

MIT License: **https://github.com/alaudet/raspi-sump/blob/master/License**

HC-SR04 User Manual: **http://www.linuxnorth.org/raspi-sump/HC-SR04Users_Manual.pdf**

Adam Lappin's Byte Creation Blog:
http://www.bytecreation.com/blog/2013/10/13/raspberry-pi-ultrasonic-sensor-hc-sr04

U-Boot Environment Variables

A close look at the anatomy of the U-Boot environment.

SACHIN VERMA

Das U-Boot is a popular bootloader for embedded systems. This wide adoption of U-Boot is hardly surprising given the number of architectures and platforms it supports. Additionally, U-Boot has a flexible compile-time configuration setup. You can select different features and drivers via config options and build a custom bootloader image for your platform. U-Boot's flexibility is extended at runtime as well. Using U-Boot environment variables, you can influence the program execution flow.

U-Boot comes with a CLI (command-line interpreter), which has basic scripting capabilities. This scripting ability combined with the U-Boot environment variables can be used to create some powerful booting scenarios. The ability to manipulate program behavior using environment variables is beneficial for both development and production setups alike. During development, people strive to test all possible paths for loading and booting images for their platforms. So, you may try to load a Linux kernel image from a local storage (Flash, SDcard, USB, eMMC and so on), or access it over the network (NFS, TFTP and so on).

U-Boot simply makes your life easier as a developer. You just need to tweak the scripts combining environment variables in a fruitful way. Production images also need some versatility. When a product's OS images need an upgrade, the bootloader must be configurable to fetch the images from different sources.

U-Boot has a number of system variables that you can modify to achieve your desired results. For example, on certain systems, initrd images loaded on top of DDR may not be accessible to the Linux kernel. To counter this, you can instruct U-Boot to load initrd at a lower DDR address. You can do this by setting the `initrd_high` environment variable.

Another common situation during development is the presence of different network configurations. On your home setup, you may be working on a static IP configuration using NFS. But, when you are out for a demo at a client location, you only have DHCP available with images kept on a TFTP server. U-Boot is highly configurable for such scenarios because it provides so many options. You can change the network configuration, modify the IP addresses of image servers and gateway servers with the help of environment variables. You could assign a console over serial port, or you could use netconsole or usbtty if you prefer.

So, What Is This U-Boot Environment?

A simple answer to that question would be "a collection of name=value pairs". Here, "name" refers to the name of the environment variable to which you want to assign some "value". This "value" could be of any type: string, hexadecimal, boolean and so forth. Whatever type the value is, it is converted into a string before being stored in a linearized environment data block. Each environment variable pair ("name=value") would be stored as a null-terminated string. So, the collection of many environment variables is nothing but a null-separated list with a double-null terminator. Figure 1 illustrates how a list of strings is actually stored. The left-hand side is just a logical representation of environment variables, whereas the right-hand side shows that the variables have been flattened and written in a serialized form.

How Is the Environment Stored?

U-Boot has two types of persistent environments.

1) Default Environment (Compiled-In, Read-Only): Every U-Boot binary has a default built-in environment of its own (Figure 2a). During compilation, a character array called `default_environment` is embedded into the U-Boot image. This character array stores the environment variables as a list of null-terminated strings with a double-null terminator. The contents of this array are populated conditionally based on the config options selected for your board. Environment variables that are commonly used can be enabled by defining the corresponding

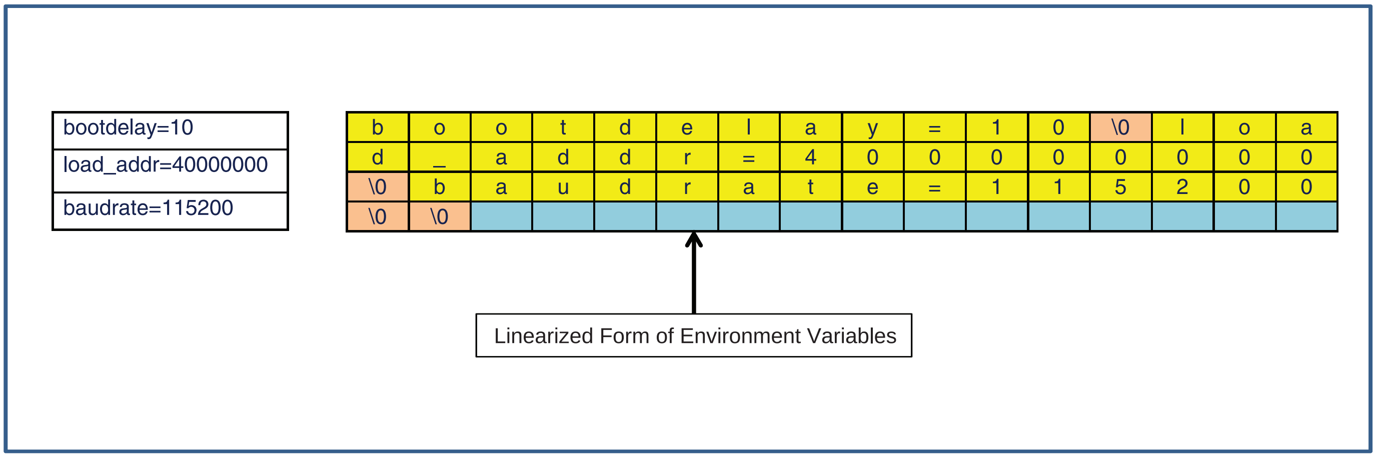
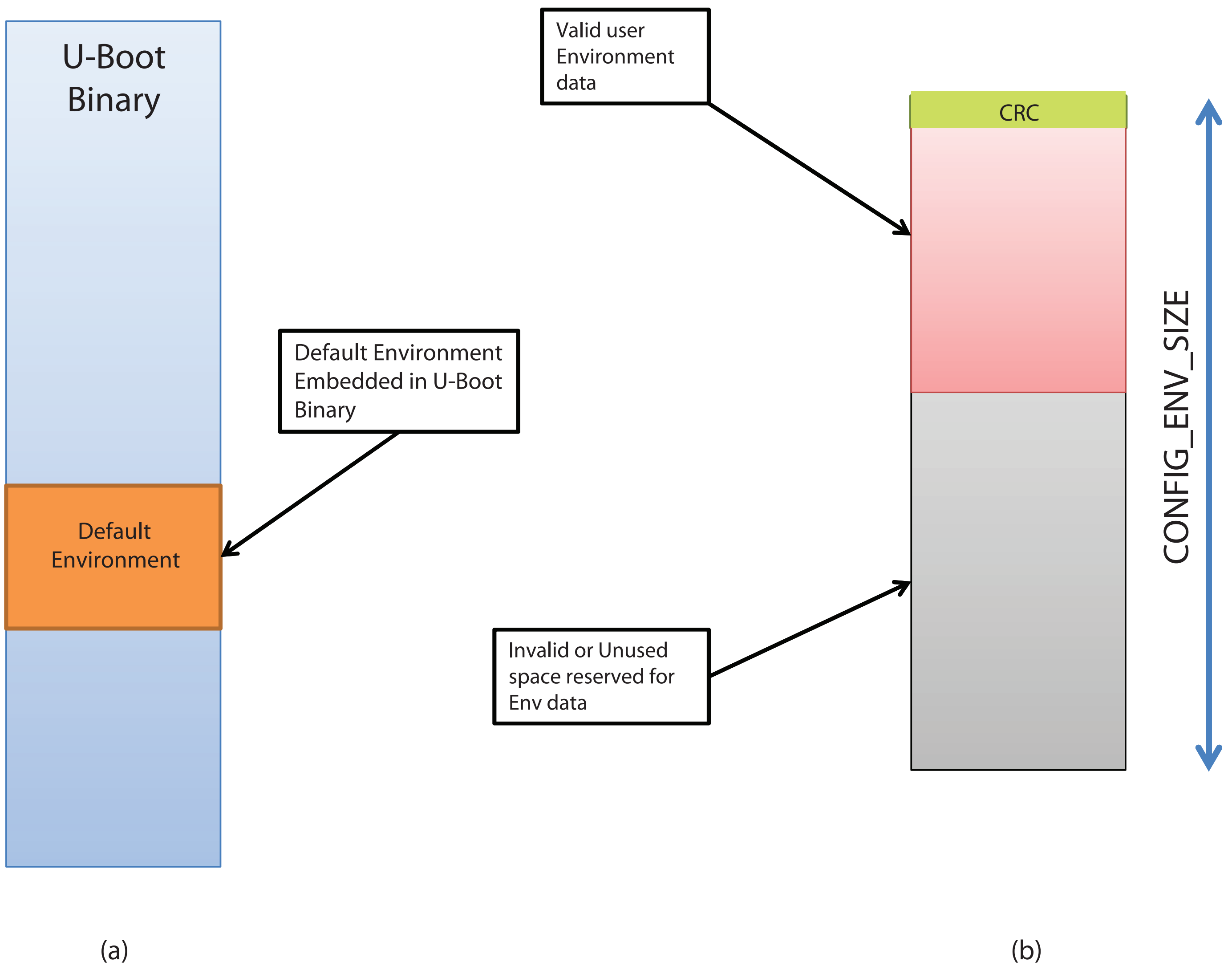

bootdelay=10
load_addr=40000000
baudrate=115200
b o o t d e l a y = 1 0 \0 l o a
d _ a d d r = 4 0 0 0 0 0 0 0 0
\0 b a u d r a t e = 1 1 5 2 0 0
\0 \0
Linearized Form of Environment Variables

Figure 1. Linearized Representation of Environment Variables



U-Boot Binary
Default Environment
Default Environment Embedded in U-Boot Binary
Valid user Environment data
CRC
CONFIG_ENV_SIZE
Invalid or Unused space reserved for Env data
(a)
(b)

Figure 2. (a) Type 1 Persistent Environment Read-Only, Embedded; (b) Type 2 Persistent Environment User-Supplied, Read-Write Enabled

CONFIGs in your board's config file (include/configs/<YOUR_BOARD>.h). Figure 3 lists some commonly used options, which, once defined, would make their way into the default environment of your board.

Apart from the standard variables used across boards, you may want to add certain environment variables that are specific to your board or that just are convenient for you. You may, for instance, want to embed the revision number of the board into this environment. You could do that by defining all of these variables in a macro

```
              Include/configs/<YOUR_BOARD>.config
. . .

#define CONFIG_BOOTARGS        "console=ttyAMA0,115200"
#define CONFIG_IPADDR          192.168.0.42
#define CONFIG_GATEWAYIP       192.168.0.1
#define CONFIG_ETHADDR         DE:AD:BE:EF:01:01
#define CONFIG_SERVERIP        192.168.0.2

. . .
```

Figure 3. System variables defined in your board's config become part of the default environment.

called CONFIG_EXTRA_ENV_SETTINGS in your board's config file:

```
#define CONFIG_EXTRA_ENV_SETTINGS \
        "board=" XSTR(BOARD) "\0" \
        "load_addr=" XSTR(CONFIG_SYS_LOAD_ADDR) "\0"
```

Remember that the default environment is "read-only", as it is part of the U-Boot image itself. Vendors normally keep some essential system variables as part of this environment.

There are some good reasons to keep a default environment as part of the image:

- Because it is read-only, you always have a default state to revert to.
- During early bootup, a user-supplied environment (defined next) may be inaccessible or must not be used due to security concerns.
- A user-supplied environment may be inaccessible due to a storage device malfunction or environment data corruption.

You shouldn't keep too much data in this default environment, as it directly adds to the weight of the binary. Keep only critical system variables in this environment.

2) User-Supplied Environment (Flashed in External Storage, Writable): Typically, vendors flash an environment data image to external storage present on your board. The format of this pre-built environment is again the same—that is, a linearized list of strings, but there is a 4-byte CRC header prefixed to it. This CRC is computed over the environment data. Figure 2b illustrates such an environment blob with CRC data, followed by valid environment data and an invalid one after that. The total size of this environment data is fixed to `CONFIG_ENV_SIZE` during compilation. So, if your environment usage exceeds this size, you would need to recompile your U-Boot binary after increasing `CONFIG_ENV_SIZE`. If you do not increase the size, U-Boot will refuse to save the environment variables.

You may decide to keep this environment in external storage, but you must configure the board's config accordingly. U-Boot must know which storage method (and at what offset) will be used to hold the user environment. U-Boot provides a number of options to configure the location of the environment data. U-Boot has the infrastructure to access environment data stored in serial flashes, NVRAM, NAND, dataflash, MMC and even UBI volumes. See the U-Boot documentation for more information on how to use these CONFIG options. Since the default environment size has to be minimized, most of the environment variables are stored here. Certain storage technologies like raw NAND flashes are inherently unreliable. To combat such possibilities (including power failure), and for robustness in general, you also can configure a redundant user environment. You can configure the location and size of this duplicate environment data as well in your board's config.

Saving Environment

Out of the two default environments (default and user), only the user is writable. So, whenever you modify a variable and issue a `saveenv` command, that variable ends up in the user environment.

When you do a `saveenv`, U-Boot does the following:

- Sorts the list of current environment variables.
- Converts them to a linearized list of strings.
- Computes CRC over this data and burns the env back at its fixed location in storage.

Creating a Pre-Built User Environment

U-Boot provides a utility named `mkenvimage` that can be used to generate an environment blob suitable to be flashed. `mkenvimage` needs at least two inputs to create the blob:

1. Environment variables in a text file (only one env "name=value" string on each line).

2. The size of the environment blob in bytes (remember, this must comply with the `CONFIG_ENV_SIZE` you have defined in your board's config).

For example, if my env data file is called my_env_data.txt, and the size of my desired env blob is 16384 (16 KiB), I would use the following command:

```
$./tools/mkenvimage -s 16384 -o env_blob my_env_data.txt
```

You can see the dump of the env blob using the `od` command:

```
$ od -t x1c env_blob

0000000  0d  d2  49  96  62  61  75  64  72  61  74  65  3d  31  31  35
         \r 322   I 226   b   a   u   d   r   a   t   e   =   1   1   5
0000020  32  30  30  00  62  6f  6f  74  64  65  6c  61  79  3d  31  30
          2   0   0  \0   b   o   o   t   d   e   l   a   y   =   1   0
0000040  00  6c  6f  61  64  5f  61  64  64  72  3d  30  78  34  30  30
         \0   l   o   a   d   _   a   d   d   r   =   0   x   4   0   0
0000060  30  30  30  30  30  00  00  00  00  00  00  00  00  00  00  00
          0   0   0   0   0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000100  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
         \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0040000
```

This environment data blob must be flashed at the predefined offset in the storage device. You can use U-Boot, Linux or any other flasher to burn this blob.

Relocation of Environment Data to RAM

During an early boot when U-Boot has not relocated to RAM, it uses the linearized form of environment data (as shown in Figure 1). But once U-Boot has relocated to RAM, this linearized form is no longer used. Instead, U-Boot imports all such env data stored from persistent storage into a RAM-resident hashtable. If the user-supplied environment is good (that is, the CRC is okay), it is imported from Flash to RAM. Otherwise, U-Boot imports the default compiled-in environment to this hashtable. Figure 4 shows how the user environment is imported in to the hashtable, whereas U-Boot along with its default environment relocates to top of the RAM. If the user environment is corrupt or inaccessible, U-Boot would import the default environment in to the hashtable.

The use of the RAM-resident data structure (hashtable) is important for various reasons:

- It boosts performance, as you are manipulating variables in RAM and not in Flash.

- You have to manipulate data only in RAM and need not access some slow Flash driver (and deal with the associated complexity).

- It allows U-Boot to deploy type checks and access control attributes on different variables while still keeping the persistent storage form simple (a linear list).

Once the environment has relocated to RAM (into the hashtable), all commands operating on environment variables will be working only on this hashtable. U-Boot does not touch the environment variables stored in the persistent storage at all (unless it needs to save the env).

Each environment variable entry

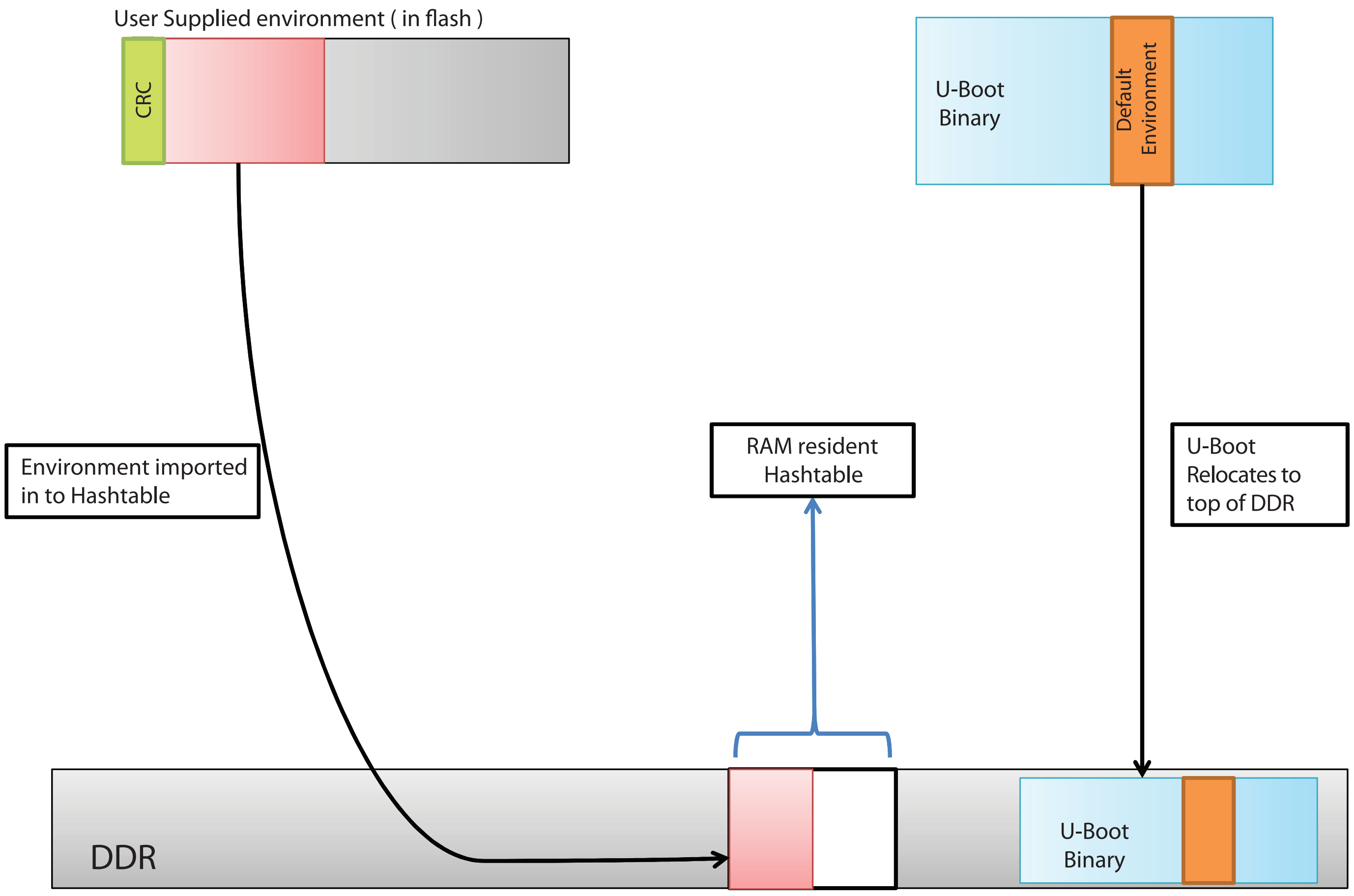
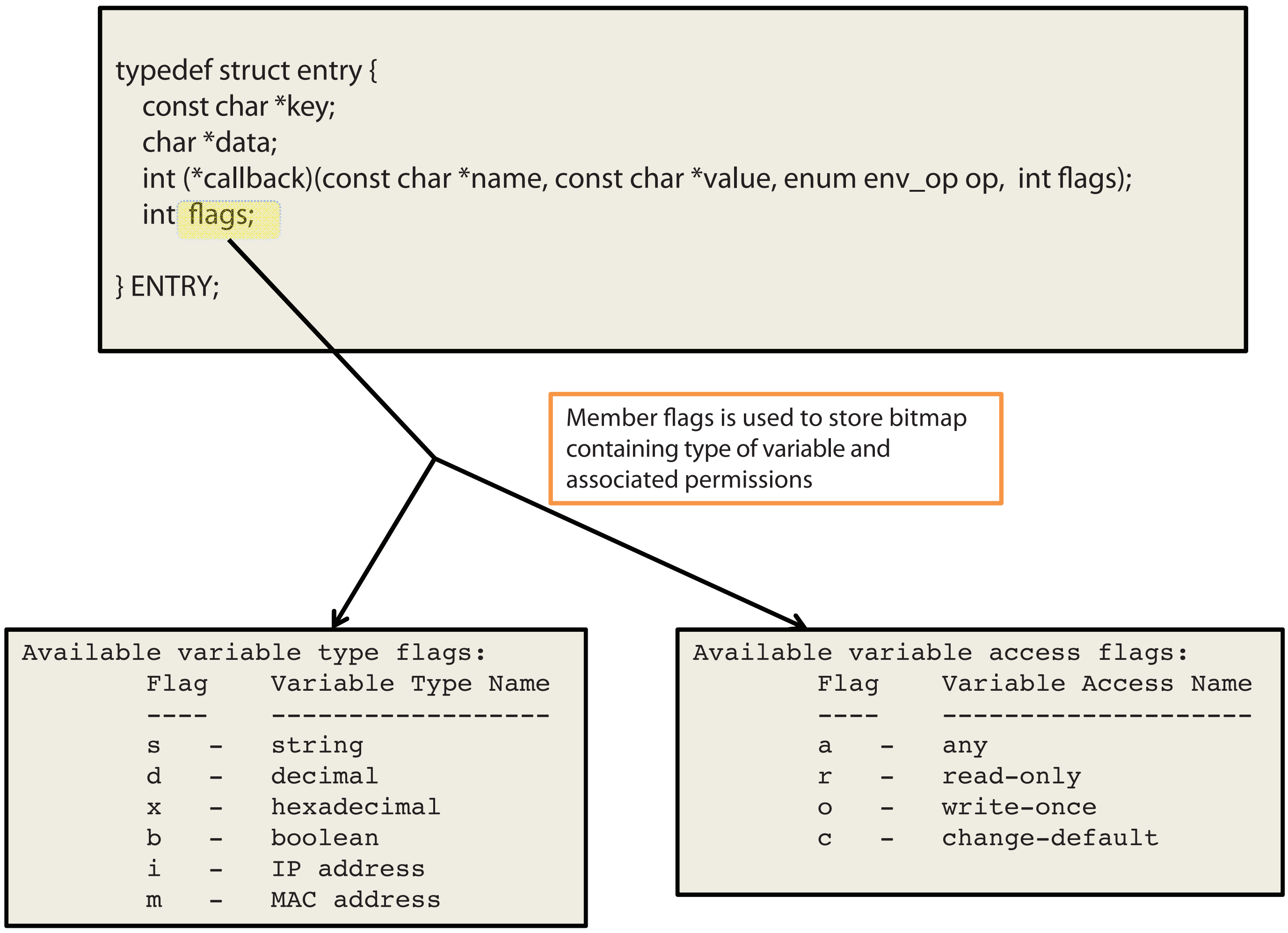

User Supplied environment (in flash)
CRC
U-Boot Binary
Default Environment
Environment imported in to Hashtable
RAM resident Hashtable
U-Boot Relocates to top of DDR
DDR
U-Boot Binary

Figure 4. The user environment is imported into a hashtable. If the user environment is corrupt, the default environment is imported instead.



```
typedef struct entry {
   const char *key;
   char *data;
   int (*callback)(const char *name, const char *value, enum env_op op,  int flags);
   int  flags;

} ENTRY;
```

Member flags is used to store bitmap containing type of variable and associated permissions

```
Available variable type flags:
        Flag    Variable Type Name
        ----    ------------------
        s   -   string
        d   -   decimal
        x   -   hexadecimal
        b   -   boolean
        i   -   IP address
        m   -   MAC address
```

```
Available variable access flags:
        Flag    Variable Access Name
        ----    --------------------
        a   -   any
        r   -   read-only
        o   -   write-once
        c   -   change-default
```

Figure 5. Hashtable representation of an environment variable. There is an associated callback function, as well as a bitmap to store type and access control.

inside the hashtable is represented by a data structure called a `struct entry` (Figure 5). Apart from the members `key` and `data`, which correspond to "name" and "value" in the linearized representation of data, you have members called `callback` and `flags`. `flags` is of integer type and is used to implement type check and access control. `callbacks` is the callback function associated with this environment variable. If defined, this callback handler would be invoked whenever any operation (like add, delete or modify) is performed on this environment variable.

How to Control/React to Environment Variable Modification?

At times you may want to do your own runtime configurations. You may want to react (accept, reject or produce some side effects) to the changes done to some environment variable. For such use cases, U-Boot provides a mechanism of deploying callback handlers. You can associate a callback function with an environment variable. As a first step, you have to register a callback handler with U-Boot. This function would be called whenever you do any modifications to the environment variable. Figure 6 shows sample code to register a callback with U-Boot. You can place such handler code in your board-specific file. The macro `U_BOOT_ENV_CALLBACK` registers the callback function `on_change_foo` with the handler named `foo_h`. Your handler is now registered with U-Boot with the name "foo".

Now, you need to establish a link to this registered handler with an environment variable. Take a look at the struct entry in Figure 6. You can see that there is a member named `callback` (a function pointer) for each environment variable. U-Boot would invoke this handler before committing the modified environment variable to the hashtable. You can make this association of callback handler with the environment variable either at compile time or at runtime. For compile-time association, you need to define the config option `#define CONFIG_ENV_CALLBACK_LIST_DEFAULT foo:foo_h` in your board's config file.

You also can do runtime association as depicted in Figure 7. Here I have

```
#include <env_flags.h>
#include <environment.h>

static int on_change_foo(const char *name, const char *value, enum env_op op, int flags)
{
    printf("callback_handler on_change_foo() invoked for variable : %s\n", name);
    switch (op) {
    case env_op_create:
    case env_op_overwrite:
    printf("foo variable modified..\n");
    break;
    case env_op_delete:
    printf("foo variable deleted...\n");
    break;
    default:
    break;
 }

 return 0;
}
U_BOOT_ENV_CALLBACK(foo_h, on_change_foo);
```

Figure 6. Code a handler you want to call for an environment variable.

```
Deepthought> setenv foo bar
Deepthought> setenv .callbacks foo:foo_h
Deepthought> setenv foo bar
callback_handler on_change_foo() invoked for variable : foo
foo variable modified..
Deepthought>
```

Figure 7. Associate the callback handler with the environment variable by setting an environment variable `.callbacks`. Here, any modification of foo would invoke `on_change_foo()`.

created a new environment variable `.callbacks`, which is a standard U-Boot system variable to make such associations. I have deployed a handler named `foo_h` for the environment variable `foo`. Once this registration is done, whenever you do some modification to variable `foo`, the function `on_change_foo()` would be invoked. You now can deliver your reaction to different types of actions (`env_op_create`, `env_op_overwrite` or `env_op_delete`).

U-Boot already deploys similar handlers for managing console changes, splash images and so on.

Type and Access Control of Environment Variables

There are certain environment variables that you want to use but do not want to do casual modifications. For example, say you have an environment variable `serial#`; you definitely want this variable to be read-only, and you want automatic rejection of any attempt to change it. Another such example is the MAC address of the device. Again, you want to keep that variable as read-only or, at worst, write-once. U-Boot supports different access modifiers: `any`, `read-only`, `write-once` and `change-default` (Figure 5). The U-Boot hashtable representation of environment variables has a member `int flags` (Figure 5). The member `flags` is used to keep a bitmap specifying the access permission associated with variable. So, whenever any modification attempt is done on variable, it must comply with the access permission; otherwise, U-Boot will reject the changes.

Another problem faced by users is the basic sanity check of environment variable type. Since the linearized form of environment keeps only strings, U-Boot needs to make sure that it can do some kind of type check before assigning a value to a variable. To address this issue, U-Boot makes use of some predefined types, such as "string",

```
Deepthought> setenv foo bar
Deepthought> setenv .flags foo:sr
Deepthought> setenv foo bar
## Error: Can't overwrite "foo"
## Error inserting "foo" variable, errno=1
Deepthought>
```

Figure 8. Each environment variable can be associated with a type and access permission in the environment variable `.flags`.

"decimal", "hexadecimal", "boolean", "IP address" and "MAC address". There are corresponding codes for these type modifiers: "s", "d", "x", "b", "i" and "m". Again, U-Boot stores this type information of variables in `flags` as a bitmap (Figure 5).

You can associate "type" and "access control" to a variable either at compile time or at runtime. For compile-time association, you need to define a config `#define CONFIG_ENV_FLAGS_LIST_DEFAULT foo:sr` in your board file. For runtime association, you can define an environment variable `.flags` (Figure 8). Here, I am associating an environment variable `foo` with type `s` (meaning the value is a string) and access control `r` (meaning it is read-only). Once deployed, if you try to modify the variable `foo`, U-Boot will reject your request. Also, if a value is not of specified "type", your update to the environment variable will fail.

Environment variables like MAC address make use of type `m`. This will make U-Boot do a sanity check on the value entered by the user to confirm whether the value is indeed a valid MAC address.

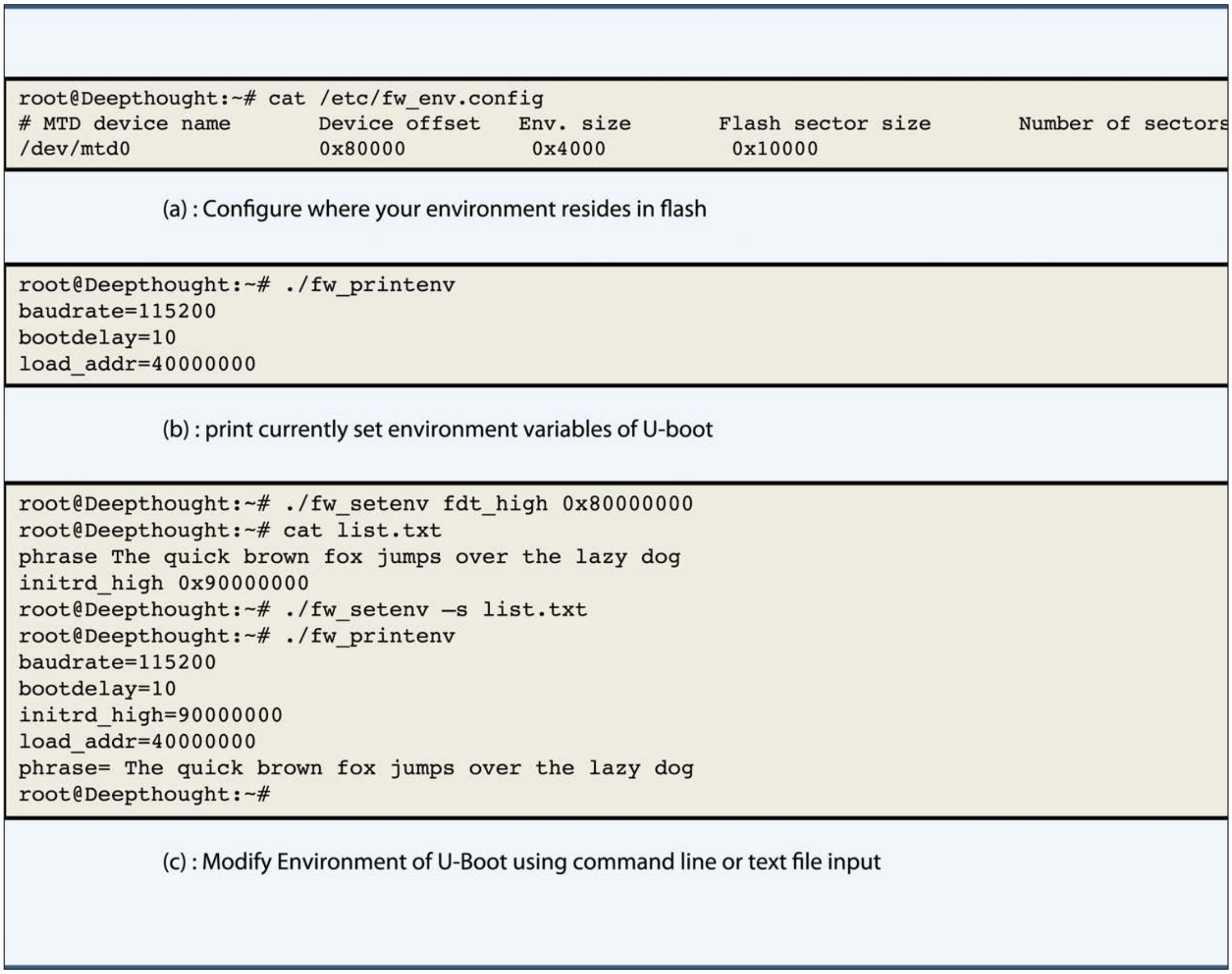
Modifying the U-Boot User Environment from Linux

U-Boot environment variables can be added, modified or deleted from Linux as well. U-Boot provides a set of utilities called `fw_printenv` and `fw_setenv` to do the job. First, you need to compile these utilities for Linux. Figure 9 shows the compilation steps for the utility. Here, I am cross-compiling it for the ARM platform. It is a multi-call binary. So, you need to make a symlink named `fw_setenv` to the binary `fw_printenv`.

To modify the environment, you first need to boot in to Linux on the target board. Next, you need to create a file called /etc/fw_env.config. This file contains all the information needed to specify the location of

```
make <your_board>_config
make HOSTCC=armv7-linux-gcc HOSTSTRIP=armv7-linux-strip tools-all
```

Figure 9. Compiling fw_printenv/fw_setenv for an armv7 Host



```
root@Deepthought:~# cat /etc/fw_env.config
# MTD device name        Device offset    Env. size       Flash sector size       Number of sectors
/dev/mtd0                0x80000           0x4000          0x10000
```

(a) : Configure where your environment resides in flash

```
root@Deepthought:~# ./fw_printenv
baudrate=115200
bootdelay=10
load_addr=40000000
```

(b) : print currently set environment variables of U-boot

```
root@Deepthought:~# ./fw_setenv fdt_high 0x80000000
root@Deepthought:~# cat list.txt
phrase The quick brown fox jumps over the lazy dog
initrd_high 0x90000000
root@Deepthought:~# ./fw_setenv –s list.txt
root@Deepthought:~# ./fw_printenv
baudrate=115200
bootdelay=10
initrd_high=90000000
load_addr=40000000
phrase= The quick brown fox jumps over the lazy dog
root@Deepthought:~#
```

(c) : Modify Environment of U-Boot using command line or text file input

Figure 10. My Configuration File

the environment data blob. Figure 10 shows my configuration file. I kept my environment in an SPI Flash, which appeared as /dev/mtd0 to my kernel. My environment blob was configured at an offset of 0x80000 from the beginning of Flash and had a size of 0x40000. The size of each sector of my Flash is 0x10000. This is all the information I needed to provide in order for the environment manipulation utilities to work.

As soon as I keyed in the command `fw_printenv`, I could see the variables that I saved in the U-Boot user environment appearing on my console.

You also can set the environment variables using `fw_setenv`. As shown in Figure 10, I make use of a text file (list.txt) containing the variables I want to set. The format is simple. The first whitespace after a name acts as a delimiter, and the characters until the end of line thereafter are considered the value for the key.

You can verify that the variables have been set by executing `fw_printenv`. These variables now would be visible from U-Boot as well.

Restoring the Default Environment

Sometimes after a lot of environment variable changes, you can corrupt the state. To restore sanity and get the original values of the default environment, you can use the `env` command:

```
env default [-f] var [...]
```

The above command would forcibly reset the specified variables to a value from the default environment.

To restore the complete environment from the default, invoke the following command:

```
env default -a
```

The `env` command is very powerful; you can use it import/export environment data from/to RAM.

Final Comments

The U-Boot environment can act as a very useful runtime configuration tool. When combined with scripting, it can make the arduous task of development and testing boot scenarios much simpler and more fun to do. ■

Sachin Verma is a Linux kernel Engineer with STMicroelectronics Pvt Limited. His interest areas include the Linux kernel, virtualization and multicore computing. He can be reached at simplysachin@gmail.com.

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

Resources

U-Boot Source Code: **http://git.denx.de/u-boot.git**

Das U-Boot Development Wiki: **http://www.denx.de/wiki/U-Boot**

Accessing the I/O Ports of the BeagleBone Black with Python

The BeagleBone Black is a wonderful little piece of hardware. You could use it to send your next rocket to Mars with just a few lines of Python.

SAMUEL BUCQUET

The BeagleBone Black (BBB) is a low-cost, low-power, credit-card size (3.4" x 2.1") board with a lot of features, and it costs about $60. It sports a 1GHz Sitara AM335x ARM Cortex-A8 (an ARMv7) processor from Texas Instruments, 512MB of RAM, and it has all the I/O capabilities you'd expect from a typical microcontroller, such as access to a CAN bus, SPI interface and i2c, analog input, PWM and so on.

But, the board also holds two PRUs, an HDMI video output, an SD card slot and 100Mb Ethernet. This makes the board a complete ARM PC, fully compatible with Linux. As icing on the cake, Beagle fancies an open hardware philosophy: all of the chips and designs are available to the public.

Right out of the box, you can use it for the following:

- A great learning platform with easy access to the connected hardware. You can play with almost all the functionalities from the Web interface with the Bonescript language. Just plug in the board via the USB client on a PC, open the page of the board (http://192.168.7.2) and voilà! (See **http://beagleboard.org/getting-started**.)

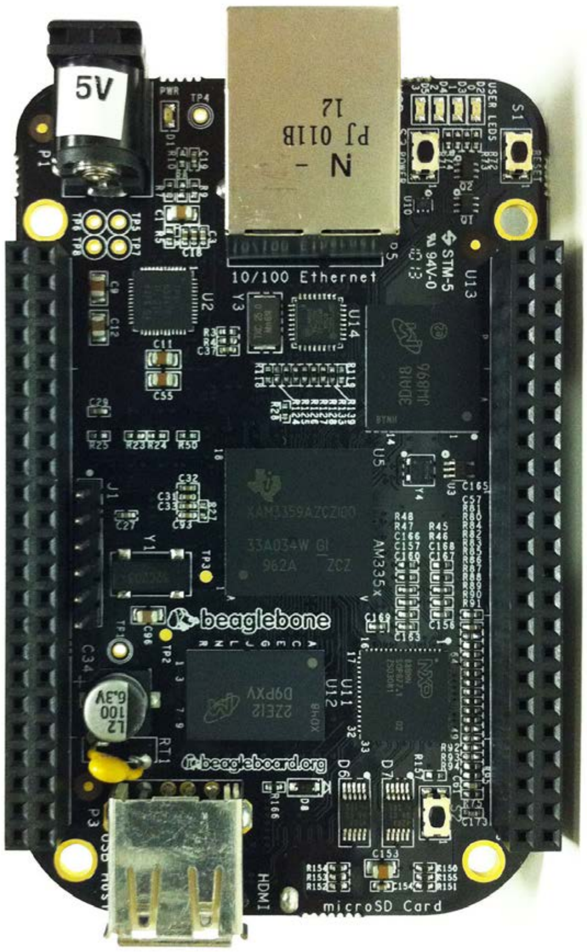


Figure 1. The BeagleBone Black board—isn't she beautiful?

- A light desktop system if you add a 5VDCC external power supply, an HDMI cable, a screen, keyboard and mouse.

This article focuses on working on a BBB from a Debian system with Python and some minimalism in mind. But, this is still a fully-fledged Linux system, not an Arduino or microcontroller.

In this article, I describe how to

There is no X session, no fancy Web interface, just the good-old command line via SSH, and the applications are launched automatically at boot via /etc/rc.local.

access some of the I/O ports:

- The serial ports, to read and write on devices with an RS232 interface like a GPS, for example.
- The GPIOs, which allow you to trap or send TTL signals, drive a relay, read a button status, and in particular, let you add a PPS to Linux.
- The analog input voltage for reading voltages coming from a lot of sensors.
- Components on the i2c bus: an RTC handled by the system and a DAC driven from your applications.

I finish the article explaining how to use the BBB as a time server, thanks to a GPS.

What Do We Use It For?

In our project, the BBB boards are embedded in small compartments, accessible only through a network connection. I chose to configure them with the bare minimum, from the eLinux version. There is no X session, no fancy Web interface, just the good-old command line via SSH, and the applications are launched automatically at boot via /etc/rc.local. They are used in a Unmanned Surface Vehicle (USV), and the interactions with the operator are done through a hardware control panel via a serial line, through Web interfaces and with networked applications. The interactions with the hardware of the USV are done through its many I/O ports, of course.

Why Python?

Do I really need to tell you? Okay, we have enough processing power, so Python equals less code and more readability of the applications, and a lot of useful modules already are available. All code and examples in this article are for Python 2.7, but it would be not very difficult to port it to Python 3.4.

Danger: Important Usage Precautions

You need to take several precautions when working the expansion headers to prevent damage to the board:

- All voltage levels are 3.3V max. *Application of 5V to any I/O pin will damage the processor and void the warranty.*
- Analog in voltages are 1.8V max. *Application of >1.8V to any A/D pin will damage the processor and void the warranty.*
- Do not apply any voltages to any I/O pins when the board is not powered on.
- Do not drive any external signals into the I/O pins until after the SYS_RESETn signal is HI (3.3V).
- Do not apply any voltages that are generated from external sources until the SYS_RESETn signal is HI.
- If voltages are generated from the VDD_5V signal, those supplies must not become active until after the SYS_RESETn signal is HI.
- If you are applying signals from other boards into the expansion headers, make sure you power up the board after you power up the BeagleBone Black or make the connections after power is applied on both boards.
- Powering the processor via its I/O pins can cause damage to the processor.

Pimp My BBB

The Debian system: When we received our BBB boards in September 2013, they were pre-installed with the Angstrom distribution. Thanks to the work of others who were also attached to Debian, I quickly was able to keep playing with my favorite Linux system on my fresh BBBs.

I installed the images provided by eLinux, but you also can fetch an image from armhf: **http://www.armhf.com/boards/beaglebone-black/#wheezy**.

Since March 2014, Debian Wheezy (stable) is an official system image available for the BeagleBone Black (rev B and C). For the latest images, see **http://beagleboard.org/latest-images**.

You can choose to upgrade to testing (or sid if you feel more adventurous) in order to enjoy more recent software. (See the Upgrading from Debian Stable to Debian Testing sidebar.)

Upgrading from Debian Stable to Debian Testing

First, update your system with `apt-get update && apt-get upgrade`.

Next, modify your /etc/apt/sources.list.d/debian.list file. Copy the lines with `wheezy` or `stable`, and replace all occurrences of `wheezy` with `jessie` on the copied lines. You can choose `testing` instead of `jessie` if you want to keep on with the testing release after jessie was made stable.

Then launch `apt-get update && apt-get upgrade` again, and if all is well (it might take a long time depending on your connection quality and the packages already installed), run `apt-get dist-upgrade`.

The BBB accepts booting from the internal memory, the eMMC or from an external SD. (See the Booting the BBB from an SD Card sidebar.)

To test another version of the system, simply download and write it on your SD. If you are satisfied with it, you have the option to put it on the eMMC.

As the environment hosting our BBBs is subject to strong vibrations, I chose to put my system in the eMMC rather than on an SD.

Flash the eMMC: In order to flash your new system to your eMMC, download the flasher version from eLinux or the official one. Write it to your SD and boot your BBB from the SD. The flashing process happens automagically. You will have to wait less than ten minutes before the four blue LEDs become steady, indicating that the flashing is over. As the official firmware is much larger, the flashing will take a lot longer (45 minutes).

Danger: you need to power the board with an external 5VDC power supply when flashing!

In order to use the armhf version,

Booting the BBB from an SD Card

Power off the board, then with the SD card inserted, hold down the S2 button near the SD slot, apply power, and continue to hold down the button until the first LED comes on.

If an NTP server is available to your BBB, good for you, but as the BBB lacks a backed battery RTC, it doesn't retain date and time after reboot, so you will have to take a few measures.

partition and format your SD card following the armhf site instructions at **http://www.armhf.com/boards/beaglebone-black/bbb-sd-install**. You then can download a recent armhf rootfs archive (**http://s3.armhf.com/dist/bone/debian-wheezy-7.5-rootfs-3.14.4.1-bone-armhf.com.tar.xz**) and copy it to your SD. Then, when booting from the SD, you likewise can copy your SD installation to your eMMC.

As the time of this writing (July 2014), these Debian images come with Linux kernel 3.8.13. This version brought many improvements to accessing the BBB hardware by the kernel via sysfs.

Here is a list of packages I recommend for working with the board from the shell and from Python:

```
# apt-get install kbd locales htop vim screen \
rsync build-essential git python-setuptools \
cython python-pip python-virtualenv python-dev \
manpages-{dev,posix{,-dev}} glibc-doc- \
reference python-serial python-smbus python- \
lxml python-psutil i2c-tools
```

For interfacing with a GPS with a PPS:

```
# apt-get install gpsd python-gps pps-tools \
bison flex git-core
```

If you want to play with NTPd:

```
# apt-get install ntp
```

Configure the System

What Time Is It? If an NTP server is available to your BBB, good for you, but as the BBB lacks a backed battery RTC, it doesn't retain date and time after reboot, so you will have to take a few measures.

First, enter the date in UTC manually, *before* anything else:

```
# date -u 072314512014.30
Wed Jul 23 14:51:30 UTC 2014
```

If your BBB must be isolated from an NTP server, one solution is to add an RTC to the board, like a ds1307. (I will show how to add one on the i2c bus.)

Finally, if you are isolated and without an RTC module, try the fake-hwclock package from the Debian repositories. It will allow your clock to restart with the last date saved when the system halted.

Configuring the Network Card Static

Edit the /etc/network/interfaces file and change the line reading:

```
iface eth0 inet dhcp
```

to:

```
iface eth0 inet static
    address 192.168.1.101
    netmask 255.255.255.0
    broadcast 192.168.1.255
    gateway 192.168.1.254
```

Then restart networking with:

```
/etc/init.d/networking stop
/etc/init.d/networking start
```

Note: if you are editing the file via SSH, you will lose your connection right now! You should do it with a keyboard and display or the serial access instead.

To DHCP or Not to DHCP: If your network hosts a DHCP server, you are fine; otherwise, you can configure your network card "static" in order to avoid a big DHCP timeout when you boot your BBB with the Ethernet cable plugged in. (See the Configuring the Network Card Static sidebar.)

A Life Line (Serial Debug): More often than not, the boards are in a place where we can't have a keyboard and display attached to them. We can work remotely by SSH, but if something goes wrong, we need to access the serial debug interface on the board.

The serial interface available through the USB connection to the board is not ready when you boot with U-Boot—you can't see the kernel starting or intervene. That's why we use the serial debug provided by the J1 connector on the board, referred to as ttyO0 by the system.

As a side note, this serial line can be made available via Ethernet with a cheap RS232IP converter if remote boot monitoring is needed.

Before connecting our BBB on a PC via this serial line, we need a TTLRS232 converter. See some serial debug references on eLinux at **http://elinux.org/Beagleboard:BeagleBone_Black_Serial**.

You can purchase a PL2303HX USB to RS232 TTL auto converter module—they are very cheap. Just make sure the end of the cable is made of jumper wires and not a fixed connector (Figure 2).

This Code Chief's Space page provides a very thorough step-by-step guide: **http://codechief.wordpress.com/2013/11/11/beaglebone-black-serial-debug-connection**.

Figure 2. An Essential Accessory—PL2303HX USB to RS232 TTL Converter

Serial Login into the System: You probably are familiar with the text consoles with `Login:` that you make appear with Ctrl-Alt-F{1..6}. We wanted the same, but through our serial debug. So, we configured a tty on the BBB allowing us to connect to it via the serial line when it is booted. Just add this line (if it's not already present) to the end of /etc/inittab:

```
T0:23:respawn:/sbin/getty -L ttyO0 115200 vt102
```

And, make sure the kernel knows the correct console on which to output its messages. In the /boot/uboot/uEnv.txt file, it should read:

```
console=ttyO0,115200n8
#console=tty
```

Finally, to connect us to the BBB through our serial debug line from another PC with a USBRS232 adapter:

```
$ screen /dev/ttyUSB0 115200
```

Access the Input/Output Ports

The Serial Ports: As you previously saw, the serial port must be accessed with a TTL/RS232 adapter. In our project, we re-used some old Maxim MAX3232s to do it, and it works, of course,

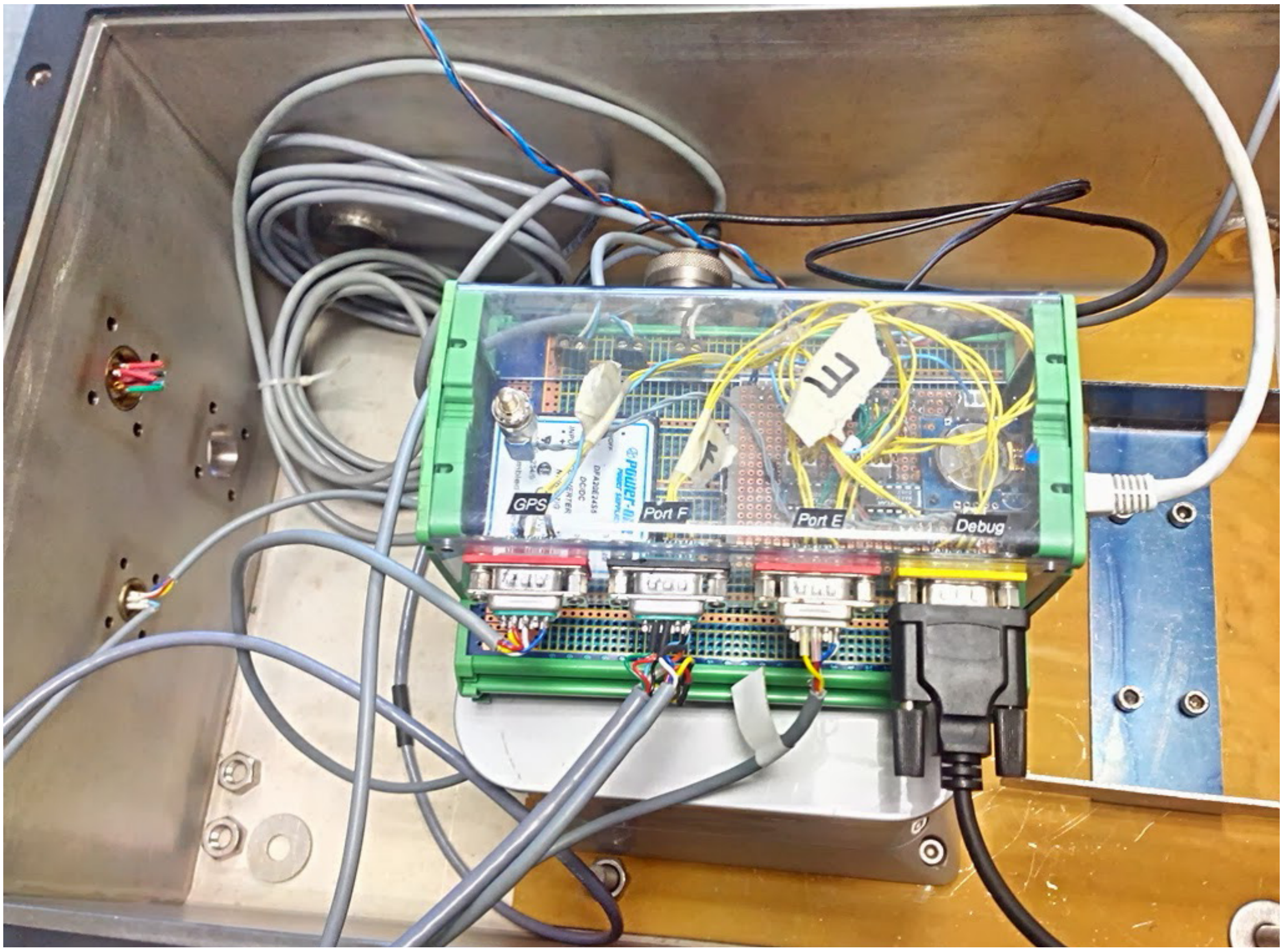


Figure 3. The Board "bbb02" in situ with three serial ports and the serial debug port, a temperature sensor on AIN0, a water detector on GPIO_48, a PPS input on GPIO_49 and a POWER_RESET button.

for the other UARTs available on the board.

The BBB comes with six UARTs, but three of them can't be addressed by our applications:

- UART0 is the serial debug on J1.
- UART3 lacks an RX line.
- UART5 can't be used altogether with the HDMI output.

So UART1, UART2 and UART4 are the ones available, and they are addressed, respectively, as /dev/ttyO1, /dev/ttyO2 and /dev/ttyO4. Note that only UART1 and UART4 have CTS and RTS pins.

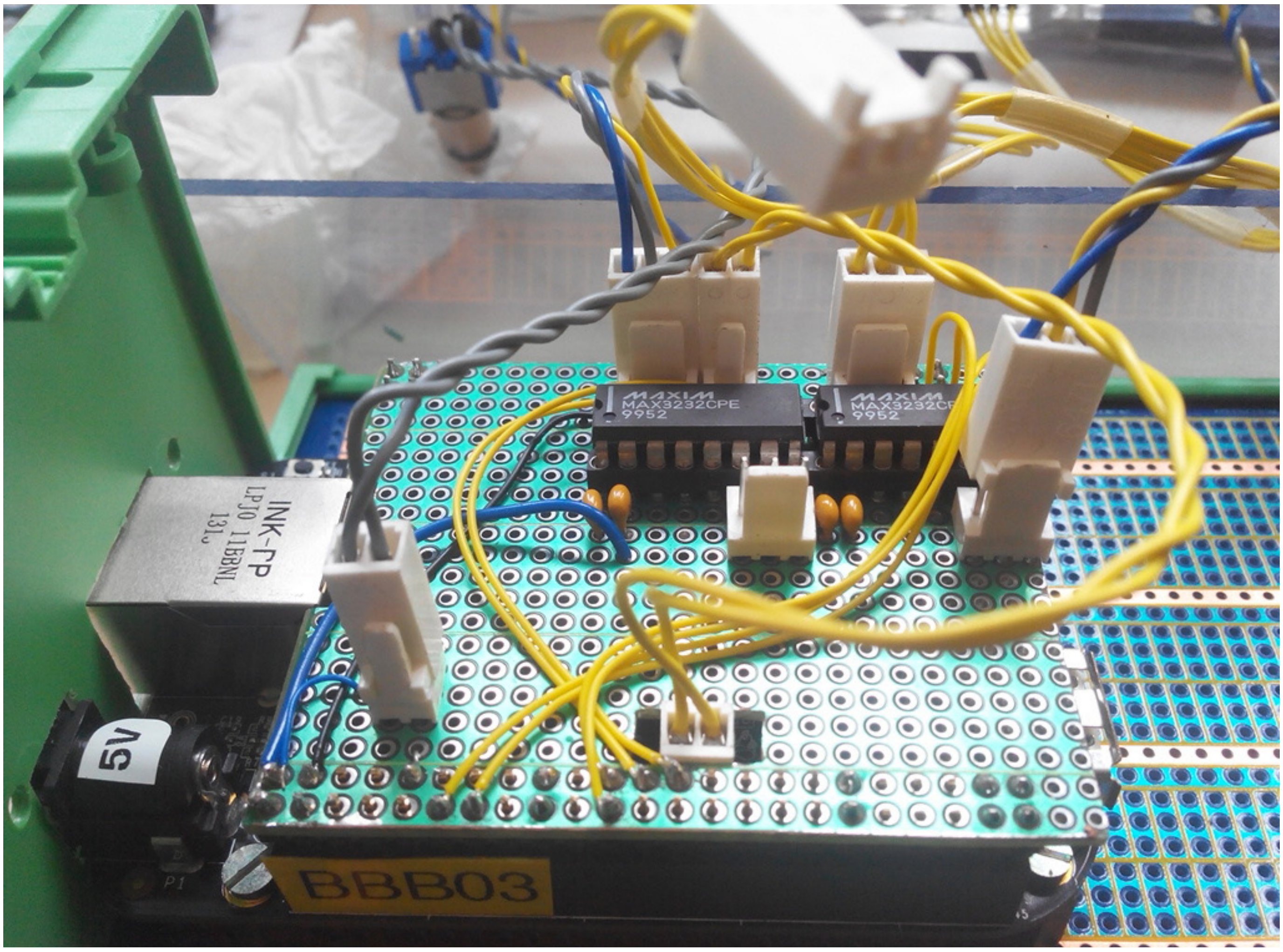


Figure 4. Our CAPE with Two MAX3232s

Load the CAPE files for each UART:

```
for i in {1,2,4}
do
  echo BB-UART$i > /sys/devices/bone_capemgr.*/ \
slots
done
```

The output of `dmesg` should show a correct initialization:

```
# dmesg |grep tty
...
[    1.541819] console [ttyO0] enabled
[  286.489374] 48022000.serial: ttyO1 at MMIO \
0x48022000 (irq = 89) is a OMAP UART1
[  286.627996] 48024000.serial: ttyO2 at MMIO \
0x48024000 (irq = 90) is a OMAP UART2
[  286.768652] 481a8000.serial: ttyO4 at MMIO \
0x481a8000 (irq = 61) is a OMAP UART4
```

Passing Arguments at Boot Time

To give command-line options to the kernel, modify the `optargs` variable in the /boot/uboot/uEnv.txt file. Only one `optargs` line is allowed, so if you need several options passed to the kernel, just add them separated by a space like this:

```
optargs=quiet fixrtc \
capemgr.enable_partno=BB-ADC,BB-UART1,\
BB-UART2,BB-UART4
```

Now you can ask the kernel to load them at boot time. Edit /boot/uboot/uEnv.txt and modify the line with `optargs`, like this:

```
optargs=capemgr.enable_partno=BB-UART1, \
BB-UART2,BB-UART4
```

(See the Passing Arguments at Boot Time sidebar.)

Okay, but Does It Work? To test the serial lines, we just have to connect two of them together (Figure 5).

Now, launch a screen session

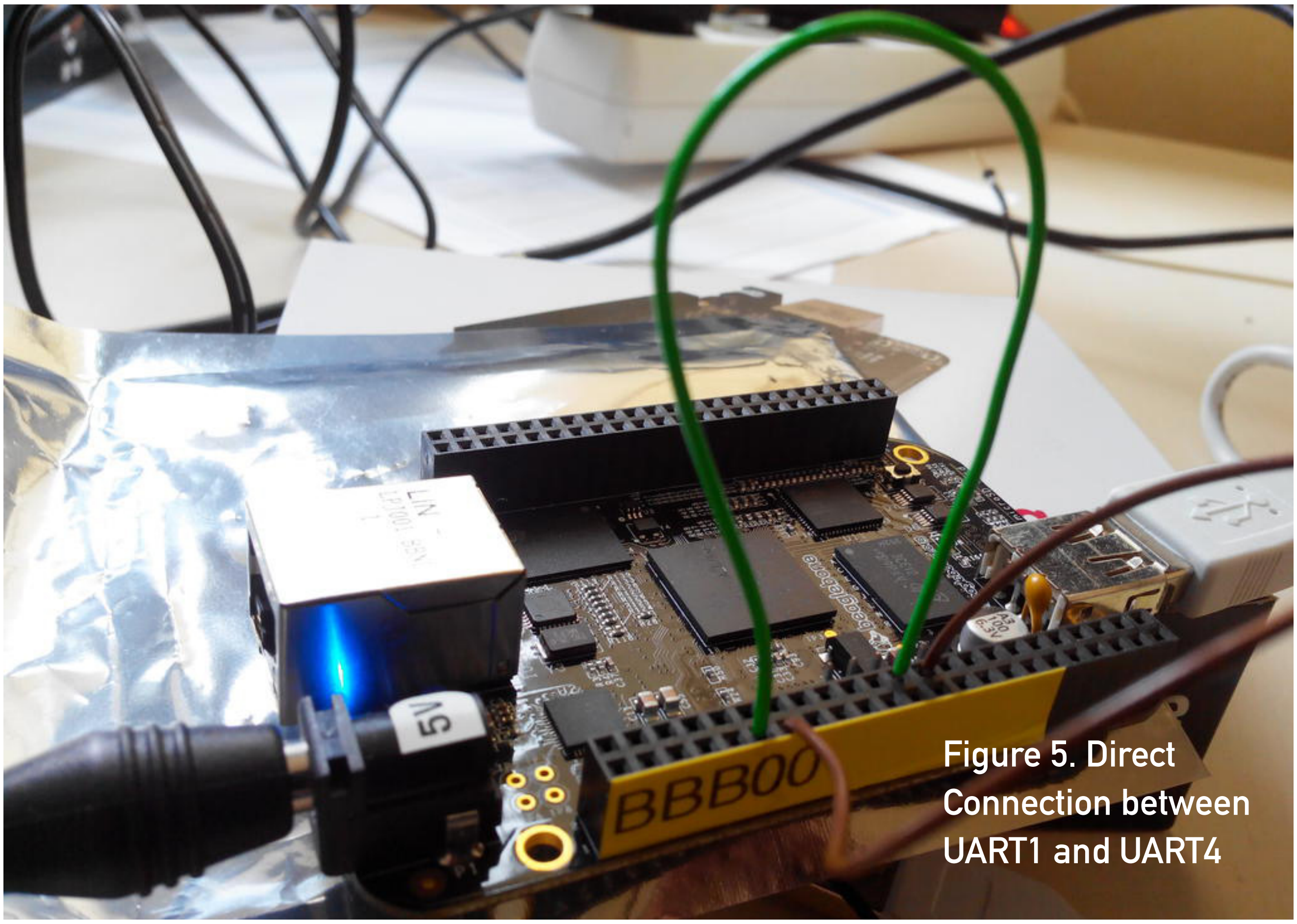


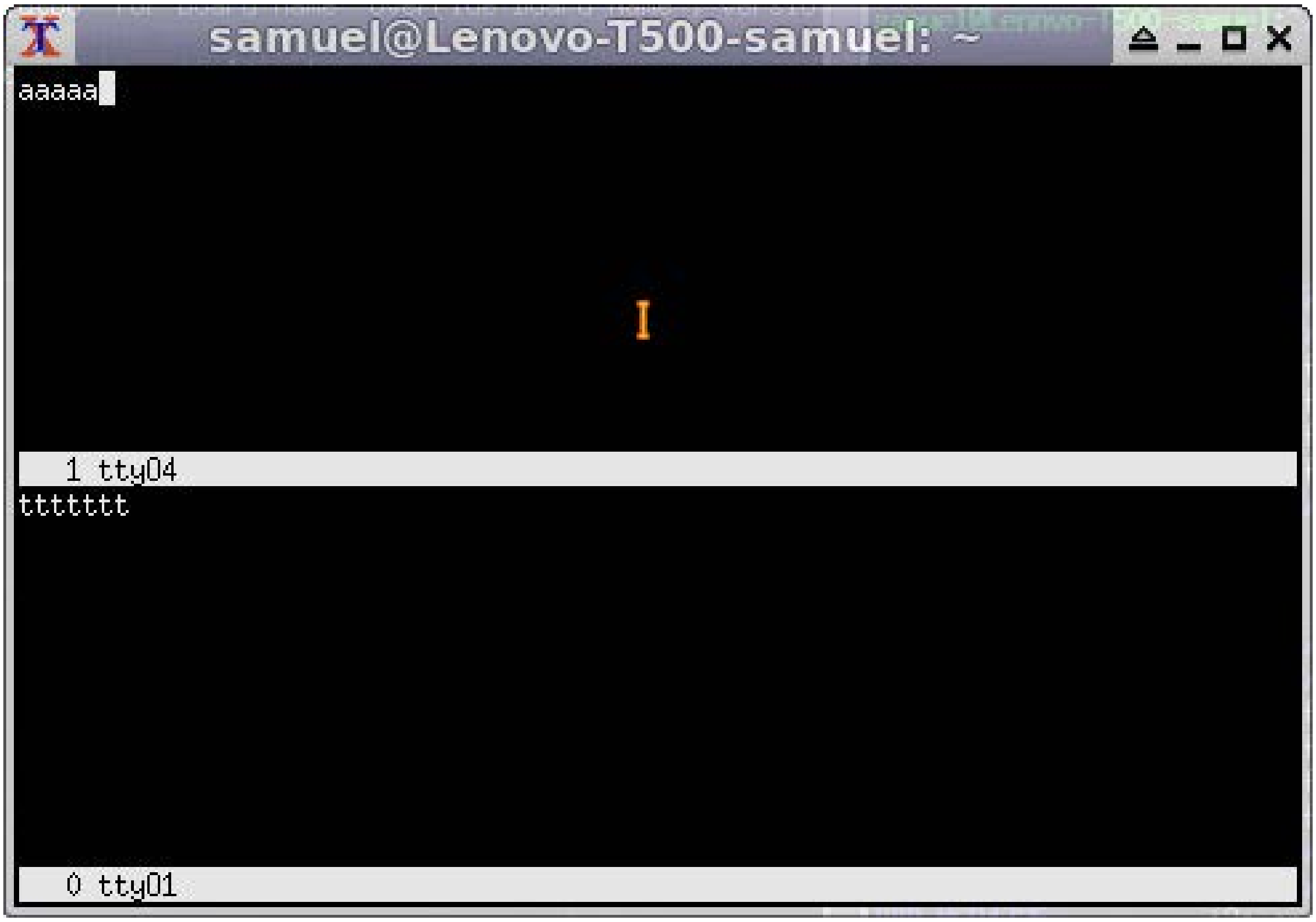
Figure 5. Direct Connection between UART1 and UART4

Table 1. Direct Connection of UART1 with UART4

UART4_Tx	PIN __13__ of P9	→	PIN __26__ of P9	UART1_Rx
UART4_Rx	PIN __11__ of P9	→	PIN __24__ of P9	UART1_Tx

on UART1 with `screen /dev/ttyO1 115200`, split your window with a Ctrl-A S, move to the next window with Ctrl-Tab, open a screen on UART4 with Ctrl-A :, then type `screen /dev/ttyO4 115200`, and check that what you type in one window appears in the other (Figure 6).

From now on, you can use the `python-serial` module to read and write on your serial ports,



```
aaaaa

  1 tty04
ttttttt

  0 tty01
```

Figure 6. ttyO1 and ttyO4 Screen Session like this:

```
import time
from serial import Serial

ctrl_panel = Serial(port=comport, baudrate=9600,
bytesize=8, parity='N', stopbits=1, timeout=0.25 )

# send a commande
ctrl_panel.write(pupitre_commande)

# read an answer
buff = ctrl_panel.read(answer_size)
# as the serial port is configured non-blocking
# with 'timeout=0.25', we make sure all the
# bytes asked for are received.
while len(buff) < answer_size:
  n = len(buff)
  b = ctrl_panel.read(answer_size-n)
  buff += b
  print "------",n,"------"
  time.sleep(0.02)
print "Serial IN: ",
print ' '.join(['%02X' % ord(c) for c in buff])
```

The `python-serial` module is quite efficient. We managed to read two serial ports refreshed at 50Hz plus a third at 5Hz (on the same board and simultaneously) with a few glitches between the timestamps, and the CPU load stayed below 50%.

The GPIO: The General-Purpose Input or Output pins allow the board to receive a signal, read a frequency on input or drive a relay on output.

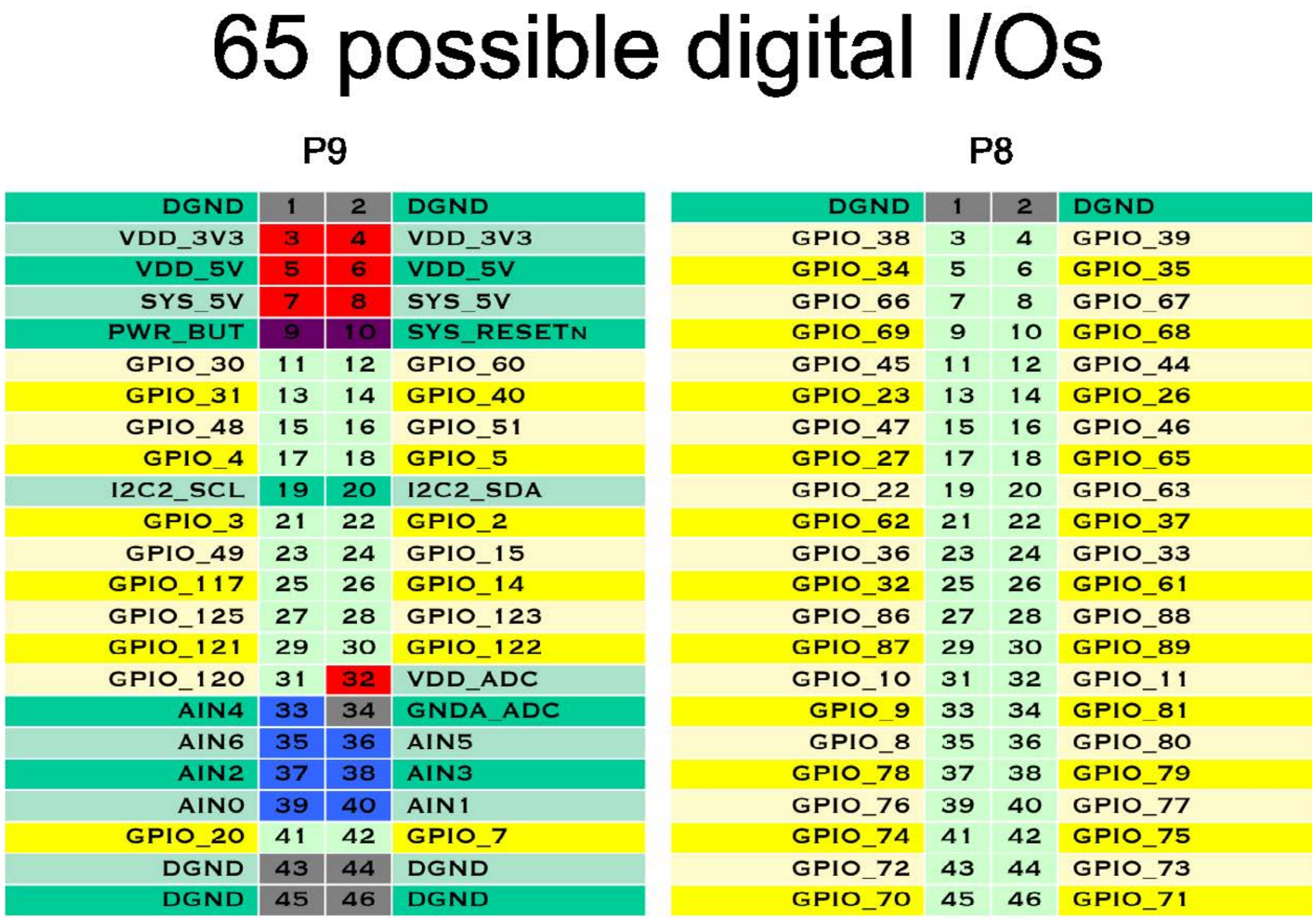
First we had to understand the mapping between the pin on the P8 and P9 connectors and the GPIO numbers as seen by the kernel. The BBB System Reference Manual gives the Expansion Header P9 pinout (**http://elinux.org/Beagleboard:BeagleBoneBlack#Hardware_Files**).

For each GPIO, the name is made up of the GPIO controller number between 0 and 3 and the pin number on the Sitara AM3359AZ. The kernel numbers the GPIO pin with PIN + (GPIO controller number * 32). If we pick the GPIO1_16 on the pin 15 of the P9 connector, the kernel will see it as GPIO_48 (16+(1 * 32)). See Figure 7 to read the direct mapping for all the GPIOs.

And, you can find a similar table for each kind of I/O port of the BBB on this Cape Expansion Headers page: **http://elinux.org/Beagleboard:Cape_Expansion_Headers**

Operate the GPIO: The gpio_sysfs.txt file, provided with your kernel documentation, explains how to work with GPIOs with a Linux kernel (**https://www.kernel.org/doc/Documentation/gpio/sysfs.txt**).

These steps are for Linux kernel version 3.8.13. Each GPIO must be



65 possible digital I/Os

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_40
GPIO_48	15	16	GPIO_51
GPIO_4	17	18	GPIO_5
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_125	27	28	GPIO_123
GPIO_121	29	30	GPIO_122
GPIO_120	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

Figure 7. Mapping GPIO Kernel Numbers with P8 and P9 Pinouts

enabled independently, so for the GPIO_48 on P9_15:

```
# echo 48 > /sys/class/gpio/export
```

Next, choose the way it will operate. If you want to process input signals:

```
# echo in > /sys/class/gpio/gpio48/direction
```

Then choose if you want to detect a "rising" or "falling" edge or "both":

```
# echo both > /sys/class/gpio/gpio48/edge
```

If you want to output signals, you can choose "out" or one of "high" or "low" (telling when the signal is active):

```
# echo high > /sys/class/gpio/gpio48/direction
```

And to stop emitting the signal:

```
# echo 0 > /sys/class/gpio/gpio48/value
```

At the end, release the GPIO:

```
# echo 48 > /sys/class/gpio/unexport
```

Wait for a Signal on a GPIO from Python: We connected an optical water detector to one of our BBBs inside an enclosure, on the GPIO_48 (P9_15) (Figure 3). The water detector sends a TTL signal if there is water passing through its lens. Here is how we wait for an event describing water presence from Python using the BBB_GPIO class from bbb_gpio.py:

```
from bbb_gpio import BBB_GPIO

water = BBB_GPIO(48,gpio_edge='both',\
    active_low=True)
for value in water:
    if value:
        print "Water detected !"
    else:
        print "No more water !"
```

The BBB_GPIO class is a generator. For each iteration, we wait for a change on the GPIO and return in `value` the GPIO status. When the GPIO is waiting for an event, we don't want to be polling aggressively on the system, but to be awakened only when the event occurs. That's what the `poll()` system call does. (See bbb_gpio.py line 58—there is a link to my GitHub page with all the code for this article in the Resources section.)

A Special Case, the PPS Signal: A GPS often delivers a PPS (Pulse Per Second) signal in order to synchronize with good accuracy the timing NMEA sentences, like $GPZDA. The PPS signal is a TTL we can connect to a GPIO; we chose GPIO_49 (P9_23). Once wired, we check whether the signal is present and can be read:

```
# echo 49 > /sys/class/gpio/export
# echo in > /sys/class/gpio/gpio49/direction
# echo rising /sys/class/gpio/gpio49/edge
# cat /sys/class/gpio/gpio49/value
1
```

Be careful as to the output voltage of the PPS, as the GPIOs of the BBB accept a TTL of 3.3V max.

The PPS signal is also a special input understood by the Linux kernel. In order to enable our GPIO input as a PPS, we have to compile a dts (Device Tree Source file) into a dtbo (Device Tree Binary Object file) with the dtc (Device Tree Compiler) tool (see the GPS-PPS-P9_23-00A0.dts file on my GitHub page):

```
# ./dtc -O dtb -o GPS-PPS-P9_23-00A0.dtbo -b 0 \
-@ GPS-PPS-P9_23-00A0.dts
```

Fetching a Good dtc

This one is tricky. One year ago, I successfully downloaded and patched dtc, but now the patch is not synchronized with the versions of dtc I can find. Thanks to Robert Nelson (https://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-Upgradedistro%22device-tree-compiler%22package), you just have to download and execute his version:

```
# wget -c https://raw.github.com/RobertCNelson/\
tools/master/pkgs/dtc.sh
# chmod +x dtc.sh
# ./dtc.sh
```

The script will fetch the tools on-line, so if your BBB is not connected, you can compile your dtbo file from another Linux machine.

Danger: the dtc program available in Debian Wheezy is not able to write a dynamically loadable dtbo file. It may lack the `-@` option, depending on the system you installed. Check the output of `dtc -h` and whether the `-@` is present. (See the Fetching a Good dtc sidebar.)

The dtbo file produced will then be loaded to the kernel:

```
# cp GPS-PPS-P9_23-00A0.dtbo /lib/firmware
# echo GPS-PPS-P9_23 > /sys/devices/\
bone_capemgr.*/slots
```

To verify that the PPS is seen correctly by the Linux kernel, you need the pps-tools package installed:

```
# ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1391436014.956450656, sequence:\
 202 - clear  0.000000000, sequence: 0
source 0 - assert 1391436015.956485865, sequence:\
 203 - clear  0.000000000, sequence: 0
source 0 - assert 1391436016.956517240, sequence:\
 204 - clear  0.000000000, sequence: 0
source 0 - assert 1391436017.956552407, sequence:\
 205 - clear  0.000000000, sequence: 0
...
(Ctrl-C to end)
```

You see here a signal received at 1Hz, with a timestamp jitter less than 1ms.

The i2C Bus: Two i2c buses on the BBB are available. The kernel sees them as i2c-0 for I2C1 on P9_17(SCL) and P9_18(SDA), and i2c-1 for I2C2 on P9_19(SCL) and P9_20(SDA).

Add an RTC to the BBB: The DS1307 or the DS3231 are RTC modules with a battery keeping the clock running when there is no power. The ChronoDot RTC (with a ds3231) is much more accurate, and the ds1307 is much less expensive.

Wire the RTC on i2c: You can feed the 5VDCC of the board to the RTC module *as long as you clip out the two 2.2k resistors* (Figure 8). The internal resistors of the BBB i2c bus will then be used.

Danger: if you power the BBB over USB, use P9_7 (SYS 5V) instead.

Enable the New RTC: Declare the

Table 2. ds1307 Wiring on the BBB i2c_2 Bus

P9	ds1307
pin 1	GND
pin 5	5VCC
pin 19	SDA
pin 20	SCL

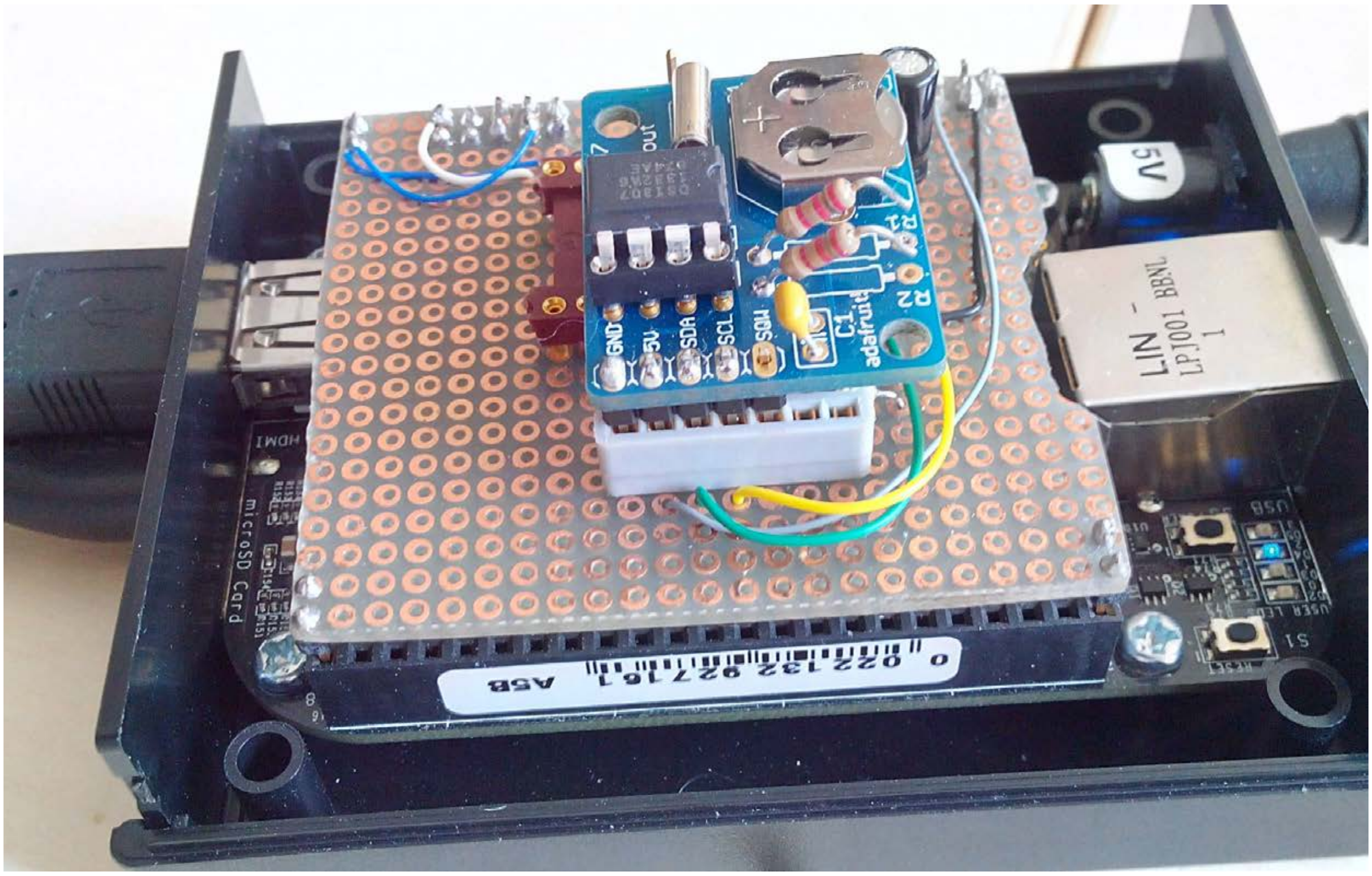


Figure 8. The Adafruit RTC ds1307 Module Wired on the BBB

new RTC to the kernel:

```
# echo ds1307 0x68 >
/sys/class/i2c-adapter/i2c-1/new_device
[   73.993241] rtc-ds1307 1-0068: rtc core: \
registered ds1307 as rtc1
[   74.007187] rtc-ds1307 1-0068: 56 bytes \
nvram
[   74.018913] i2c i2c-1: new_device: \
Instantiated device ds1307 at 0x68
```

Push the current UTC date to the ds1307:

```
# hwclock -u -w -f /dev/rtc1
```

Verify the clock:

```
# hwclock --debug -r -f /dev/rtc1
hwclock from util-linux 2.20.1
Using /dev interface to clock.
Last drift adjustment done at 1406096765 seconds \
after 1969
Last calibration done at 1406096765 seconds after\
 1969
Hardware clock is on UTC time
Assuming hardware clock is kept in UTC time.
Waiting for clock tick...
...got clock tick
Time read from Hardware Clock: 2014/07/23 15:42:51
Hw clock time : 2014/07/23 15:42:51 = 1406130171 \
seconds since 1969
Wed Jul 23 17:42:51 2014  -0.438131 seconds
```

To benefit from the new RTC permanently, as soon as the system boots, modify the /etc/init.d/hwclock.sh file with this little dirty hack. Add to the end of the file:

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/\
new_device
HCTOSYS_DEVICE=rtc1
hwclocksh "$@"
```

Danger: I had to comment the udev part of the file, and I'm still trying to figure how to do that part in a cleaner way:

```
#if [ -d /run/udev ] || [ -d /dev/.udev ]; then
#     return 0
#fi
```

If Something Goes Wrong with a Component on i2c: If the kernel can't see the ds1307, for example, try to detect it on the i2c bus with this:

```
# i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

If 68 is not here, but UU is printed, the kernel already keeps hold of the ds1307 with a driver. Recheck your kernel messages and try to unload the driver for the ds1307 with the following, and then retry:

```
# echo 0x68 >
/sys/class/i2c-adapter/i2c-1/delete_device
# rmmod rtc_ds1307
```

If this is just -- instead of 68, recheck your wiring.

Add a DAC, the MCP4725, on i2c: The MCP4725 is a 12-bit digital analog converter, allowing you to output a voltage from a numerical value between 0 and 4095 (2^{12} - 1). Its EEPROM allows you to store a value in a register that becomes the default value as soon as the DAC is powered on. When you want to drive a motor with it, you then can store a default safe value in EEPROM, and then make certain that when the power is restored, your motor doesn't start at full speed.

The wiring is almost identical as for the ds1307 module; take the 3.3V VDD (P9_3) as VREF for the MCP4725.

The MCP4725 address on i2c is 0x60 or 0x61; select it with the A0 pin on the MCP4725. Thus, you can use two MCP4725s on the same bus, so with two i2c buses, you easily can output four independent voltages from your BBB.

Write a Value to the MCP4725: To set a voltage ouput on the MCP4725, write to the i2c bus. So for a value of 0x0FFF (4095) at the address 0x60 on i2c-1:

```
$ i2cset -f -y 1 0x60 0x0F 0xFF
```

Write to the MCP4725 from Python: The Python module to access the MCP4725 (see the i2c_mcp4725.py file on my GitHub page) is a bit more complex, because we try to handle all the functionalities of the DAC. It depends on the python-smbus package. Here is how we use it:

```
import time
from smbus import SMBus
from i2c_mcp4725 import MCP4725

dac = MCP4725(SMBus(1), int('0x60', 16))
safe_value = 2047
# write to the DAC and store the value in EEPROM
dac.write_dac_and_eeprom(safevalue, True)

# read the value ouput by the dac and the content
# of its EEPROM
dac.read_and_eeprom()
```

```
print dac

# send a mid-ramp 1.69V to 3.38V
for value in range(2048,4096):
  # write to the DAC without storing value
  dac.write_dac_fast(value)
  time.sleep(0.2)
```

Python subtleties allow us simply to do this:

```
# send a new value to output a voltage
dac(new_value)
# check the value currently ouput by the DAC
current_value = dac()
```

How to Read and Verify the Output Voltage? The BBB has seven analog input ports named AIN{0..6}. They are 12-bit analog digital converters, and they accept a max voltage of 1.8V. To read the voltage from an analog input, wire the GND (P9_1) and the + of the voltage you want to measure.

We re-inject the output of our MCP4725 in AIN0. In this case, the VDD for the MCP4725 is 3.38V (the 3.3V of the PIN 3 and 4 of P9). We divide it by two with two resistors, as it must not be greater than 1.8V. And, we wire the output of the resistors to P9_39 (AIN0).

The vRef for the MCP4725 is VMAX (3.38V), so the voltage we send is:

$$V_{out} = outValue \times \frac{V_{max} \times 0.5}{4096} = outValue \times \frac{1.69}{4096}$$

The vRef for the AIN is always 1.8V, so in order to convert our 12-bit numerical value into a voltage reading, we have:

$$V_{in} = inValue \times \frac{1.8}{4096}$$

For a numerical raw value "out", we must read a numerical raw value "in", such as:

$$inValue = int(outValue \times \frac{1.69}{1.8})$$

Read AIN0 from sysfs: If the BBB ADC kernel driver is not loaded, load it now with:

```
# echo BB-ADC /sys/devices/bone_capemgr.8/slots
```

If you need it loaded automatically at boot, do like we did for the BB-UARTs (see the Passing Arguments at Boot Time sidebar).

To read a raw numerical value on AIN0:

```
$ cat /sys/bus/iio/devices/iio\:device0/\
in_voltage0_raw
3855
$
```

Is the Input Consistent with the Output? We can see that the value

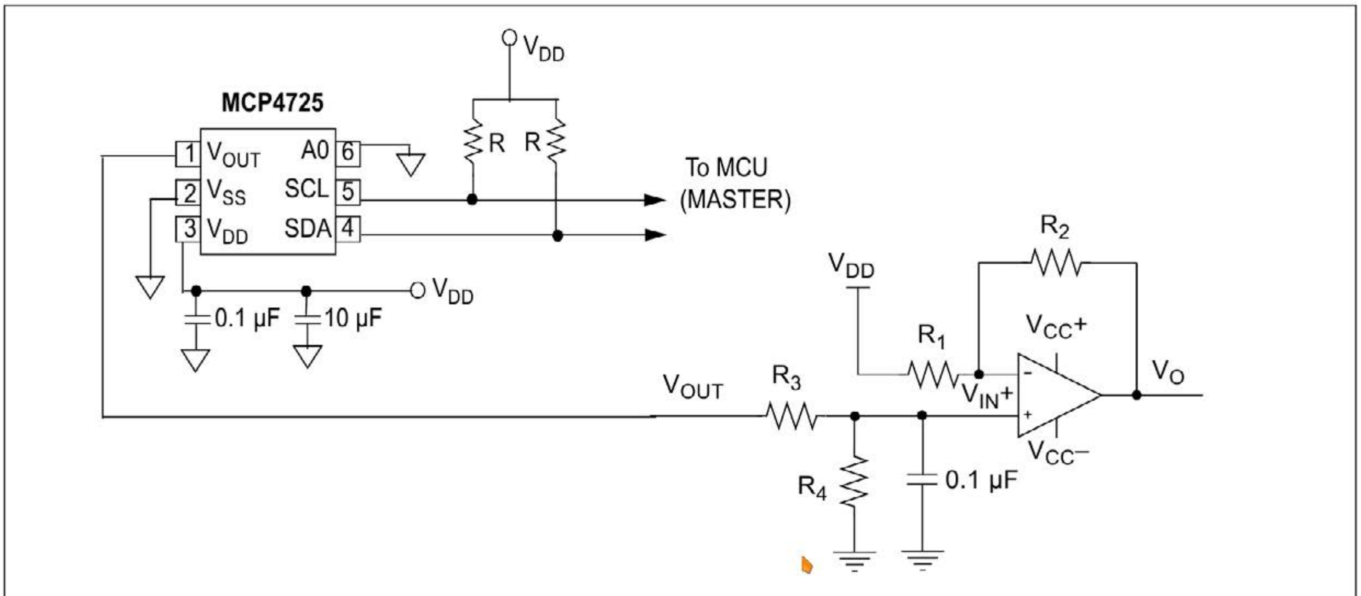
Table 3. Discrepancies between DAC Output and ADC Input

OUT raw value	IN theoretical raw value	IN read raw value
255	239	**243**
2047	1927	**1929**
3839	3614	**3613**
4095	3855	**3835**

sent to the MCP4725 is correctly reread on AIN0 (Table 3).

The values match, but there still are some inaccuracies. We need to record a table of corresponding values between the MCP4725 and AIN0 in this configuration. Our control loop driving our propeller's speed can better handle the re-injection of the output voltage.

Read AIN0 from Python: To read a temperature sensor wired on AIN0 giving 1mV for 0.1°C, we use the



$$V_{OUT} = V_{DD}\frac{D}{2^{12}}$$

where D = DAC Input Code (0 – 4095)

$$V_{IN+} = \frac{V_{OUT}R_4}{R_3 + R_4}$$

$$V_O = V_{IN+}\left(1 + \frac{R_2}{R_1}\right) - V_{DD}\left(\frac{R_2}{R_1}\right)$$

Figure 9. Bipolar Operation Circuit

BBB_AIN class from the bbb_ain.py file (see my GitHub page):

```
import time
from bbb_ain import BBB_AIN

tempe = BBB_AIN(0, valeurmax=180)
for value in tempe:
  print "%.4f" % value
  time.sleep(0.5)
```

Once again, we use a generator. At each iteration, we read a value on the AIN. There is no interrupt mechanism on the AIN; we read at the frequency of the "for loop"—that's why we have to pause at each iteration. The only catch is the error "Resource Temporarily unavailable" (error=11), which may occur occasionally.

How We Use It: To pilot our propellers, we need to output a voltage between –10 V and 10 V. We use two MCP4725s on i2c_1: one with A0 on V~SS~ (0x60) and the other with A0 on V~DD~ (0x61). We use a bipolar operation type circuit to output –10V/10V from the MCP4725's 0/3.3V output (Figure 9).

The MCP4725 outputs are re-injected into AIN0 and AIN1 in order to improve the control loop accuracy. And, we read the speed of the propellers back

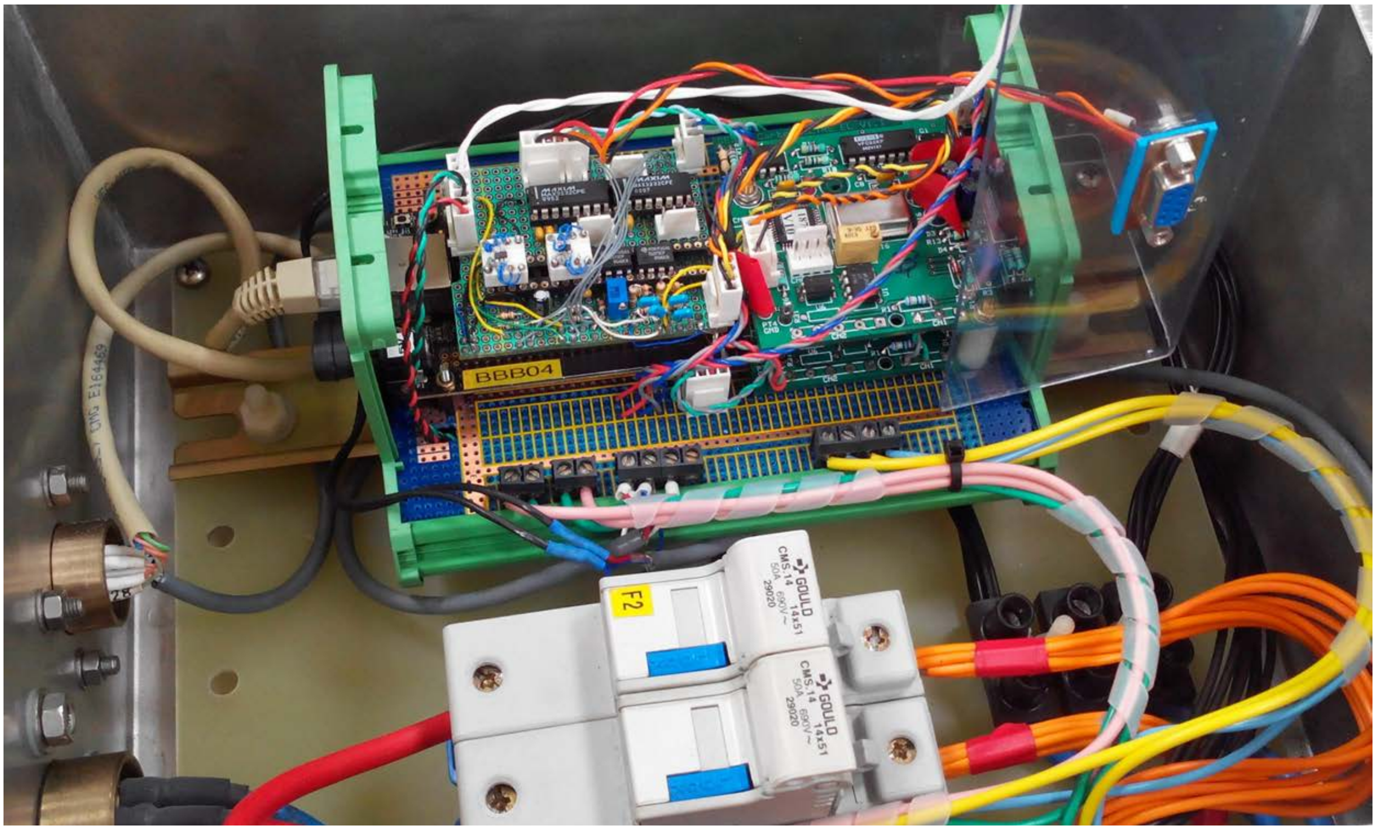


Figure 10. The Box in Charge of the Propellers

with two frequency/voltage converters, which we feed to AIN2 and AIN3 (Figure 10).

The BBB as Time Server from a GPS: What if you put together the interface to your GPS, the NTP server of your Linux machine equipped with an RTC and the accuracy provided with the PPS signal? You can build an NTP server for other CPUs in your network, and with good accuracy, as soon as the GPS is aligned.

Keep in mind that you need a good RTC wired to the BBB for a real NTP server, so choose one like the Adafruit Chronodot (**http://www.adafruit.com/products/255**) rather than the simple DS1307.

GPSd: Now that you know how to handle a serial port, you can install the GPSd software. GPSd connects to a local GPS, as the one we wired to UART4, and serves GPS data to clients:

```
# apt-get install gpsd python-gps gpsd-clients
```

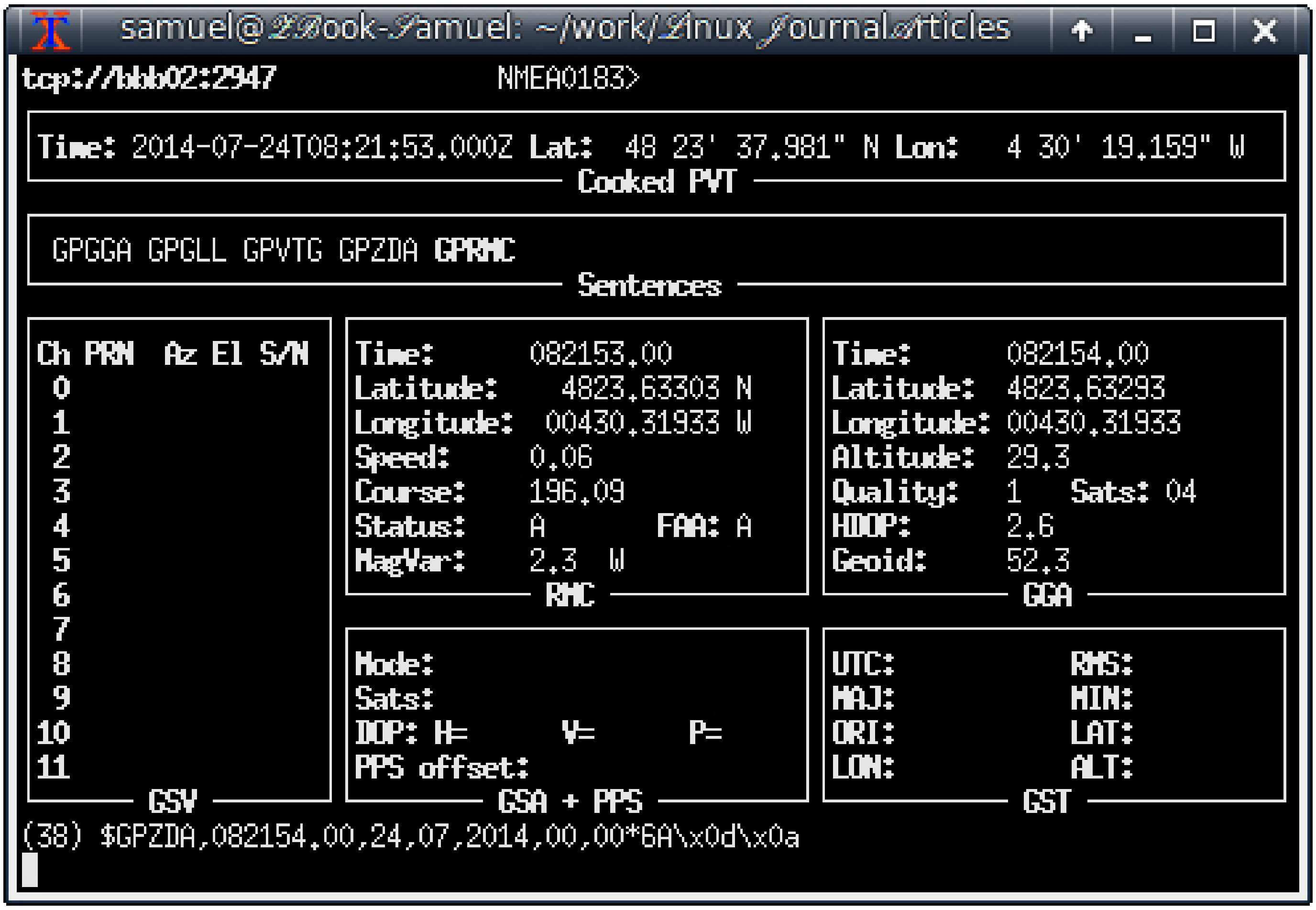

```
tcp://bbb02:2947                NMEA0183>
Time: 2014-07-24T08:21:53.000Z Lat:  48 23' 37.981" N Lon:   4 30' 19.159" W
                               Cooked PVT
 GPGGA GPGLL GPVTG GPZDA GPRMC
                               Sentences
Ch PRN  Az El S/N  Time:       082153.00       Time:      082154.00
 0                 Latitude:     4823.63303 N  Latitude:  4823.63293
 1                 Longitude:  00430.31933 W   Longitude: 00430.31933
 2                 Speed:      0.06            Altitude:  29.3
 3                 Course:     196.09          Quality:   1    Sats: 04
 4                 Status:     A        FAA: A HDOP:      2.6
 5                 MagVar:     2.3  W          Geoid:     52.3
 6                           RMC                          GGA
 7                 Mode:                       UTC:              RMS:
 8                 Sats:                       MAJ:              MIN:
 9                 DOP: H=      V=      P=     ORI:              LAT:
10                 PPS offset:                 LON:              ALT:
11
      GSV                  GSA + PPS                     GST
(38) $GPZDA,082154.00,24,07,2014,00,00*6A\x0d\x0a
```

Figure 11. A Session with gpsmon

Edit the /etc/default/gpsd.conf file, modify it to connect to your serial port (here ttyO4), and tell GPSd to listen to all network interfaces (-G):

```
START_DAEMON="true"
GPSD_OPTIONS="-G -n"
DEVICES="/dev/ttyO4"
USBAUTO="false"
GPSD_SOCKET="/var/run/gpsd.sock"
```

Then restart it:

```
# /etc/init.d/gpsd stop
# /etc/init.d/gpsd start
```

By now, your GPS is available to all clients on your network, and you can try to connect to GPSd with QGIS or OpenCPN, for example, but a rather simple solution is with gpsmon. On another machine with the gpsd-clients package installed, launch:

```
$ gpsmon tcp://bbb02:2947
```

And, you should see a screen like the one shown in Figure 11.

Connect to GPSd from Python: The python-gps package provides what you need, but the dialog sequence with GPSd is not trivial. I wrote a little Python class GPSd_client (see the gpsd_client.py file on my GitHub page) in order to be able to access my GPS, like this:

```
$ ipython
Python 2.7.8 (default, Jul 22 2014, 20:56:07)
Type "copyright", "credits" or "license" for more \
information.
...
In [1]: from gpsd_client import GPSd_client

In [2]: gps = GPSd_client('bbb02')

In [3]: for gpsdata in gps:
    print gpsdata
   ...:
GPS(time=u'2014-07-24T08:53:54.000Z', latitude=\
48.3938485,longitude=-4.505373, altitude=\
30.4, sog=0.051, cog=187.71, ept=0.005, mode=3)
GPS(time=u'2014-07-24T08:53:55.000Z', latitude=\
48.393848, longitude=-4.505373167, altitude=\
30.3, sog=0.067, cog=194.8, ept=0.005, mode=3)
GPS(time=u'2014-07-24T08:53:56.000Z', latitude=\
48.393847667, longitude=-4.505373167, altitude=\
30.2, sog=0.062, cog=184.8, ept=0.005, mode=3)
GPS(time=u'2014-07-24T08:53:57.000Z', latitude=\
48.393847333, longitude=-4.505373167, altitude=\
30.2, sog=0.036, cog=189.77, ept=0.005, mode=3)
GPS(time=u'2014-07-24T08:53:58.000Z', latitude=\
48.393847167, longitude=-4.505373, altitude=\
30.1, sog=0.041, cog=175.46, ept=0.005, mode=3)
^C--------------------------------------------\
------------------------------
```

Again, we use a generator. For each iteration, the GPSd_client class opens a session with GPSd, listens for a report with position and time information, closes the session and returns a `namedtuple` with the information we wanted.

NTPd: One of these GPSd clients is NTP. NTPd processes the NMEA GPS timing sentences to set the date and uses the PPS signal to be more accurate. The handling of the PPS signal is available only in the development versions of NTPd, not the version in the Debian repositories. The version we installed is ntp-dev-4.2.7p416. Grab it and compile it like this:

```
# apt-get install libcap-dev
# wget http://www.eecis.udel.edu/~ntp/ntp_spool/\
ntp4/ntp-dev/ntp-dev-4.2.7p416.tar.gz
# tar xf ntp-dev-4.2.7p416.tar.gz
# cd ntp-dev-4.2.7p416/
# ./configure --enable-all-clocks --enable-\
linuxcaps
# make
```

Modify the /etc/ntp.conf file for the connection to GPSd and the PPS signal listening:

```
# Server from shared memory provided by gpsd
server 127.127.28.0 prefer
fudge 127.127.28.0 time1 0.040 refid GPS

# Kernel-mode PPS ref-clock for the precise seconds
server 127.127.22.0
fudge 127.127.22.0  flag2 0 flag3 1  refid PPS

# allow ntpd to serve time if GPS is OFF
tos orphan 5
```

The 0.040 might need adjusting relative to your GPS, but it's a safe bet. The 127.127.22.0 is the NTPd reference to /dev/pps0. If your GPSd declares a /dev/pps to the kernel, your real PPS signal might become /dev/pps1. The bottom line is try to load your PPS signal before starting GPSd.

Verify that NTPd has started and serves the time with ntpq:

```
# ntpq -p
     remote           refid      st t when poll \
     reach   delay   offset  jitter
=================================================\
==============================
*SHM(0)          .GPS.            0 l   13   64 \
  377    0.000  -50.870   0.886
oPPS(0)          .PPS.            0 l   12   64 \
  377    0.000   -1.419   0.128
```

You can see here that `.GPS.` is identified as the system peer by ntpd (*), and `.PPS.` is also correctly recognized and valid (o).

The PRU, a Very Hot Topic

The two Programmable Realtime Units of the BBB can work independently of the main CPU on I/O ports, AINs and PWM. They are sophisticated enough to share memory with a CPU up to 300MB, have a rich instruction set and can trigger or receive interrupts.

Recently, some hard workers managed to make the use of the BBB PRUs more accessible. And, there is this great promising GSOC 2014 coming, BeagleLogic: **https://github.com/abhishek-kakkar/BeagleLogic/wiki**.

See also the work of Fabien Le Mentec: "Using the BeagleBone PRU to achieve real time at low cost" (**http://www.embeddedrelated.com/showarticle/586.php**).

Conclusion

The BeagleBone Black is a very fun platform to play with. As a Linux sysadmin for nearly 20 years, I'm very comfortable using it to access electronic hardware I'm not so well acquainted with usually.

I want to thank my co-worker, Rodolphe Pellaë, whose skills in electronics were essential in connecting all the components to the board. He did our four homemade Capes in no time and did them well.

The community orbiting the BBB is large with a lot of good on-line resources. We managed to learn some complex stuff in very little time and with almost no confusion, because we can profit from the work of those people and from the Linux and Python ecosystems.■

Samuel Bucquet is a system developer and a sysadmin on robotic platforms in the French DOD. He is married with four kids and lives in Brest, France, and he is a longtime Linux aficionado.

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

Resources

You will find the source of the Python code used in this article on my GitHub account: **https://github.com/samgratte/BeagleboneBlack**.

PyBBIO is a Python Library for the BBB mimicking the Arduino IO access by Alexander Hiam: **https://github.com/alexanderhiam/PyBBIO**.

Of course, there is also the Python Adafruit Library, initially for the Raspberry Pi, now for the BBB: **https://github.com/adafruit/adafruit-beaglebone-io-python**.

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**

DOC SEARLS

Learn GNU/Linux the Fun Way

A great "hello world" from rural Utah.

Sometimes a gift just falls in your lap. This month, it came in the form of an e-mail out of the blue from Jared Nielsen, one of two brothers (the other is J.R. Nielsen) who created The Hello World Program, "an educational web series making computer science fun and accessible to all" (http://www.thehelloworldprogram.com). If it had been just that, I might not have been interested.

But when I looked at it, I saw it was hugely about Linux. And the human story was interesting too. Wrote Jared, "Working in rural Utah with minimal resources, we combine technology and craft to make educational yet entertaining videos and tutorials. Learn to code with our cute and clever puppets." So I said I'd like to interview them, and here's how it went.

DS: What got you going on this?

JN: Growing up, we wanted to be making creative media, such as games, videos, animations, etc., but in the days before the Internet, training was either difficult or expensive to find, especially in small-town Utah. We figured things out on our own, through trial and error, visits to the library and countless hours watching PBS. The Hello World Program is the show we wish we watched as kids.

DS: Do you or your brother have kids yourselves?

JN: Neither of us have kids, but we are both kids at heart.

DS: How long have you been doing it?

JN: We started kicking around ideas for an educational Web series in January 2012. Then we sat on our hands for a few months. We published our first video, "What Is a Robot?" in May of that year. Since then, The Hello World Program has significantly evolved. We introduced several new characters and expanded our scope to include Web development and programming.

DS: Where in rural Utah are you? (A side thing—I love Utah and have shot it a lot from the air: https://www.flickr.com/search/?text=Utah&user_id=52614599%40N00&sort=interestingness-desc.)

JN: Our home is Richfield, the gateway to adventure in Southern Utah.

DS: Cool! I shot your house from the sky just this past May 4th, in fact. (Here it is: https://www.flickr.com/photos/docsearls/14861798514/in/set-72157646284287131).

So, why Linux? We love Linux here, but not all computer science education starts with Linux, especially for kids. But you started there, which is very cool.

Between the two poles of servers and smartphones—and the ubiquity of computers—we think the majority of users will eventually run a Linux distribution.

JN: Linux is the future! So we hope. Between the two poles of servers and smartphones—and the ubiquity of computers—we think the majority of users will eventually run a Linux distribution. One of our goals is to remove the economic barrier associated with learning computer science. Not only are the majority of Linux distros free, they are amazingly powerful and customizable and can breathe new life into old hardware.

Linux is also very hands-on, and learning it is also learning how a computer works. Other popular operating systems obscure the inner workings of the computer from the user. With Linux, students can choose how deep they want to dive into their machines.

DS: And you do all your production on Linux?

JN: Yes. We have such an appreciation for this operating system that we challenged ourselves to produce all of our media using Linux machines. We edit our videos with Lightworks (**http://www.lwks.com**). Our computer graphics are rendered in Blender (**http://www.blender.org**). Our audio is processed using Audacity (**http://audacity.sourceforge.net**) and Ardour (**https://ardour.org**), and our stopmotion animations are created using Entangle (**http://entangle-photo.org**) and compiled with avconv (**https://libav.org/avconv.html**).

DS: Why puppets? (Side thing—my daughter Colette is a college professor who teaches puppet theater: http://www.umbc.edu/theatre/searls.html.)

JN: We learned about the world by watching *Sesame Street*, *The Muppets* and *Mr. Rogers' Neighborhood*. We wanted to create a show in the same vein with a contemporary sensibility. We loved the idea of merging analog craft with digital technology. In addition to puppets, a lot of our new content features stopmotion and hand-drawn animation as well as 3-D computer-generated animation.

DS: Why Python? We can guess, but we'd rather ask anyway.

JN: Python is an excellent choice for a beginner because it issues immediate results, it's very easy to read, and it can be used for a wide variety of applications. Best of all, it is included with most Linux distributions.

DS: Do you teach locally as well as on-line (for example, in local schools)? If so, how and where?

JN: We ran workshops with hacker/Makerspaces in the past, but found that our two-man team didn't have the time and resources necessary to teach and continue producing new content for our site.

DS: Do you have commercial or other ambitions besides what you're doing now? (Such as seeing these adopted in schools?)

JN: We would love to see our curriculum adopted in schools, though our ambitions are more autodidactic. Our primary goal is to empower young people to make their own media.

Advertiser Index

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

ADVERTISER	URL	PAGE #
All Things Open	http://allthingsopen.org	2
Drupalize.me	http://www.drupalize.me	7
EmperorLinux	http://www.emperorlinux.com	21
Silicon Mechanics	http://www.siliconmechanics.com	3

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.

DS: Do you have a business with this as well?

JN: Yes. We're bundling our videos and selling them as a digital download. Our first complete segment, Daisy's Web Dev Diary, is available for purchase via Gumroad (**https://gumroad.com/l/daisy**).

We're also planning to release e-books for each of the four tracks of The Hello World Program. And we're slowly building a merchandise store, for fans to purchase Hello World-themed T-shirts, stickers and posters (**http://www.zazzle.com/dototot**).

DS: You identify on Twitter as a "Creative media company and think tank" (https://twitter.com/dotototdotcom). Tell us more about both (especially the think tank part).

JN: The Hello World Program is our first project. As a creative media company, we are producing an educational and entertaining series of videos and tutorials combining art and technology. As a think tank, we recognize that understanding the basics of computer science will be a necessary skill set for the future. Through STEM initiatives, technological literacy will be taught to students at a younger age and eventually incorporated in standard curriculum. We're predicting that Linux and Python will be at the forefront of this movement. We support an open-source future and consider our current project a contribution to furthering the cause.

Otherwise, we're perpetually brainstorming new endeavors beyond puppets and video. But we're not ready to talk about them yet.

DS: Can you give us some success stories with kids?

JN: Our booth at Maker Faire Bay Area 2013 was wildly successful. It was titled "Robot Puppet Party", and we invited young makers to use our puppets to create their own short videos. We were awarded two Editor's Choice blue ribbons. But more important, the kids loved it.

We're also launching an Indiegogo campaign called "Hands-On Computer Science: The Hello World Program", which should go up in September (http://igg.me/at/hello-world).

DS: Good. That's just in advance of this issue of the magazine. Any other projects?

JN: We're launching a new segment, "Superusers: The Legendary GNU/Linux Show" (https://www.youtube.com/watch?v=DQbODhkyA4g&feature=youtu.be).

DS: Excellent! I like Aramis the Gnu and Adalie the penguin!

JN: We're also launching a crowdfunding campaign on Indiegogo that should be up by the time you read this.

DS: Where can readers follow you?

JN: Find us at http://www.dototot.com and on Twitter at https://twitter.com/dototdotcom or https://twitter.com/helloworldshow.■

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.