LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

A LOOK AT KDE's KStars Astronomy Program

FEBRUARY 2016 | ISSUE 262 | www.linuxjournal.com

Programming How-Tos

+ Working with Command Arguments in Your Shell Scripts

Program a BeagleBone Black to Help Brew Beer

Write a Short Script to Solve a Math Puzzle

Interview: Katerina Barone-Adesi on Developing the Snabb Switch Network Toolkit

WATCH: ISSUE OVERVIEW

CONTENTS

FEBRUARY 2016
ISSUE 262

FEATURES

COLUMNS

IN EVERY ISSUE

Welcome to the **KStars Telescope Setup Wizard!**

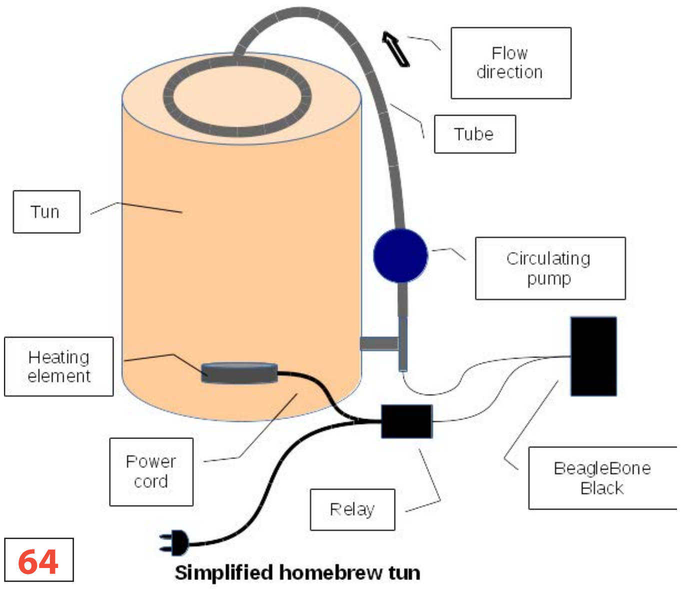
This Wizard will help you to connect to your telescope and control it from KStars. You will be asked to verify some basic information.

You can get extended information on telescope support in KStars by pressing the help button at any point during the Wizard.

Please click next to continue.

22

46



64

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

LINUX JOURNAL

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

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

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

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

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

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

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

LINUX is a registered trademark of Linus Torvalds.

peer1 hosting

Where every interaction matters.

break down
your innovation barriers

power your business to its full potential

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

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

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

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

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

SHAWN POWERS

For the Love of Linux

I love my job. I teach Linux by day and write about Linux at night. It's easy to fall in love with your work when the things you do align with your passions. All of us here in the *Linux Journal* community have a love for Linux and open source, but even inside our world, there are some topics that are just downright fun! This month is full of articles we're passionate about.

Joey Bernard starts off with a look at KStars. If you're a space nut like me, you'll want to check out his detailed look at the KDE-native astronomy program. It's dark enough to see the stars only at night, but with Joey's help, you can surf the night sky any time of day! Then, what better way to end a day of LCD star gazing than to read a tech book on your favorite subject?

VIDEO:
Shawn Powers runs through the latest issue.

Reuven M. Lerner provides a breakdown of some of his top picks on topics from programming to podcasting. It's hard to go wrong with a good book, and Reuven will help you find one.

With the help of Dave Taylor, you will learn to deal with command arguments in scripts using `getopt` in your code. Dealing with arguments doesn't seem like a big deal, but what if people combine them? (For instance, `-rf` instead of `-r -f`?) Rather than write pages of conditionals, Dave explores how to use `getopt`, which does all the dirty work for you. If you need to write a script that accepts command-line arguments, this month's column is a must-read.

I take a step away from the keyboard this issue and head around to the back of the computer—specifically, to the power cord. My family recently moved into an old turn-of-the-century house, and even with a massive electrical overhaul, the power is flaky at best. I decided to share my experience this month and clarify some of the various hardware options available to help regulate and stabilize the electricity coming into your home or business. The fact that I'm writing this on a desktop computer rather than a laptop means my hard work has paid off!

Susan Sons has an incredible interview with Katerina Barone-Adesi this month where she gets the details on Snabb. If you've never heard of Snabb, you'll be glad you read the interview, because it's an incredible network toolkit that isn't built in to the Linux kernel. If the idea of bypassing the Linux kernel to increase speed seems odd, you definitely want to read this interview!

You might remember Kyle Rankin writing a few years ago about using a Pogo Linux device (and later a Raspberry Pi, I think) to keep his homebrew beer chilled while it aged. Klaus Kolle takes a different approach, and using a BeagleBone Black, he controls the heating of the mash during his beer-making process. Whether you need to keep your homebrew hot or cold, it turns out Linux can be the answer in either case! Check out his cool project this month, and potentially improve your beer while improving your geek cred.

Sol Lederman finishes off the issue with some awesome problem solving. The best use of Linux, or technology in general, is to make quick work of something that would take a long time to do on your own. Scripting was designed for that exact purpose. In this command-line tutorial, Sol describes how to write a one-liner to find a specific date, and then put that one-liner into a script to make it more flexible. Most good programs start with a problem that needs to be solved, and in this case, the problem is trying to find the next Friday the 13th.

We all have passions that drive us to learn and explore. For some of us (me!), it's feeding the birds. For others, it's brewing the perfect beer. The best part about being a Linux geek is that we can take advantage of Linux's open-source nature and make our passions really come to life. We hope you share some of the excitement we brought into this issue and can't wait to hear about what sorts of projects you improve with the help of our favorite OS! ■

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

Hardware Interrogation

I've just read Federico Kereki's article about interrogating a Linux system titled "What's in the Box? Interrogate Your Hardware" in the December 2015 issue. I love this kind of article and hope to see more!

—Brian Clark

Federico Kereki replies: *Thanks, Mr Clark, for your kind words. The article grew out of my actual need to know about the hardware in my own machine, and because of Linux's openness, I learned even more than I had expected. I'm glad you liked my results!*

Server Hardening

Regarding Greg Bledsoe's "Server Hardening" article in the November 2015 issue: great article—lots of detailed help in one source. One question: is it possible to mention specific logs you reference in the article? I get lost quickly when seeing the large number of logs scattered about my Debian server.

—Tom Browder

Bug in Script

I believe that the final version of the script that Dave Taylor came up with in his Work the Shell column titled "Analyzing Comma-Separated Values (CSV) Files" in the December 2015 issue of *Linux Journal* contains an oversight. Specifically, it does not handle the case in which more than one field contains commas (for example, the dollar amount field and the comment field). I have modified Dave's script to take this into account. Hopefully, this will be of some help. I always enjoy Dave's column and have learned a lot from it. Here's the modified script:

```
#! /bin/bash -


# fixcsv
# fix CSV files with embedded commas

# The problem is that some spreadsheet fields may contain
commas. In the sample case, this includes the dollar
amount and comment fields. I believe you overlooked the
case in which both the dollar amount and comment fields
contain commas. Your script assumes that there is at
most one such instance.

# The simplest solution is to export the spreadsheet
contents with some field delimiter that can never appear
in any field, e.g., a tab. Then write the script
using this delimiter.

# Original code

# while read inline
# do
#   if [ ! -z "$(echo $inline | grep \")" ]
#   then
#     f1=$(echo $inline | cut -d\" -f1)
#     f2=$(echo $inline | cut -d\" -f2)
#     f3=$(echo $inline | cut -d\" -f3)
#     echo $f1`echo $f2|sed 's/,//g'`$f3
#   else
#     echo $inline
#   fi
# done
# exit 0

# This works correctly ONLY when there is EXACTLY ONE
field enclosed in double quotes.

# Revised code

# For each line that contains at least one field enclosed
in double quotes, process each such field from left to
right until no fields are enclosed in double quotes and
all remaining commas are field separators. The steps are:
(1) replace the double quotes enclosing the field being
processed with a temporary delimiter to isolate that
specific field, (2) remove any commas embedded in the
isolated field, (3) reconstruct the line without the
temporary delimiters. The temporary delimiter must be
a single character (for the cut command) that cannot
appear in the input file. I selected an asterisk (*),
but other characters can be used. Some characters (such
as asterisk, colon, hyphen, and equals) work fine, while
others (such as tab and semicolon) do not.

td=*  # temporary delimiter

while read inline
do
  while [ ! -z "$(echo $inline | grep \")" ]
  do
    inline=$(echo $inline | sed "s/\"/$td/" | sed "s/\"/$td/")
    f1=$(echo $inline | cut -d"$td" -f1)
    f2=$(echo $inline | cut -d"$td" -f2)
    f3=$(echo $inline | cut -d"$td" -f3)
    inline=$(echo "$f1$(echo $f2 | sed 's/,//g')$f3")
  done
  echo $inline
done
exit 0

# Test input file fixcsvtest.txt:

$ cat fixcsvtest.txt
```

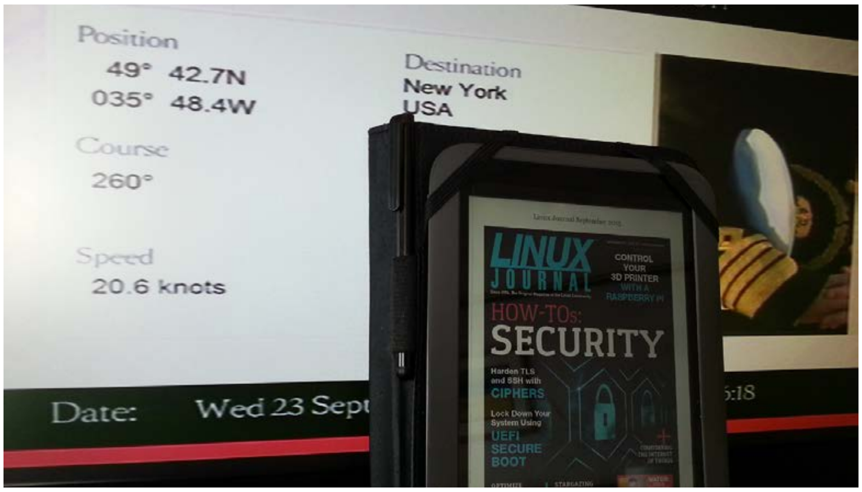
```
4/7/14,subscriptions,199.99,Ask Dave Taylor Monthly
4/10/14,subscriptions,"1,300.99",Linux Journal
4/10/14,subscriptions,"1,300.99","Linux Journal, APR 2014"
4/10/14,subscriptions,19.99,"Linux Journal, annual"

ab,cd,ef,gh
ab,cd,ef,"g,h"
ab,cd,"e,f",gh
ab,cd,"e,f","g,h"
ab,"c,d",ef,gh
ab,"c,d",ef,"g,h"
ab,"c,d","e,f",gh
ab,"c,d","e,f","g,h"
"a,b",cd,ef,gh
"a,b",cd,ef,"g,h"
"a,b",cd,"e,f",gh
"a,b",cd,"e,f","g,h"
"a,b","c,d",ef,gh
"a,b","c,d",ef,"g,h"
"a,b","c,d","e,f",gh
"a,b","c,d","e,f","g,h"
$

# Test Results

$ ./fixcsv < fixcsvtest.txt
4/7/14,subscriptions,199.99,Ask Dave Taylor Monthly
4/10/14,subscriptions,1300.99,Linux Journal
4/10/14,subscriptions,1300.99,Linux Journal APR 2014
4/10/14,subscriptions,19.99,Linux Journal annual

ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
ab,cd,ef,gh
$
```

—Jeff Mumma

Dave Taylor replies: *Thanks for your note, Jeff, and I do believe you're correct that I didn't test the case where more than a single field of the input data included commas. Bah, pesky debugging! I like your mods, and yet still have a niggling sense that the entire problem can be sidestepped with the perfect regular expression. If I only had a few weeks to create it!*

Photo of the Month

I thought you would like to see an unusual place where *LJ* is being read this month: 49 degrees north, 35 degrees west. That's the middle of the Atlantic Ocean at 20 knots heading for NYC. The satellite Internet costs are rather steep on board so I brought a few issues with me. Must dash as the sun is over the yard arm.



—Roger Greenwood

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

WRITE *LJ* A LETTER

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

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

The **OOM killer** is a tough nut to crack. How can a system recover when it's violently thrashing and out of RAM? Once upon a time, you'd just have to reboot. And today, that still might be necessary, but less so, because the OOM killer attempts to identify and stop the process that seems to be causing the hangup. The problem is, it may not choose the right process every time. Another problem is that the whole thing is super tough and complex.

Michal Hocko recently tried to peel off a sliver to work on, taking the lead from **Mel Gorman** and **Oleg Nesterov**. Apparently, the current OOM killer would allocate an extra batch of memory just for the process it wanted to kill to actually give it enough breathing room to terminate properly. But under some circumstances, the process would accept the extra memory and still hang the system. Then with no more memory to dole out, the OOM killed couldn't try again, and it was time to hit the reset button.

Michal posted a patch to create a new kernel thread that would reclaim that extra memory if it went unused. Then the OOM killer could try the same thing on a different process and hopefully have a different result. And although there were no major objections to Michal's patch itself, a variety of folks objected to the idea of making any kind of incremental improvement to the OOM killer, when the Big Problem had not yet been solved.

The Big Problem, as described by **Johannes Weiner**, was how to resolve memory deadlocks in general. Only by solving that problem could the OOM killer successfully kill the processes it needed to, even to the point of killing all user processes, just to keep the kernel up.

But, Michal made a point of keeping the discussion clamped down to a consideration of only the small fixes he'd proposed. He acknowledged that he had no solution for the Big Problem, and he pointed out that no one else seemed to have a viable solution for the Big Problem either. And until something viable came up, Michal saw no point in stalling OOM killer development. If something could be done to improve it, he felt, then it should be done.

By and large everyone went along with this, but still, it's clear there's a lot of pressure on the OOM killer system to come up with some kind of new idea or at least to create a policy-based system that puts control of the choices of processes to kill into the hands of system administrators rather than the kernel algorithms themselves.

Linus Torvalds had some advice for anyone writing kernel code that needs to lock resources: it's probably better to use existing locking implementations rather than rolling your own—at least, until you know what you're doing. As he put it:

> People need to realize that locking is harder than they think, and not cook up their own lock primitives using things like trylock without really thinking about it a *lot*.
>
> Basically, `trylock()` on its own should never be used in a loop. The main use for trylock should be one of:
>
> 1) Thing that you can just not do at all if you can't get the lock.
>
> 2) Avoiding ABBA deadlocks: if you have an A->B locking order, but you already hold B, instead of "drop B, then take A and B in the right order", you may decide first to `trylock(A)`, and if that fails, you then fall back on the "drop and relock in the right order".
>
> But if what you want to create is a "get lock using trylock", you need to be very aware of the cache coherency traffic issue at least.
>
> It is possible that we should think about trying to introduce a new primitive for that `loop_try_lock()` thing. But it's probably not common enough to be worth it—we've had this issue before, but I think it's a "once every couple of years" kind of thing rather than anything that we need to worry about.
>
> The "locking is hard" issue is very real, though. We've traditionally had a *lot* of code that tried to do its own locking, and not getting the memory ordering right, etc. Things that happen to work on x86 but don't on other architectures, etc.

—ZACK BROWN

Android Candy: Quick Games

The biggest problem I have with gaming is that it takes far too long to get "into" games. I'm generally very busy, and my gaming time usually lasts as long as it takes for the dentist to call me in from the waiting room (or possibly how long it takes me to use the bathroom, but *eiw*, let's not go there). For me, the perfect game can be fun even if I can play only for a few minutes. It also has to be very quick to learn, because "learning to have fun" isn't very much fun at all.

There are a few old standbys that work well: *Bejeweled*, *Angry Birds*, *Peggle*, *Plants vs. Zombies*, *Candy Crush* and so on. The problem is, although those games are fun, I grow tired of them fairly quickly—either that or they're so addictive I fear my family will disown me for ignoring them while I play just one more level.

Here are some games I've been playing lately:

- *Swish*: it's a puzzle game, but it's just different enough that I find it fun. The physics seem "right" to me while playing, and the graphics are really great. The premise is that you're an alien playing basketball in space. You know, like aliens always do. There are 60 levels, and there's enough of a challenge to make it fun.

- *Dumb Ways to Die 2*: I loved the first *Dumb Ways to Die* game. It was absurd. It was funny. It was cartoonishly morbid. Part 2 is more of the same, and that's a very, very good thing.

- *Asphalt Series*: there are a bunch of games in the Asphalt racing series. Some of them work better than others, and all of them are fairly large downloads. I like *Asphalt* because you get to race really cool cars really fast. If you ever played *Burnout* on the PlayStation, *Asphalt* games will seem pleasantly familiar. You have to pay for gas if you play for too long, but I generally don't have enough time and play each session only until my gas is gone.

What are your favorite "five-minute games" for the Android platform? I never can really get into a book in that short amount of time, so it tends to be the only time I play games. If you have any suggestions, drop me a message at shawn@linuxjournal.com and I'll try to follow up in later months with the best of the best.

Note: you can find all the games mentioned here in the Google Play Store.—SHAWN POWERS

They Said It

Sometimes we do a thing in order to find out the reason for it. Sometimes our actions are questions not answers.
—John Le Carré

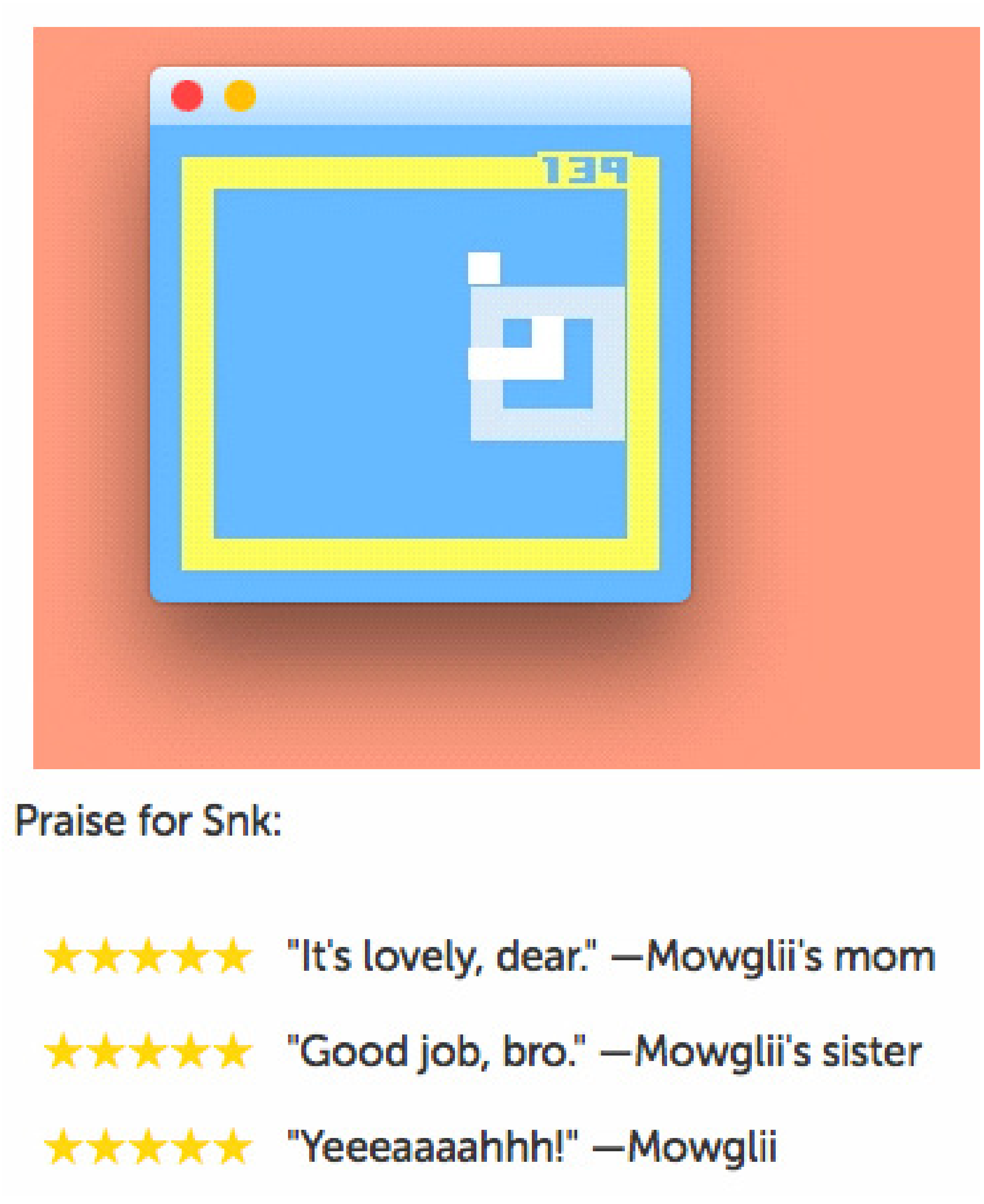
The greatest justice in life is that your vision and looks tend to go simultaneously.
—Kevin Bacon

There are some things you learn best in calm, and some in storm.
—Willa Cather

The only true happiness comes from squandering ourselves for a purpose.
—William Cowper

If your ship doesn't come in, swim out to it!
—Jonathan Winters

Non-Linux FOSS: *Snk*



Praise for Snk:

★★★★★ "It's lovely, dear." —Mowglii's mom

★★★★★ "Good job, bro." —Mowglii's sister

★★★★★ "Yeeeaaaahhh!" —Mowglii

I'm apparently in a silly-game mood this month, because I stumbled across an open-source project I couldn't keep all to myself: *Snk*. If you remember the classic game of *snake*, *Snk* is the same concept, but smaller, harder and with music.

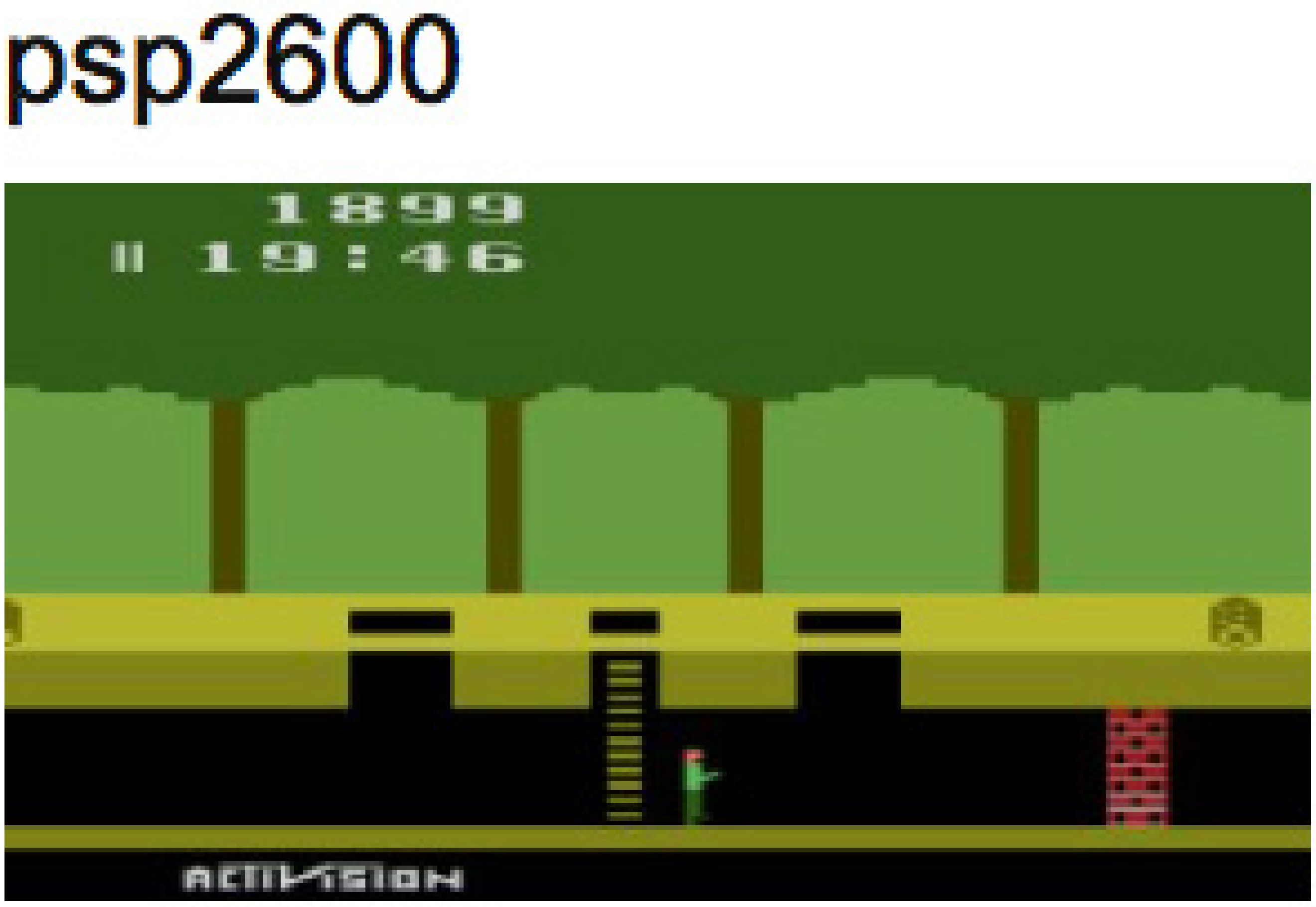
I actually really like the *Snk* program because it's fairly simple, and the developer (Mowglii) has put the Xcode project on GitHub for folks to download. If you're just starting with OS X Swift development, *Snk* is a project you can tweak for some learning on the fly.

My favorite part of the project is actually the "reviews" for it. Head over to http://www.mowglii.com/snk to grab *Snk* and its source code today. And, good luck with level three; I run directly into the wall every time!

—SHAWN POWERS

Handheld Emulation: Achievement Unlocked!

I love video game emulation. My favorite games were produced in the 1980s and 1990s, so if I want to play them, I almost always have to emulate the old systems. There is usually a legal concern about ROM files for games, even if you own the original cartridges, so I'm not going to tell you where to find ROMs to download or anything like that. What I *am* going to share is my recent discovery of the perfect handheld gaming system. Oddly enough, it was never intended to be



(Image from http://wololo.net)

an emulator.

The PSP is truly incredible hardware. The PSP Vita is its bigger, younger sibling, but if you have an old PSP, I urge you not to throw it away. With a simple firmware hack (also legally questionable, I suppose), it's possible to load emulators that will play Atari, NES, SNES, Game Boy, Genesis, PS1 and most other console games almost flawlessly.

I never had a PSP, but I was able to get a PSP Go in mint condition on eBay for $89. The PSP Go comes with 16GB of storage, so you don't even need to get its proprietary memory card to load it up with games!

One of the problems with the emulation scene is that sites seem to come and go fairly regularly. I found all the information I needed to get my PSP Go ready to play *Mario* by doing some Google searching for PSP emulators. Specifically, this page was great: http://wololo.net/emulators-for-the-psp-ps-vita-the-ultimate-download-list.

If you already have a PSP device, the instructions for custom firmware installation is simple. If you don't have one, deciding which version of the PSP to purchase is one of the toughest steps. If you like the larger layout, I recommend the PSP 2000 model. It has an incredible screen and fewer buggy design choices than the original. If you're looking for portability, I'm very fond of the PSP Go I purchased from eBay. The screen is smaller, but it's still plenty large and has beautiful quality. Good luck, and have fun!

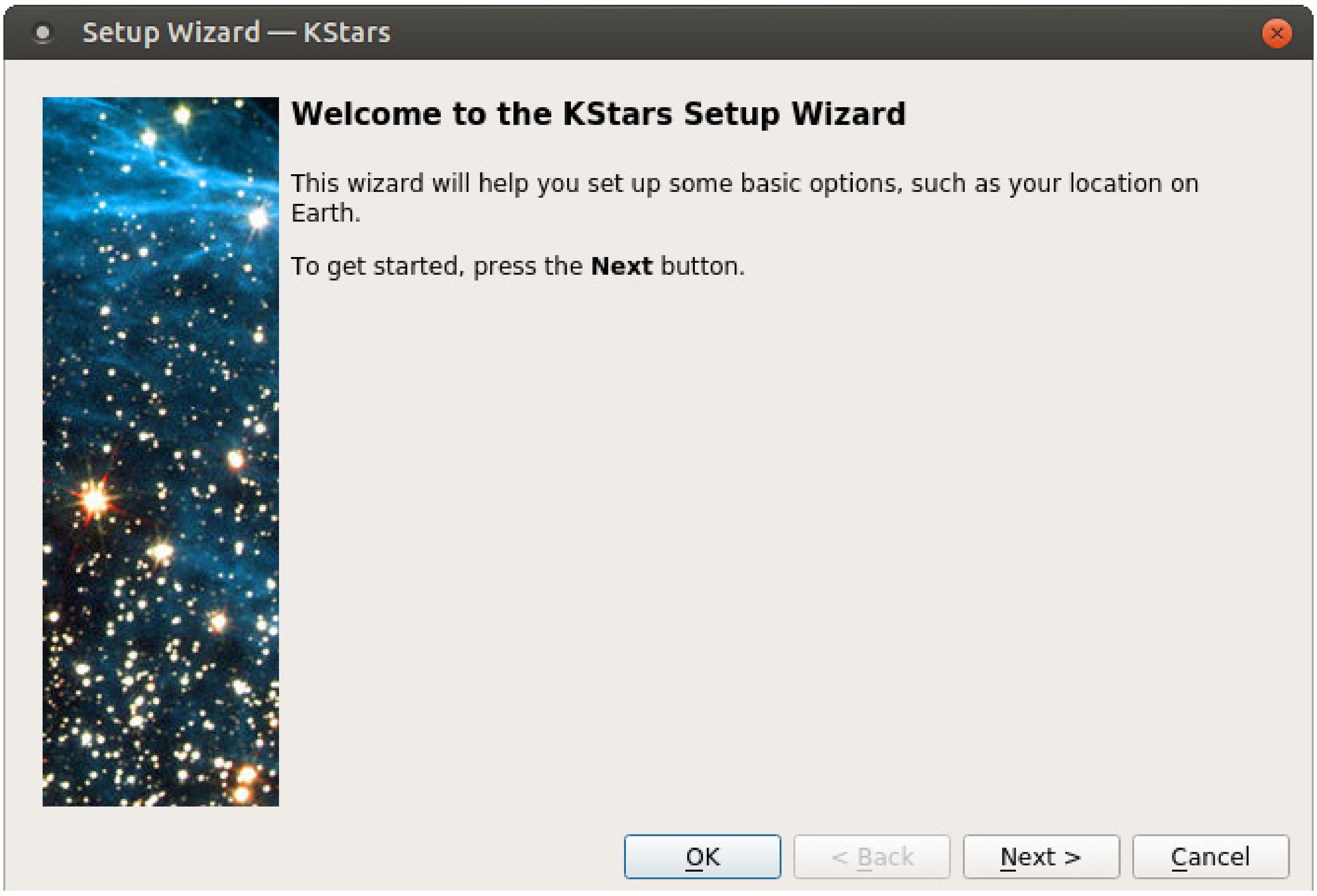
—SHAWN POWERS

Astronomy for KDE

Although I have covered a large number of science applications in this space, I haven't really looked at too many options available within the KDE desktop environment. This has been due to my own biases in using a GTK-based desktop environment, but now I'd like to look at some of the packages available for people who really like to use KDE on their own machines. So in this article, I'm starting off with the KStars astronomy program.

If you have the full KDE environment installed, you already should have it available. If you don't, you should be able to install it. For example, you can install KStars on Debian-based distributions with this command:

```
sudo apt-get install kstars
```


Setup Wizard — KStars

Welcome to the KStars Setup Wizard

This wizard will help you set up some basic options, such as your location on Earth.

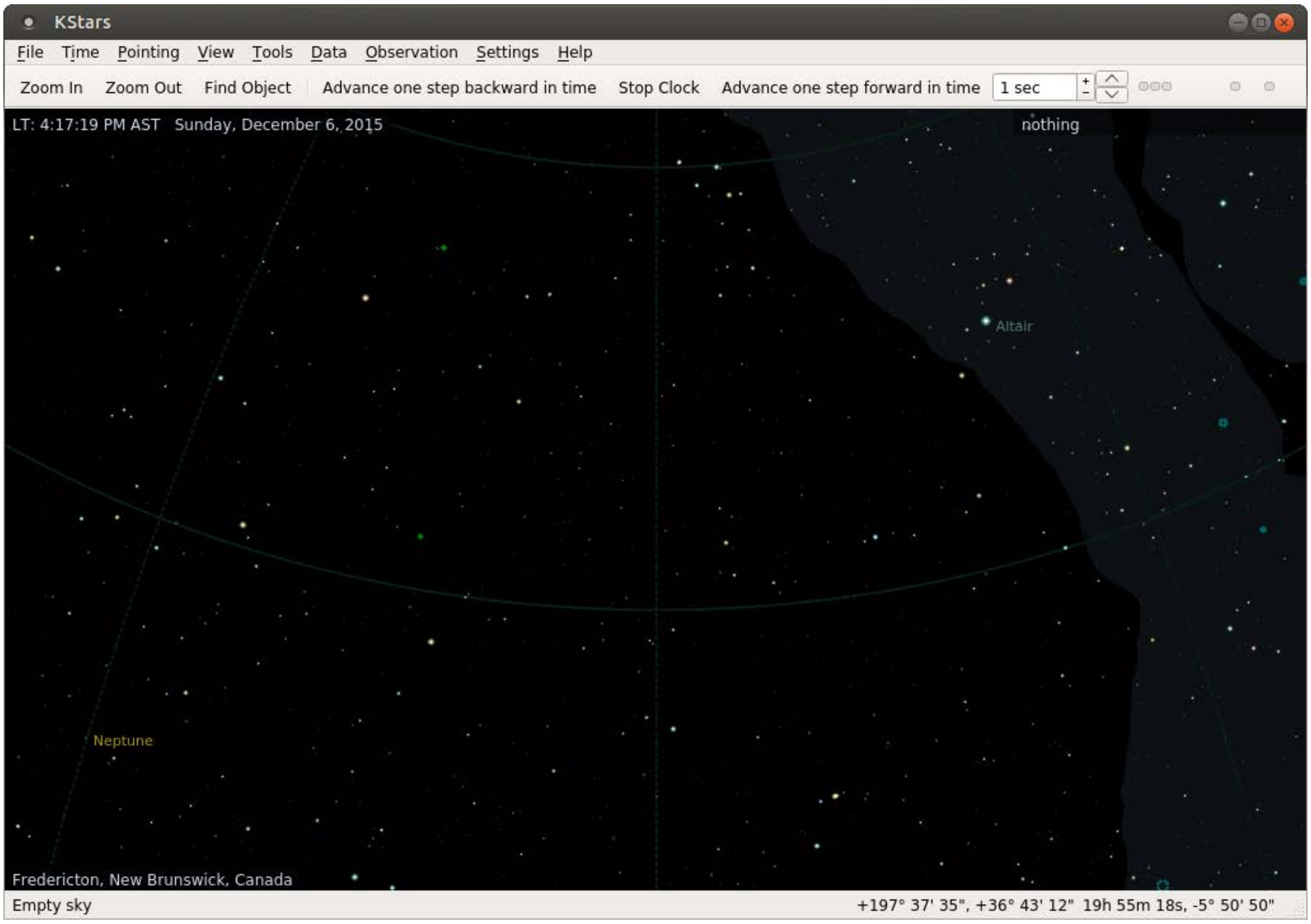
To get started, press the **Next** button.

Figure 1. When you first start KStars, you need to go through the setup wizard.

If this is the first KDE-based application that you are installing, it also will need to pull in a rather large set of dependencies—that's just the price of using a new GUI toolkit. Of course, any other packages will be incrementally smaller since all of the shared dependencies already will be there.

To launch KStars, you either can click on a menu item in your desktop environment or enter the `kstars` command in a terminal window. The first time you start KStars, you need to go through the setup wizard to configure elements like your location. Once KStars finishes starting up, you should see a display of the sky from the location you set during the setup.

You can pan the display around simply by clicking and dragging the star field to see locations of interest. The items that are labeled depend


KStars
File Time Pointing View Tools Data Observation Settings Help
Zoom In Zoom Out Find Object Advance one step backward in time Stop Clock Advance one step forward in time 1 sec
Altair
Neptune

LT: 4:17:19 PM AST Sunday, December 6, 2015

nothing

Fredericton, New Brunswick, Canada

Empty sky

+197° 37' 35", +36° 43' 12" 19h 55m 18s, -5° 50' 50"

Figure 2. You get a display of the sky from your location when KStars starts.

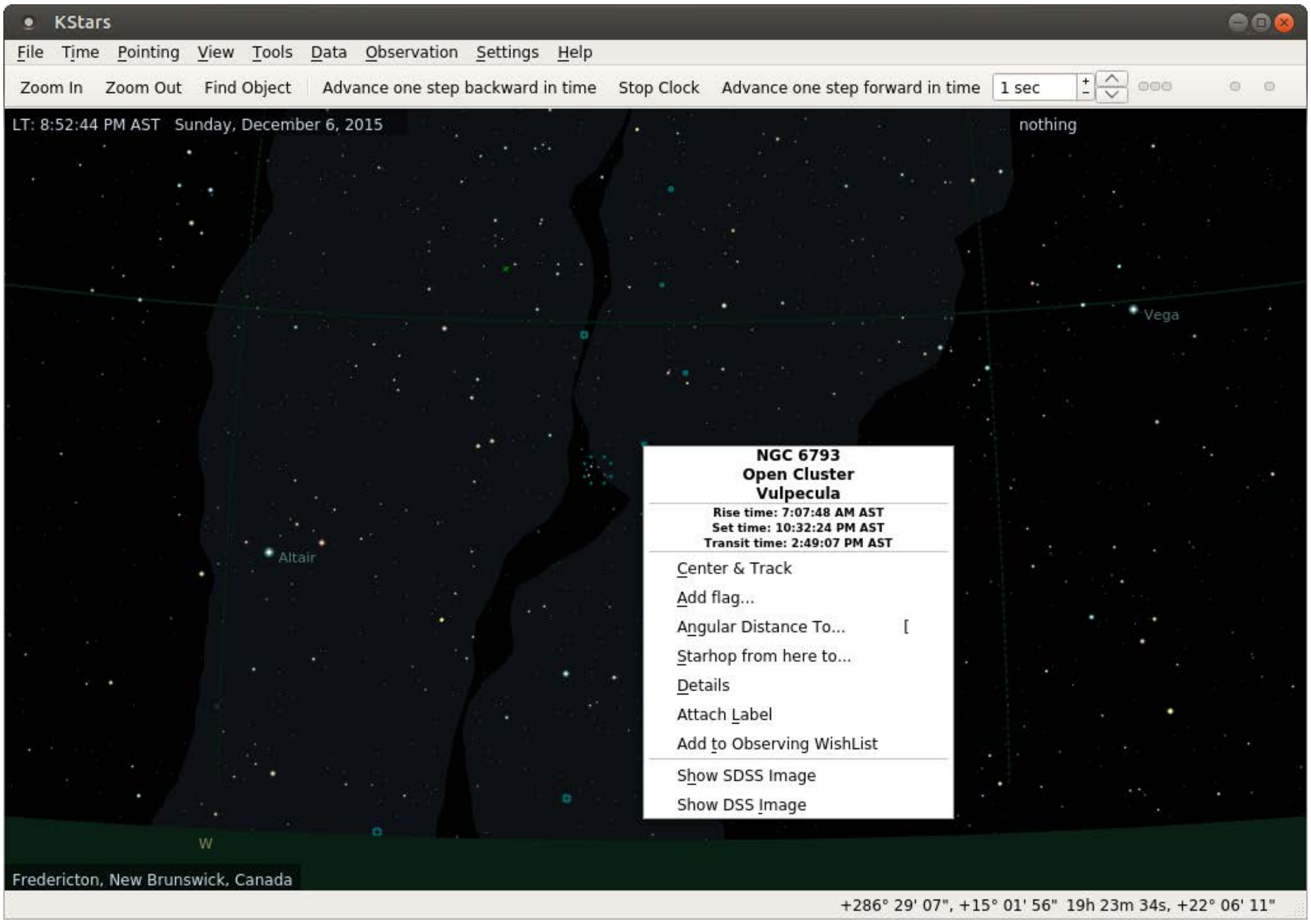
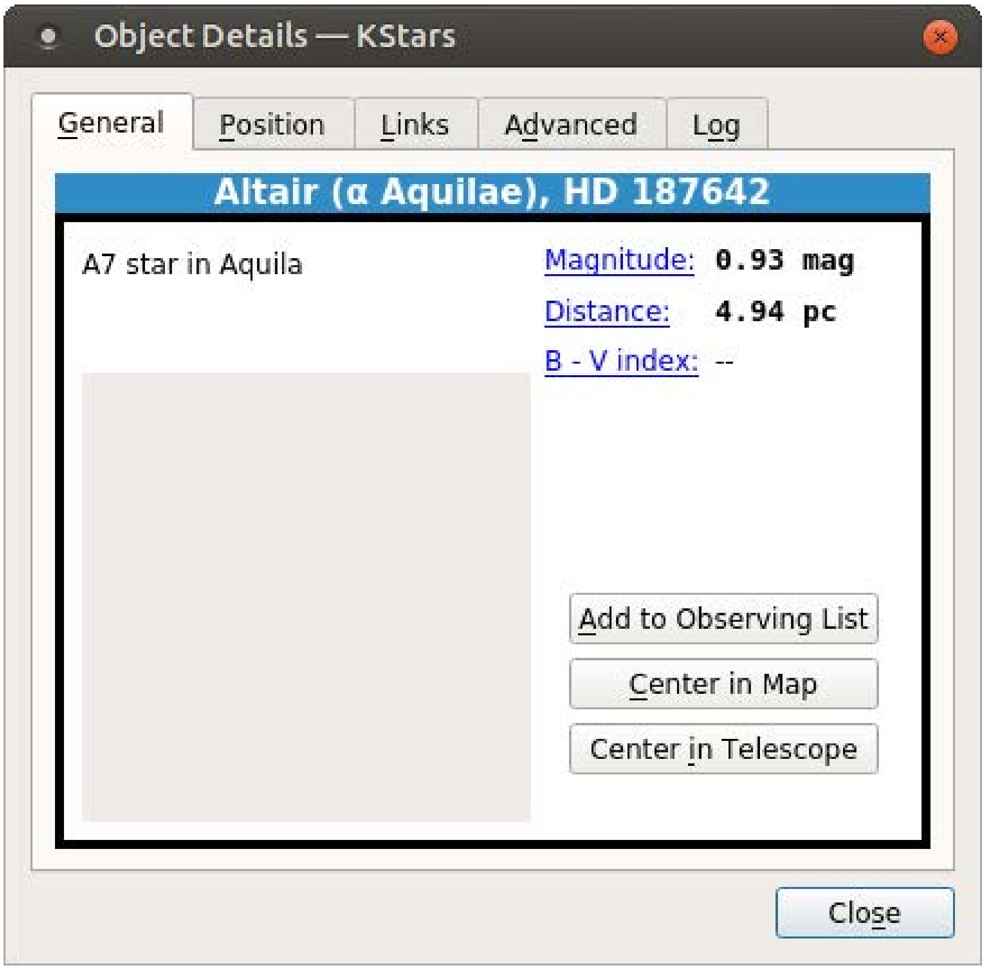

KStars
File Time Pointing View Tools Data Observation Settings Help
Zoom In Zoom Out Find Object Advance one step backward in time Stop Clock Advance one step forward in time 1 sec
LT: 8:52:44 PM AST Sunday, December 6, 2015
nothing
Vega
Altair
W
NGC 6793
Open Cluster
Vulpecula
Rise time: 7:07:48 AM AST
Set time: 10:32:24 PM AST
Transit time: 2:49:07 PM AST
Center & Track
Add flag...
Angular Distance To...
Starhop from here to...
Details
Attach Label
Add to Observing WishList
Show SDSS Image
Show DSS Image
Fredericton, New Brunswick, Canada
+286° 29' 07", +15° 01' 56" 19h 23m 34s, +22° 06' 11"

Figure 3. You can interact with the objects in the display.

on your zoom level. Two buttons at the top of the display allow you to zoom in and out. Beside these is a third button, labeled Find Object, that you can click on to get a search window. This window can use filters to search for particular types of objects, like comets or asteroids, or you can search through all of the objects that KStars knows about.

Once you find an object of interest, you can click on it and have the display move around until the object is in the center of the display. If it is below the horizon, a warning box will pop up asking if you still want to re-center the display. Once you have selected your object, you can right-click on it to get a drop-down menu of things you can do with it.

The header of this drop-down contains the full name of the object, along with rising and setting times.


Object Details — KStars

General | Position | Links | Advanced | Log

Altair (α Aquilae), HD 187642

A7 star in Aquila

Magnitude: 0.93 mag
Distance: 4.94 pc
B - V index: --

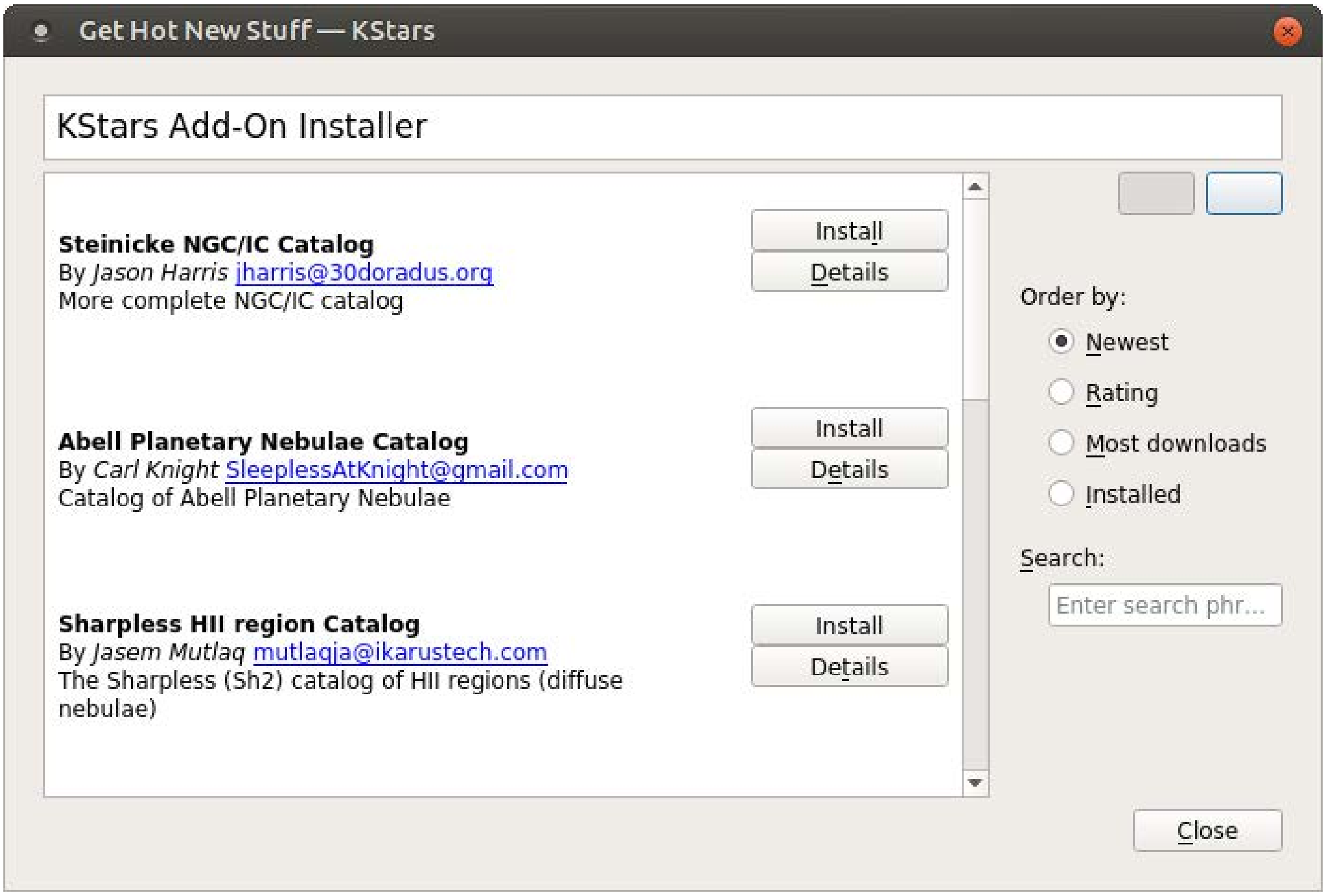
Figure 4. The detail window has a lot of extra information available for most objects.

Below that, the first option is to center and track the object. This is useful because the default display mode is to have the display updated in real time. This way, you always have a view of what the sky looks like right now. You can select the details option to pull up even more information on the object.

You can add a flag to make the object easier to keep track of or even add tracks as the object moves across the field of view. You also can calculate the angular distance to another object or plan out a star-hopping path to some other object.

For some objects, there may be images from the Deep Sky Survey available. If they are, there will be options on this drop-down menu labeled "Show DSS Image" or "Show SDSS Image".

Although KStars comes with quite a bit of data when you install it, this


Get Hot New Stuff — KStars

KStars Add-On Installer

Steinicke NGC/IC Catalog
By *Jason Harris* jharris@30doradus.org
More complete NGC/IC catalog

Abell Planetary Nebulae Catalog
By *Carl Knight* SleeplessAtKnight@gmail.com
Catalog of Abell Planetary Nebulae

Sharpless HII region Catalog
By *Jasem Mutlaq* mutlaqja@ikarustech.com
The Sharpless (Sh2) catalog of HII regions (diffuse nebulae)

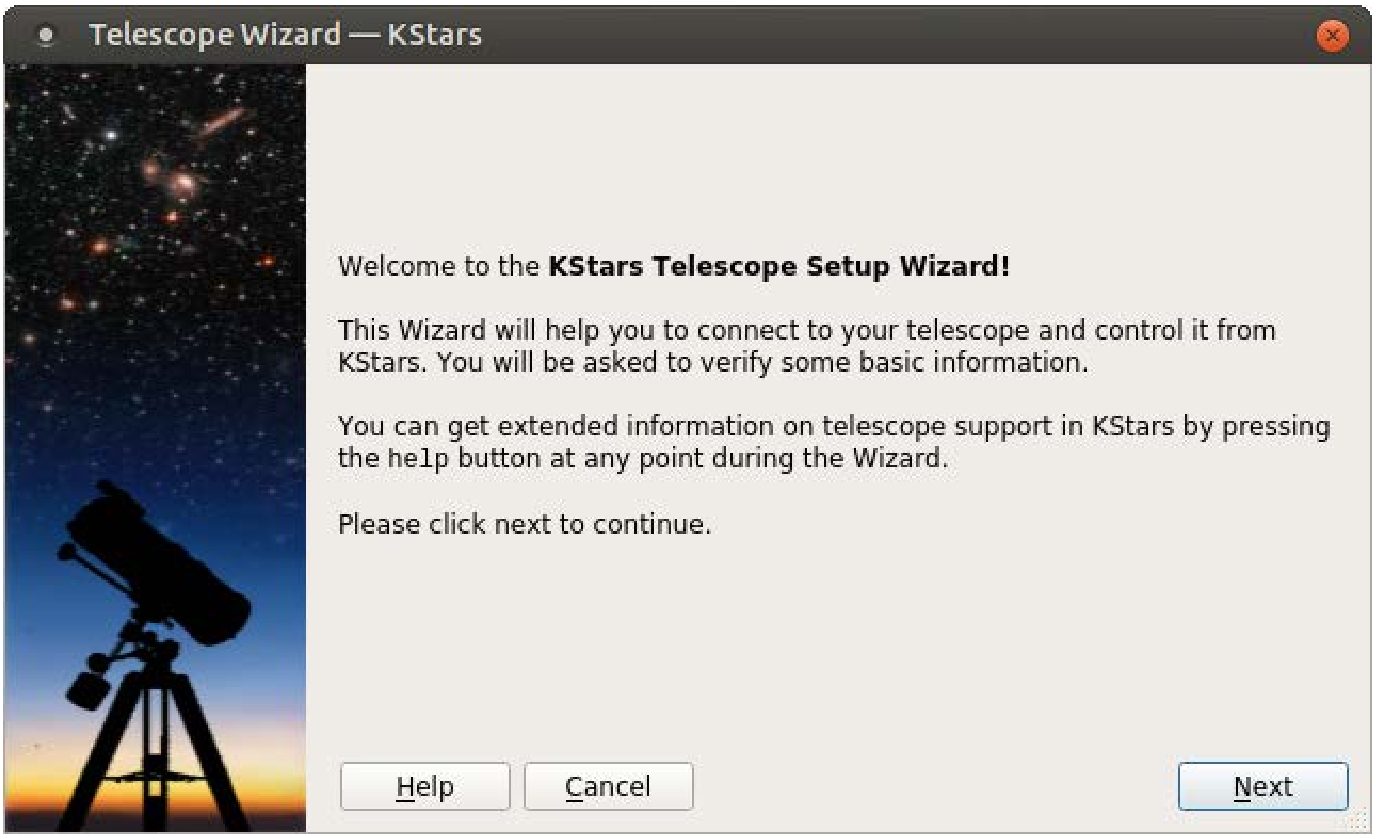
Order by: [✓ Newest] Rating Most downloads Installed

Search: Enter search phr...

Figure 5. Several extra data sets are available that you can download and install.

isn't everything that is available. You can add new data sources by clicking on the menu item Data→Download New Data. This will open up a new dialog window giving you a list of what data catalogs are available to download.

For some of the data sets, like comet and asteroid information, there is a constant updating of the detailed information available to the astronomical community. You can download those updates by clicking on the menu item Data→Updates→ and selecting which catalog to update. You even can import your own data by going to the configuration window and clicking the Import Catalog... button on the Catalogs section. When you are in the configuration window, you can see that you also can change settings on how to view several different categories, such as solar system objects, satellites and supernovae.



Welcome to the **KStars Telescope Setup Wizard!**

This Wizard will help you to connect to your telescope and control it from KStars. You will be asked to verify some basic information.

You can get extended information on telescope support in KStars by pressing the help button at any point during the Wizard.

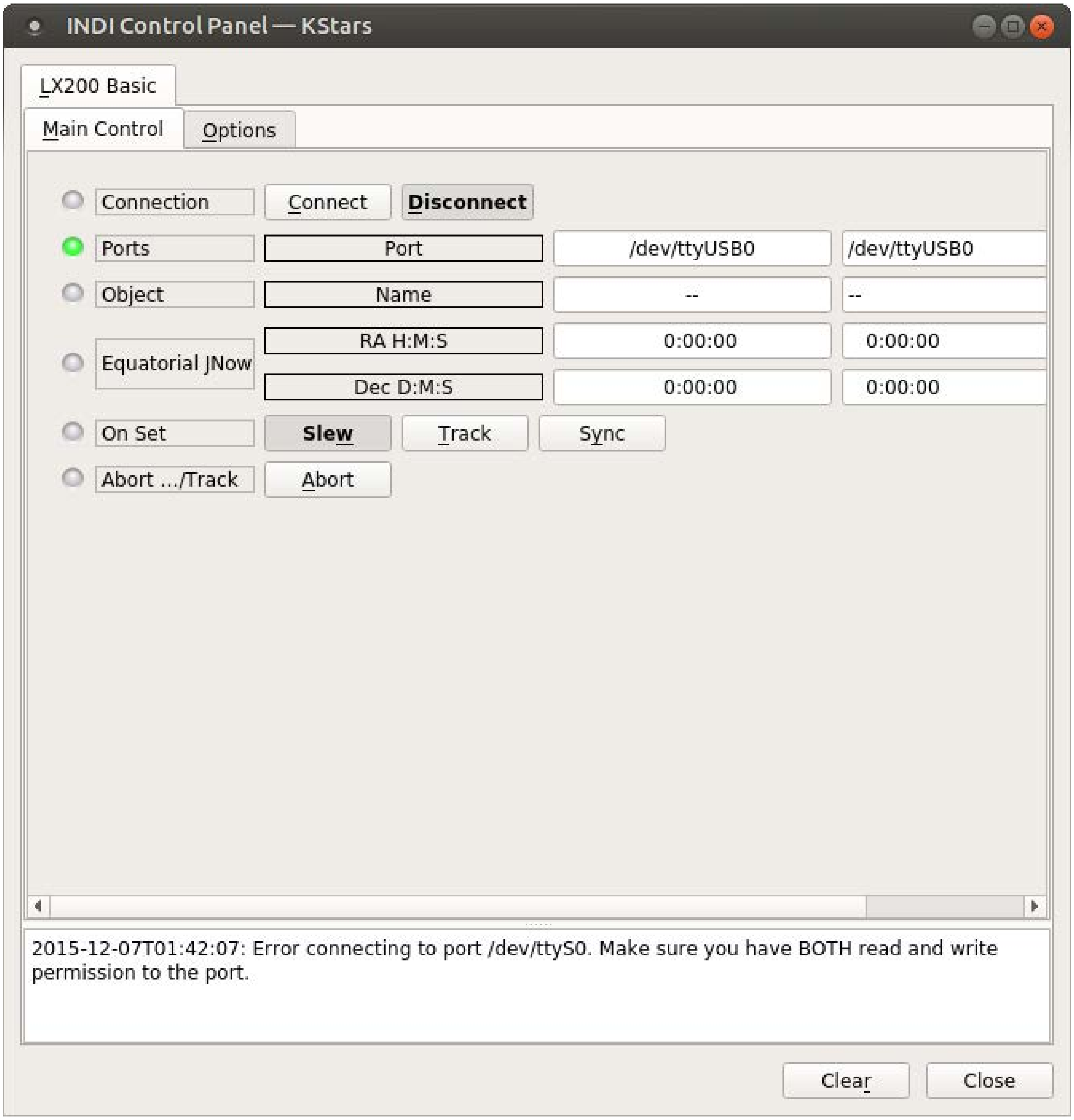
Please click next to continue.

Figure 6. You can add a telescope as a device under the control of KStars.

The guides section defines what extra information is displayed, such as constellation lines, names and the Milky Way. The INDI (Instrument Neutral Distributed Interface) section controls how KStars talks to your connected telescope. You can pull up the telescope wizard by clicking on Tools→Devices→Telescope Wizard.

Once the connection is made, you can pull up the control panel and send instructions to your telescope. You can even automate your observations using this functionality. Clicking on the menu item Observation→Observation Planner will pop up a window where you can define a complete set of observations you want to run. If it is a bit confusing at first, a wizard is available to help walk you through setting up a plan.

While you will make your own observations, you also may want to look at observations made by other researchers. The file format most often used is the FITS format (Flexible Image Transport System). KStars includes a FITS viewer, which has a number of analysis tools.



```
2015-12-07T01:42:07: Error connecting to port /dev/ttyS0. Make sure you have BOTH read and write
permission to the port.
```

Figure 7. Once it is connected to your computer, you can use KStars to control your telescope.

You can open a FITS file by clicking on File→Open FITS.

Several tools are available to work with the data in the image. You can look at the basic statistics of the image, including the width, height, maximum,

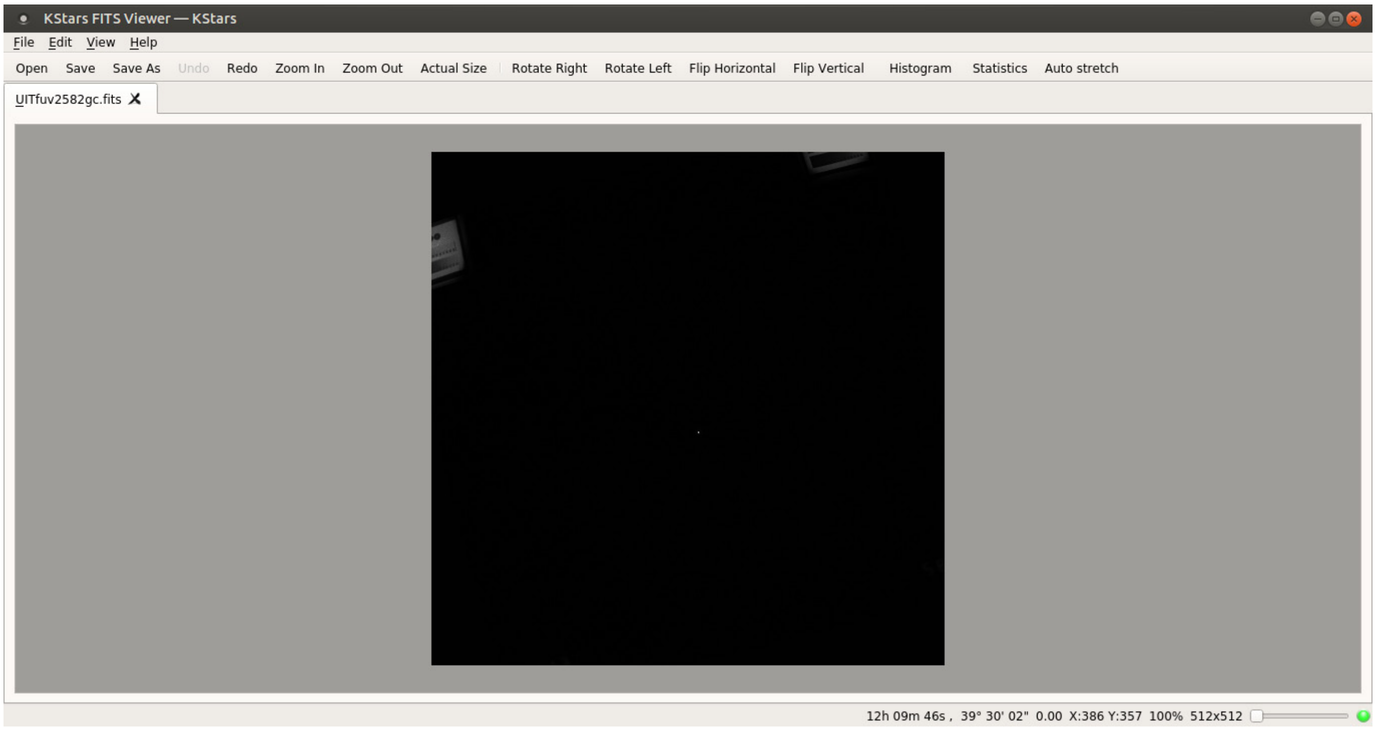

KStars FITS Viewer — KStars
File Edit View Help
Open Save Save As Undo Redo Zoom In Zoom Out Actual Size Rotate Right Rotate Left Flip Horizontal Flip Vertical Histogram Statistics Auto stretch
UITfuv2582gc.fits
12h 09m 46s , 39° 30' 02" 0.00 X:386 Y:357 100% 512x512

Figure 8. The FITS viewer lets you analyze and manipulate observational images.

minimum and mean of the pixels. You can pull up a histogram, showing the frequency spread of the image data. The View menu item includes several more tools where you can equalize the image, pass it through a high contrast filter or apply an auto stretch. There is even an entry to mark any stars that KStars can identify automatically.

You now have another astronomy program in your toolkit that you can use when your scientific research moves in that direction. In the coming months, I plan to look at other scientific software packages within this desktop environment and see just how much research can be done with KDE.—JOEY BERNARD

Poppins

My friend and fellow *Linux Journalian* Kris Occhipinti recently posted a reminder on Facebook for everyone to back up regularly in 2016. Although it's something we already should be doing, if you're not a regular backer-upper, you should start today! The method of backup isn't nearly as important as the act itself, but this month, I found a new project that simplifies the backup process nicely.

Poppins is an open-source project that builds on the SSH and rsync programs to create an incremental backup system that is simple, fast and reliable. Tons of other backup programs are available, but Poppins doesn't try to be a full-blown system; rather, it's a simple one-liner that will do file rotation, snapshots and more. It can be automated with cron, or you can run it manually from the command line. (But you should really, really make a cron job!)

Other backup systems have more robust interfaces, restoration options and so on, but the beauty of Poppins is its simplicity. It's just one step above manually running rsync yourself, but that one step means it's simpler to do. In my world, simple is about the only way to make sure something gets done at all!

In fact, even though it's a new project and still in beta, Poppins gets this month's Editors' Choice award. It's simple enough that you might actually get around to using it, and it has enough features to really benefit you in the case of a catastrophic failure. Check it out today at http://poppinsbackups.wordpress.com, or go right to the Bitbucket page: http://bitbucket.org/poppins.

—SHAWN POWERS

A Technical Conference Exploring Open Tech and the Open Web

GREAT WIDE OPEN

March 16 & 17

Downtown Atlanta

In the heart of Technology Square
and Georgia Tech University

www.greatwideopen.org

SOME OF THE TOP TECHNOLOGISTS IN THE WORLD WILL BE FEATURED:

Kelsey Hightower
Developer Advocate
Google Cloud Platform

Danese Cooper
Distinguished Member/CTO
PayPal/Wikimedia Foundation

Chris Van Tuin
Chief Technologist/ Western
Red Hat

Steve Klabnik
Developer/Author
Mozilla

Erica Stanley
Founder
Acire Studios, Women Who Code - Atlanta

Trek Glowacki
Core Team Member
Ember.js

REUVEN M. LERNER

Tech Book Roundup

Looking to sharpen your professional skills? Here's Reuven's latest roundup of useful books to improve your knowledge and career.

In the computer industry, you don't have a choice—you must constantly keep learning new things, just to stay in place. I spend many hours each week reading blogs, watching conference videos and just exploring technologies on my own, because if I don't do that, I'm going to find myself unable to use the latest technologies to help my clients.

But of course, there is at least one other tried-and-true way to learn: books. Sure, books (or their electronic equivalents) have been eulogized numerous times during the last few years. But, programming-related books continue to be published, some by well known publishers and others by independent programmers.

Every so often, I take time in this column to review the latest crop of books that have helped make me more effective in my work. As always, the descriptions and reviews of these books (and other media) are not only subjective in terms of the content, but also in terms of the timing; it's quite possible that I only recently discovered a book that came out a while ago. I try to mention books here that you might find interesting or useful in your work, much as I've found them to be interesting and useful in my own work.

Perhaps the most interesting trend we're now seeing in the book market, and especially when it comes to computer books, is the number of self-published ebooks being produced. (Heck, I've written two of them myself: one on Python and another on regular expressions.) These books rely on word of mouth, social media and marketing by the developers/authors themselves.

There's no doubt that traditional publishers continue to have great reach and are producing more books

But, if you're trying to teach children to program and are agnostic on the language used, I'm not sure if Ruby is the right way to go.

(and well edited books, at that) than independent developers ever can hope to match. But the number of self-published ebooks continues to grow, as well as those sold via sites like https://leanpub.com. It will be interesting to see what happens in the coming years as book-creation systems, such as GitBook and Softcover, make it even easier to create and sell your own books.

So, what have I been reading, and what would I suggest you read as well?

Programming Books

I continue to enjoy reading programming books, partly because there's always something new to learn, partly because different authors offer different perspectives, and partly because I often can pick up insights and tricks that I can incorporate into my courses. I continue to use Ruby on Rails for Web development, so I always am happy to look at Ruby-related books.

One of my most favorite Ruby books is *Metaprogramming Ruby* by Paolo Perotta and published by the Pragmatic Bookshelf. The book has been updated to include Ruby 2, and although Ruby's underlying object system hasn't changed in many years, every version of Ruby is a bit different—and knowing how to take advantage of those differences can make a huge difference in your productivity. If you are using Ruby in your work, you really should read this book.

Teaching children to program always has been a passion of mine, so I'm always interested in books and software that try to teach kids how to program. Several years ago, I mentioned the book *Python for Kids*, and it shouldn't be much of a surprise to discover that there's now a book for kids to get into Ruby, called *Ruby Wizardry*. I think I should distinguish between two different goals here though. If you really want your children to learn Ruby, this book is an entertaining way to do so, with numerous examples and a storyline I'd like to think kids would enjoy. But, if you're trying to teach children to program and are agnostic on the language used, I'm not sure if Ruby is the right way to go. (See below for my recommendation of an on-line system that seems to be doing that right.) My limited experience with trying to teach Ruby, or even Python, to children has been fairly lackluster—not because the materials are bad, but because kids are less interested in the language than in what they can do with the language.

I have been using a great deal of Python in recent years and also have been teaching a huge number of Python classes around the world. I'm always looking for ways to explain advanced aspects of Python better; thus, Obi Ike-Nwosu's self-published *Intermediate Python* ebook caught my attention. I found that many aspects of this book put ideas into nice context and into a focus that otherwise might have been lost. If you have been working with Python for more than a few months and want to deepen your knowledge and understanding on a few topics, this book seems like a good place to go.

Finally, I continue to work a great deal with PostgreSQL, a database whose features and community continue to grow in size and quality with every passing year. Hans-Jurgen Schonig, a very accomplished PostgreSQL consultant and contributor, has written *Troubleshooting PostgreSQL* (published by Packt), a book that is something like a first-aid kit for PostgreSQL developers and DBAs. He touches on all of the sensitive topics, including indexing, transactions, locking, monitoring, replication and backups. Even if you have been using PostgreSQL for some time, *Troubleshooting PostgreSQL* is an excellent investment, one that will more than pay for itself in time saved and hair (not) pulled.

Data Science

It was probably somewhat inevitable as someone who has worked with

Even if you have been using PostgreSQL for some time, *Troubleshooting PostgreSQL* is an excellent investment, one that will more than pay for itself in time saved and hair (not) pulled.

databases, uses Python and has an interest in statistics that I would slowly but surely slide into data science. I've thus been spending time during the last year, and especially the last few months, learning more and more about this fascinating subject—both so I can help my clients solve data-science problems and so I can offer training in the use of Python for data science. There has been an explosion of books about data science in the past few years, and I've been working my way through no small number of them.

Two high-quality books about data science are *Doing Data Science* by Cathy O'Neil and Rachel Schutt, published by O'Reilly, and *Data Science from Scratch* by Joel Grus, also published by O'Reilly. Grus uses Python and pushes hard in the use of functional programming techniques in Python, along with the standard data-science stack available to Python developers, such as matplotlib and pandas. But, he also provides no small amount of insights into how—and, just as important, when and why—we should use such techniques as linear regression vs. multiple regression vs. k-nearest neighbors. The book contains a very large number of code examples, and because it's written in Python, it's likely to use paradigms with which Python developers already are familiar, such as iteration and comprehensions. The former book, which I have recommended in earlier book-review columns, gives a serious introduction to the how and why, not only including code, but also the mathematical ideas behind that code.

Another book on this topic, currently in early release from O'Reilly, is *Python Data Science Handbook* written by Jake VanderPlas. If *Doing Data Science* takes a mathematical approach, and *Data Science from Scratch* takes a functional-programming approach, then this book takes what I'd call a practical approach—introducing the elements of Python that a budding data scientist most needs to understand, keeping the programming and mathematics at a reasonable level for people who are unfamiliar with either or both. I'm still reading through this early-release edition, but as an introduction to learning science with a relatively easy learning curve, it may soon become my most-recommended book for people new to the subject.

Ivan Idris has written two books for Packt Press that are good overall introductions to data-science tools in Python, namely *Python Data Analysis* and *Python Data Analysis Cookbook*. The former introduces the topics and (as with the book by Grus, mentioned previously) provides a tutorial into areas like visualization, time series and machine learning. The cookbook is a complementary volume; instead of teaching based on subjects, it takes a problem-solution approach. I'd suggest that reading the tutorial first is a good idea; you then can read through the cookbook's table of contents and/or index to find the specific issues on which you need to work.

If you're interested in data science, but would rather hear about the insights and perspective than the mathematics, read *Dataclysm: Who We Are (When We Think No One Is Looking)* by Christian Rudder, one of the founders of OKCupid. In this book, Rudder finds fascinating correlations that provide insights into human behavior—thanks to the huge number of data points collected by the dating site he runs.

I would be remiss if I were to ignore the amazing e-mail list produced by Analytics Vidhya, a name that doesn't quite roll off of the tongue of English-speaking Americans, but which offers daily insights for anyone interested in the technology, algorithms and practice of data science: http://analyticsvidhya.com.

Other Subjects

I might be a software developer, but I also run a small company, so I'm always looking for books that can help me run my company even better or find new business opportunities.

One book, published just before I wrote this column, comes from a group of consultants with a Slack channel. They jointly wrote and published *The independent consulting manual*, an ebook meant for people who are consulting, or want to consult, but aren't sure where to start, how to find clients, how to price themselves or how to create products. The book (at http://independentconsultingmanual.com) has many famous contributors, all of whom have made the leap to running their own businesses, and the lessons they've learned, and contributed, make for extremely worthwhile reading.

One of the authors of the above ebook, and a co-panelist of mine on the Freelancers Show podcast, is Philip Morgan. Morgan has done a lot to encourage consultants to position themselves—that is, to define a specific niche in which they can be the best-known experts. His book, *The Positioning Manual for Technical Firms* (available at http://ThePositioningManual.com) is full of excellent advice, including

In *What If*, Munroe asks crazy hypothetical questions, but then proceeds to answer them with the perfect balance of science and humor.

many reminders that although advertising yourself in a specific niche might seem like a way to lose money, it's actually the ticket to making more and to becoming a more successful consultant.

Of course, I read about other subjects too. If, like me, you're interested in the intersection of history, science and food, you should read the book *Consider the Fork* by Bee Wilson, which describes the evolution (and diversity) of kitchen utensils and gadgets. Wonder why China and America have different-shaped spoons or how forks and the shape of our teeth are related? This book describes it all. I was introduced to this book by the great "Gastropod" podcast, which talks about these subjects in great depth.

Another great book that I read this year is *What If* by Randall Munroe, the author of XKCD. In *What If*, Munroe asks crazy hypothetical questions, but then proceeds to answer them with the perfect balance of science and humor. I read the first part of this book while eating at a restaurant, and I feel bad for the other patrons who had to endure me laughing quite loudly at the absurdities in this book, mixed together with (of course) great illustrations and explanations.

Finally, my children are getting older, so I have less and less of a chance to read to them. However, I did manage to read *The Terrible Two* by Mac Barnett and Jory John, a beautiful and funny story about pranks (or "hacks", as we called them during my undergrad days at MIT), friendship and the all-important goal of making a school principal look foolish in front of the students.

Podcasts and Other Resources

I continue to listen to a very large number of podcasts each week, partly because I enjoy them so much and partly because they're so convenient for someone like me, who often commutes by bus or train to clients' offices.

Two new Python-related podcasts have emerged in the last year: "Talk

Python to Me" and "Podcast.__init__" are both interesting, with great interviews and many insights into technologies that I either use every day or should start to learn.

Other podcasts that I enjoy tend to be on the subjects of politics and economics—from Slate's "Political Gabfest" and "Slate Money" to NPR's "Planet Money" to "Podcast for America", an irreverent look at the US political race. But there are many other great podcasts to try, such as the BBC's "More or Less" about statistics, "Functional Geekery" about functional programming, "Freelancer Transformation" about how to become a successful consultant and "Startup", which is all about what it's like to run a venture-based startup company.

I mentioned earlier my interest in helping children learn to code, and I've discovered (thanks to my ten-year-old son) a terrific site that does just that: Code Monkey (at http://PlayCodeMonkey.com) uses CoffeeScript of all things to teach programming to children. However, it does so using inventive and creative graphics, high-quality error messages and a staged approach that does a very good job of introducing programming to kids aged 8–12. In my son's case, his school paid for entry into all of the levels, but I would gladly have paid the $30 that Code Monkey charges to do all of the levels it offers, as well as build some of my own. It's rare for me to say that I have seen an educational technology that lives up to its hype, but this is definitely one of those times.

Finally, let me mention a site that I run called Daily Tech Video (http://DailyTechVideo.com), which offers a new, high-quality conference video every day. If you're interested in open-source technologies, programming languages, databases and even some computer history, I invite you to take a look at my site and even to suggest high-quality conference talks I might have missed. You also can follow the site's update on Twitter at @DailyTechVideo.■

Reuven M. Lerner trains companies around the world in Python, PostgreSQL, Git and Ruby. His ebook, "Practice Makes Python", contains 50 of his favorite exercises to sharpen your Python skills. Reuven blogs regularly at http://blog.lerner.co.il and tweets as @reuvenmlerner. Reuven has a PhD in Learning Sciences from Northwestern University, and he lives in Modi'in, Israel, with his wife and three children.

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

O'REILLY®

Fluent

CONFERENCE

THE WEB PLATFORM

March 7–10, 2016
San Francisco, CA

Practical training in JavaScript, HTML5, & web development.

Save 20%

Register today.

use code PCLinuxJournal

DAVE TAYLOR

Working with Command Arguments

getopt—the right way to parse starting arguments in your shell script.

In my last article, I went interstellar and created a multi-planet time translator, allowing you to learn just how old you'd be if you lived on Saturn or Pluto. So in this article, I want to head back to a more fundamental aspect of shell scripting: working with command arguments.

I suspect that most shell scripts go through an evolution with their command flags, actually starting out with none, then maybe one or two parsed in a sloppy fashion, finally upgraded to a proper implementation of `getopt`, and then, perhaps, even being completely rewritten as a C++ or Ruby program as the complexity keeps inexorably increasing.

The really easy way to parse a starting flag, of course, is just to use a conditional statement:

```
if [ "$1" = "-a" ]; then
  flaga=1
fi
```

There are a couple problems with this approach, however, not the least of which is that it ends up taking a fair amount of space in the code. Because to be proper, the full sequence also would include `flaga=0` before the conditional, because (repeat after me) you never can assume that the shell will correctly instantiate the value of a newly defined variable.

The other problem is that after this code sequence, the command parameters are out of sync: `$1` is still either possibly the starting flag value

Where I always get tripped up is that $# is the number of arguments, not the total number of words in the command.

(`-a`) or another argument or value that the user has specified.

To make this a bit more logical, let's imagine a shell script that's a wrapper to something like the terrific `curl`: give it a URL, and it'll grab that content and save it to the local directory as a file. Add the imagined `-a` flag, and it'll also include progress information. So, a typical usage might look like this:

```
getpage.sh -a http://www.linuxjournal.com/index.html
```

In this situation, a number of command parameter variables are going to be instantiated as the shell script is invoked, notably `$# = 2`, `$0 = getpage.sh`, `$1 = -a` and `$2 = http://www.linuxjournal.com/index.html`.

Where I always get tripped up is that `$#` is the number of arguments, not the total number of words in the command. Therefore, if this script is invoked without any arguments at all, `$#` seems like it should be 1 (what about the command name?), but actually it's 0.

This goes back to the dawn of UNIX development actually, and it's known as the "0 index problem". In arrays, the first value is referenced as being in slot 0, or array[0]. For some developers, that makes complete sense, and for others, it can be confusing. Indeed, I've seen C programs where the writer ignores the first slot entirely and indexes starting at 1, not 0.

With that in mind, after the conditional has been used to check for the `-a` flag in the first variable space ($1), what really needs to happen is that all the positional variables above this one (for example, $2, $3 and so on) need to shift down a spot. Ideally, after the conditional, $1 contains the URL value regardless of whether the starting flag was specified.

That way, the next statement block in the script safely can assume that $1 will be the URL and not have to test redundantly to see if it's still `-a`.

This is done with the `shift` command, and so, here's the proper way to test conditionally for an optional variable in a shell script:

```
flaga=0
if [ "$1" = "-a" ]; then
  flaga=1
  shift
fi
```

That works exactly as you'd hope, but leads to the next question: what happens if the flag has an optional value that the user can specify? In the case of this curl script, perhaps the flag is something like -o output file.

That's actually an easy addition to the above code:

```
outputspecified=0
if [ "$1" = "-o" ]; then
  outputspecified=1
  outputfilename="$2"
  shift 2
fi
```

As you can see, `shift` takes a single numeric argument that reflects how many slots you want to have everything cascade. It's easy to see if you consider the positional values before and after the above code block:

```
$ sh getpage.sh -0 test.html SomeURL
$# = 3
$1 = -a
$2 = test.html
$3 = SomeURL
-----
$# = 1
$1 = SomeURL
$2 =
$3 =
```

Initially, all three positional variables are set, and the arg count ($#) is 3, as makes sense given the command invocation. But after the `shift 2`, everything's moved down two slots, and the arg count also is decremented two, as you can see above.

Now, what if your development of the script hits a juncture where you realize that it'd be useful to have three different starting arguments, one of which indeed takes an argument?

Dealing with Lots of Starting Flags

Even two flags can be a pain, because it'd be terrible code that would force the user to specify them in order, but without that, -a, -c and -o could require you to test for all three, parse and shift, then test for all three again, parse and shift, then test for all three one more time. That would be a nightmare, and I haven't even mentioned how those pesky users

The `getopt` command is going to become your best friend if you're building complex user-facing scripts, no question.

have a tendency to combine flags too, so would your script properly parse `-ac -o` or `-oc -a` too?

Enter `getopt`.

The `getopt` command is going to become your best friend if you're building complex user-facing scripts, no question. It basically works by extracting all the optional flags and parameters, then lets you parse through them in a uniform manner.

The standard usage is to break down combined args with an invocation of `getopt`, then use the `set` command to replace the existing starting flags with the new, neater args, then run through a loop, parsing them one by one.

Yes, this is easier just to demonstrate, so let me show you the command flag parsing segment from a different production script. This particular command has three possible starting flags, `-n`, `-p` and `-t`.

The first step in the script is to have `getopt` normalize whatever the user has specified:

```
args=$(getopt np:t $*)
```

Comparing this statement with the usage error below, you can see that arg 1 to `getopt` is a list of all acceptable flags, with : denoting those flags that have a required additional argument if the flag is specified—easy enough.

Now the status variable `$?` can be tested: if it's non-zero, there was an error in parsing the flags, and in most scripts, it's time to fail out with a usage statement:

```
if [ $? != 0] ; then
  echo \
    "Usage: $(basename $0) {-p SFX} {-n} {-t} PTN NEWPTN"
  echo "    -n   sequentially number matching files"
  echo "    -p   use specified suffix SFX for filenames"
  echo "    -t   test only - don't execute resultant cmds"
  exit 0
fi
```

You'll often see scripts that have the usage sequence pushed into a separate function to keep the code clean. Also, note the use of `$(basename $0)` in the first echo. That's a handy trick to compensate for the fact that most of the time `$0` is going to be the full name of the

Now all the positional parameters are neatly organized and ready to parse, something that's traditionally done with a case statement wrapped in a for loop (no difference from an enigma wrapped in a dilemma, of course).

script, including path. So tapping basename is just for aesthetics!

Finally, the statement that does the real work:

```
set -- $args
```

Now all the positional parameters are neatly organized and ready to parse, something that's traditionally done with a case statement wrapped in a for loop (no difference from an enigma wrapped in a dilemma, of course). It looks like this:

```
for i
do
  case "$i"
  in
   -n ) renumber=1 ; shift ;;
   -p ) fixpng=1   ; sfx=$2 ; shift 2 ;;
   -t ) doit=0     ; shift ;;
 esac
done
```

I have a particular style with the semicolons in my case statements, mostly just to ensure that I use the needed `;;` sequence to terminate each of the individual conditionals, but otherwise, it should be easy to understand.

The only thing missing from this code fragment is something I alluded to earlier. What is it?

And, that's it for this article. Now, go back to your latest shell script, and just for practice, go ahead and add some optional starting flags and parse them with `getopt`.■

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

13th Annual

2016 HPC for Wall Street – Cloud & Data Centers Show & Conference

April 4, 2016 (Monday)

Roosevelt Hotel, NYC

Madison Ave and 45th St, next to Grand Central Station

Plan to attend. Low-Cost Conference Program and Free Show.

2016 Capital Markets are coming to the 2016 HPC for Wall Street.

All-Star Conference program for 2016.

Plan to attend the largest meeting of HPC, Cloud, Big Data, Data Centers, Virtualization, Low Latency for the Capital Markets.

See the program from 2015.

The 2016 program will have the same all-star lineup of speakers.

Location. Location. Location. The Roosevelt is next to Grand Central Station and within walking distance of JPMorgan Chase, Deutsche Bank, Morgan Stanley, NASDAQ – all in midtown.

Register online today: www.flaggmgmt.com/linux

2015 Sponsors

IBM, Cisco, redhat, hp, lenovo, ORACLE, Juniper Networks, ARISTA, NYC NewSQL New York, Linux Journal, HPC wire, TRADERS, datanami, Linux New Media The Pulse of Open Source, Money management executive, EnterpriseTec, Integration developer news, WSTA Wall Street Technology Association

www.flaggmgmt.com/linux

Show Hours: Mon, April 4 8:00 - 4:00
Conference Hours: Mon, April 4 8:30 - 4:50

Show & Conference:
Flagg Management Inc
353 Lexington Avenue, New York 10016
(212) 286 0333 fax: (212) 286 0086
flaggmgmt@msn.com

The all-star lineup of speakers from HPC 2015

Dave Weber Global Financial Services Director, Lenovo

Ken Barnes SVP Corp Dev, Options Information Technology

Bernard S Donefer Associate Director, Baruch College

Mike Blalock Global Sales Director, Intel

Andy Bach Chief Architect, Financial Service, Juniper Networks

Jeffrey M. Birnbaum Founder and CEO, 60East Technologies

Dino Vitale TD Securities

Harvey Stein Head of Credit Risk Modeling, Bloomberg

Fadi Gebara Sr Manager, IBM Research

Terry Keene CEO, iSys

Rob Krugman VP Digital Strategy, Broadridge Fin Sols

Lee Fisher VP Marketing, Redline Trading Solutions

Jeremy Eder Perf Engineering, Red Hat

Matt Smith Sol Architect, Red Hat

David B. Weiss Sr Analyst, Aite

Rick Aiere Architect Specialty, AIG

Shagun Bali Analyst, TABB Group

Jeffrey Scheel Senior Technical Staff, IBM Linux Tech Center

Ed Turkel Mgr WW HPC Mkting, Hewlett-Packard

Charles Milo Enterprise Technical Specialist, Intel

Alex Tsariounov Principal Architect - Adv. Platforms, London Stock Exchange

Ugur Arslan Quantative Analyst

Davor Frank Sr Solutions Architect, Solarflare

Phil Albinus Editor, Traders Magazine, SourceMedia

David Malik Sr Director, Advanced Services, Cisco Systems

Russ Kennedy SVP of Product Strategy, Cleversafe

Ryan Eavy Exec Dir, Architecture, CME Group

Markus Flierl VP Software Dev, Oracle

Nick Ciarleglio Distinguished Syst. Engineer, FSI Product Mgr Arista Networks

SHAWN POWERS

The Powers That Be

When the lights go out, do your servers too?

I live in an old house with old wires in an old town with a dated infrastructure. I've never really considered the quality of the electricity being pumped into my house, but since we moved to this charming house loaded with character (that's the nice way of saying "old"), it's something I've had to deal with often. I made a lot of assumptions and even made some expensive purchases that in hindsight were silly. So for this article, I figured I should talk about power, because if you're reading *Linux Journal*, you're familiar with things that plug in to the wall.

The Symptoms

The biggest red flag for me in the new house was my Internet connection behaving poorly. I've literally had the technicians out here more than two-dozen times trying to track down why I have massive packet loss. At first it was unusable (50% packet loss is something TCP/IP can't even error-correct), but after months of struggling, I have only minimal loss. Unfortunately, my day job requires me to have a solid Internet connection, so I've been forced to install two separate connections and provide failover support if (when!) one of them goes down. Although the Internet connections may be the most sensitive to electrical issues, I know that unstable voltage and noise can be bad for all my servers as well.

Quite honestly, even if you're not having Internet issues, there are some things to look for that might be symptoms of power problems. Now that I'm paying closer attention, I notice that a few times a day, our lights flicker or quickly dim. I assumed at first it was when a large motor kicked on (causing a sag) or something like that, but now I don't think it is. It appears to be unrelated to anything in the house, and it happens often enough that it's not just a fluke. If you notice tiny glitches

I've actually never had problems with EMI/RFI on server equipment, but if you have any audio equipment and you hear a buzz, or odd noises in your recordings, a line conditioner might really help.

in the Matrix, it might be a sign that your power fluctuates.

The Minimum Protection

Even if your house's electrical service is rock-solid, it's very important to have your electronic equipment on a surge protector. Although they used to be expensive, you can get a really nice one for less than $20 at most stores. A surge protector's job is to protect your equipment from current fluctuations, but only *extreme* fluctuations or surges (thus the name). A surge protector won't generally help with minor issues like those in my house, but during a lightning storm, you'll be glad you spent the money.

If you live in a place that rarely gets lightning storms, or if you just don't think a lightning strike is likely in your area, I still highly recommend a surge protector, or surge suppressor as they're often called. When I worked at a local school district, there was a failure on one leg of the three-phase power feed that caused a huge surge to enter the building. It was a sunny day without lightning, and still the surge destroyed thousands of dollars worth of non-protected equipment.

Conditioner, Not Just for Hair

The local cable technician recommended I purchase a power conditioner in order to give my equipment clean power. This actually started my research on power, and before I did much studying, I ordered a rack-mounted power conditioner for all my servers. I wish I had done more research before ordering it, but it does serve a purpose, and what it does, it does very well.

A power conditioner does a couple things. It protects your equipment from surges. In fact, a surge protector is a power conditioner of a sort; it's just the most basic kind. The other main purpose of a power conditioner is that it "cleans" the power signal by filtering out electromagnetic interference (EMI) and radio frequency interference (RFI) that can really mess

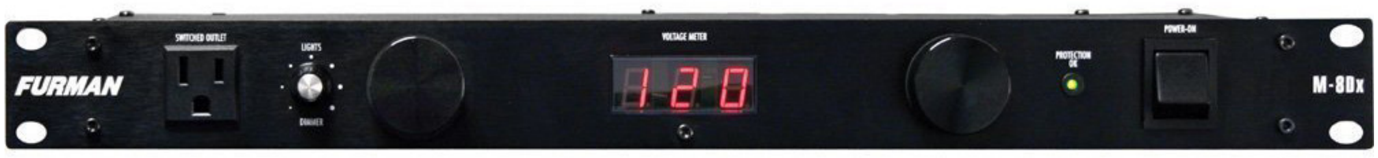


Figure 1. I love the readout on this line conditioner. I just wish it were a voltage regulator!

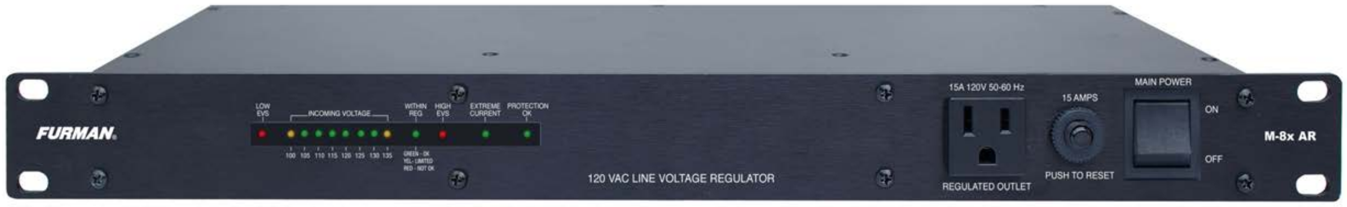


Figure 2. I don't regularly look at the front panel, so really this readout is sufficient. I just prefer the large LCD numbers showing incoming voltage.

with sensitive equipment. I've actually never had problems with EMI/RFI on server equipment, but if you have any audio equipment and you hear a buzz, or odd noises in your recordings, a line conditioner might really help.

The model power line conditioner I bought has a really cool display on the front that shows the line voltage. In fact, although it doesn't do anything to help with voltage fluctuations, it is nice to have the readout showing me as voltage comes and goes. Figure 1 shows my power conditioner. It's in my office now, but I can see the voltage vary anywhere from 103V to 124V throughout the day. That's a huge variance, and it's very likely what has been causing part of my Internet problem.

The Big Guns

Although the power line conditioner is great at identifying voltage issues, it doesn't actually do anything to fix them. Unfortunately, the tool to normalize voltage is quite expensive. I paid around $400 for a rack-mounted voltage regulator. The good news is that most voltage regulators not only regulate the voltage, but they also condition the line and protect from surges. (But you really should read the fine print to make sure.)

The voltage regulator on my server rack functions just like a power strip. It plugs in to the wall, and then I plug my devices in to the back. All the devices see a perfectly steady 120V power source, however, even if my actual house voltage gets wacky.

It's not a magic device, and it won't work in the case of a brown out, but it can handle voltage sags down to 95V, which is lower than I've ever seen it drop in my house.

The specific voltage regulator I purchased is the Furman M-8X AR (Figure 2). It supports up to 15 amps of service, which is all my server rack power line is rated for anyway. I do wish it had a clearer readout for incoming voltage like my line conditioner does (the line conditioner is the Furman M-8Dx), but that doesn't affect its performance, just my desire for data!

Sometimes, the Voltage Is Zero

That last thing I want to touch on here is a battery backup. There's a lot to be said regarding the type of battery backup you use, but I just cover them briefly here.

Standby UPS: This is the type of UPS most folks use, even in server rooms. There is a battery, which is kept charged by the main power line, and if the power goes out, it quickly powers an inverter to supply 120V current. This is very efficient, because the inverter doesn't constantly convert the battery power into AC, it only "does work" when the power goes out.

The downside of all this efficiency is that there's a small cutover time when the line power is disrupted before the inverter takes over. It is usually measured in milliseconds, and the more expensive the standby UPS, the quicker the cutover time. This is typically not a problem for most computer hardware, but particularly sensitive computers may reboot during switchover.

Standby-Ferro UPS: This type of UPS functions in the same way as a standby UPS, except there is a built-in transformer that regulates the voltage and provides a bit of a "buffer" during the cutover to inverter power. Since transformers use magnetism while converting voltage, there is a residual magnetism that usually smooths over the transition so that even sensitive equipment doesn't suffer.

Since they have built-in voltage regulation, they help with fluctuating voltage even when there's not a power outage. Unfortunately, they are known to overheat with some generators, especially those not generating pure sine waves. Still, with the buffering effect of the voltage regulation circuitry, Standby-Ferro units are usually preferred and usually cost significantly more than standard Standby UPS units.

Line Interactive UPS: These types

of backups provide the smoothest transition during an outage, because they constantly use their inverter to produce electricity. The only switchover is whether the inverter is taking power from the batteries or the wall current, and since both are connected, there's not even a millisecond of cutover time to worry about.

These are the most expensive types of UPS devices, and although they're very nice for sensitive equipment, they're often overkill for protection from the occasional outage. Still, if you're looking for the best of the best and price is no object, a Line Interactive UPS is the Rolls Royce of the UPS world.

My Cheap, Massive Solution

I have terrible luck with UPS devices. It seems like the batteries are always dead when our power goes out, and I have no idea until it's too late. And then, even the fairly expensive rackmount units are often disposable without the ability to change their batteries. Even if the batteries are replaceable, and you actually know they've gone bad, UPS devices don't last very long. Usually the best you can hope for is to keep your servers up long enough that they can shut down properly instead of just losing power. That's a great benefit, but it's frustrating if you want to function during an outage.

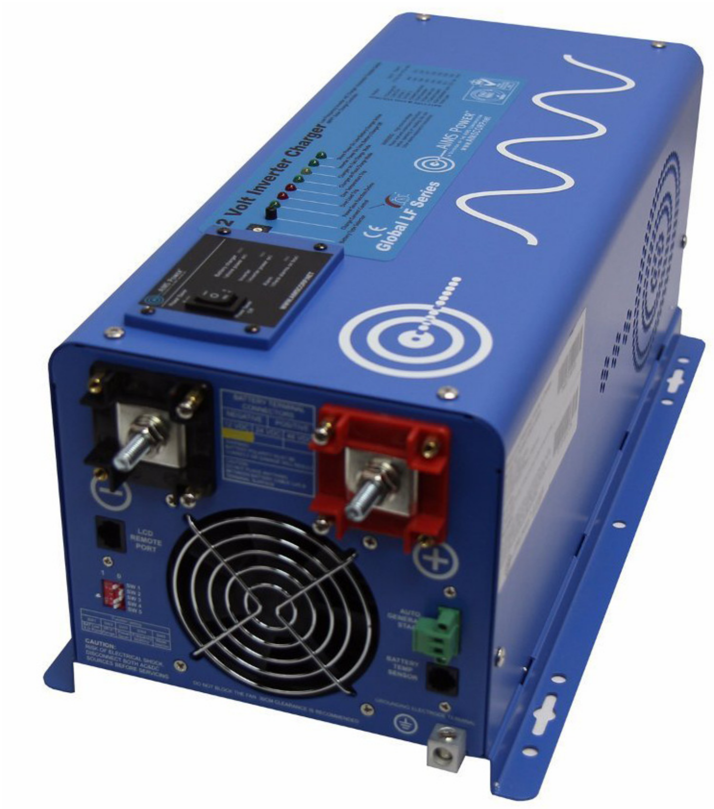


Figure 3. I really like having a separate inverter/charger and battery. If the battery dies, I can replace it without replacing the inverter hardware itself. Plus, solar power will be fun to play with!

For my home, I decided to separate the inverter from the battery. I bought an AIMS inverter/charger unit (Figure 3) that is hard-wired inline with my house wiring. The specific unit I bought is the 3000-watt pure sine inverter. It gives clean voltage and uses external batteries. In my case, it uses a huge 12v marine battery. This will keep my little server rack running for quite some time before the battery is drained, and since it's a robust lead acid battery, it seems to last better than the tiny rackmount UPS batteries. Time will tell.

I decided to go with 3000 watts because I wanted to make sure the full 15 amps of service would be supported, even though I don't use nearly that much on my rack. I also like the AIMS power inverter because it comes with connections that support charging with a solar array along with grid power. I have no intention of going "off grid" with my unit, but I love the idea of offsetting a little of my electrical bill with solar panels. (That will likely be a future article!)

Probably More Than You Need

Quite honestly, my power problems are likely more pronounced than most folks. In our last house, I never had any problem with our line power at all, and the idea of line conditioners and voltage regulators seemed absurd. With our new house, however, if I want to be able to work from home and use computers all day, I have to make sure my power is rock-solid, and my two Internet connections are reliable. But who knows, maybe I actually will decide to go off the grid someday. If that ever happens, I'll be sure to write about solar panels and battery banks. Until that time comes, enjoy your conditioned electricity!■

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.

SUSAN SONS

Fast Network Routing, Meet Userspace

Interview with Katerina Barone-Adesi about developing with the Snabb Switch network toolkit, working outside the Linux kernel for cleaner code and faster throughput.

Recently, I had the pleasure of sitting down with Katerina Barone-Adesi, a developer from the Snabb Switch community to talk about Snabb Switch, writing network infrastructure code and more. In case you aren't familiar with Snabb Switch, it is described in its README as "a simple and fast packet networking toolkit". The thing that caught my attention was Snabb Switch's unusual architecture: it's small and fast, but written largely in LuaJIt, which I didn't expect. It circumvents the kernel almost entirely, preferring to interface directly with network hardware and network code that is slower than that in Snabb Switch.

SS: Can you give me some examples of what Snabb Switch can be used for?

KB: Snabb should be interesting to anyone who wants to do high-speed networking with the flexibility of writing software and seriously needs to consider several gigabits a second or more of traffic. Secondarily, it is interesting to hobbyists and researchers. Alexander Gall, at SWITCH, wrote an L2VPN on Snabb, because the features he wanted were not available elsewhere. It is a really powerful tool for individuals and small teams who need to do unusual or new things with networks.

SS: So, how did Snabb Switch get started?

KB: Snabb was started by Luke Gorrie, who has a fairly deep background in networking, along with a professional history using Smalltalk, Forth, etc. He was influenced by some of Alan Kay's ideas about building small, comprehensible systems that individuals can understand—and also by the desire to do flexible, high-performance software-defined networking on commodity x86 hardware.

It started with a code budget—the binary should fit on a floppy, compile in less than a second (and less than a minute including dependencies), and be less than 10,000 lines of code. It has grown a bit beyond that. Essentially, it lets people do more or less line speed networking using 10 gigabit cards, on a single core, without the overhead of using the kernel. The kernel is involved peripherally (for instance, it uses hugepage support), but Snabb implements its own drivers, and the cards it is handling are not managed by the kernel.

SS: A lot of what's out there in this space today is *big*...when you're introducing people to these tools (either building them or using them) do they usually get where you're going right away? If not, what are the parts that are hardest for them to wrap their heads around?

KB: I would say the biggest initial hurdle, assuming some knowledge of networking, is that Snabb is a fairly young project. Some parts are well documented, others are not, and there are not many examples of how its libraries work, even scattered around GitHub. Snabb itself is pretty small and well factored, which helps a lot. More generally, networking is a vast space. When it comes to introducing people to using tools built on Snabb, there is often some incredulity about using a garbage-collected high-level language to do high-performance networking, but LuaJIt has a solid track record in this space by now. Aside from that, most people are not used to thinking of the timescales involved. Depending on the expected average packet size, if you are driving a 10 gigabit card, you have tens to hundreds of nanoseconds per packet. With these constraints, you need to think about when you will hit RAM, and things like system calls are prohibitively expensive.

SS: Why did you end up choosing LuaJIt instead of, for example, straight C?

KB: Snabb is written in LuaJIt, so the choice of language (and license—it's APLv2) came with the choice to use Snabb. That said, using LuaJIt is excellent for projects like this. It is a just-in-time trace compiler, so the actual code being run is shaped by the packets being seen, and there are no branch mispredictions within a trace. It also makes it extremely easy to use C when appropriate. It has an extremely nice FFI, good support for packed structs, and calling C code seems as easy as it would be if the whole project were in C. The code itself tends to be more concise and quicker to write—it is quite comparable to Python or Ruby in that regard. It is quite pleasant not to have to read past boilerplate details.

SS: What tools are you using to track down bottlenecks and get the optimization you need for the tight time constraints?

KB: The tools that we are using to track down bottlenecks include PMU counters and a combination of logical thinking about what is likely to be a bottleneck, followed by measuring changes to see whether they actually make an improvement. Having a small system makes the latter a surprisingly tractable approach. We also look at LuaJIt's trace dumps, which include IR (Intermediate Representation) and x86 assembly.

SS: Have you had to break parts out to C yet?

KB: I have not personally had to break parts out to C yet. People occasionally do—for instance, if they want to hide an unpredictable branch from the trace compiler. Another alternative that a colleague of mine used recently is DynASM, which allows you to generate assembly dynamically. He used it for some AVX2 instructions to access memory 256 bits at a time. One of the very surprising things about LuaJIt is that it often is as fast as C, and swapping code out for C blindly can genuinely make performance worse, as you lose the benefits of traces.

SS: How much do you have to know about the specific hardware this will be running on to keep the speed up? Is architecture plus network card enough, or are you falling into "oh, sorry, you have a brand of RAM we never tested..." level of optimization specificity?

KB: So far, micro-architecture (Haswell vs. Sky Lake) and raw CPU speed do make a noticeable difference in performance for some applications. For others, like the packetblaster app that ships with Snabb, any modern CPU Snabb can run on is so far beyond what is needed that the bottleneck is the network cards.

SS: So, to recap, Snabb Switch is an open-source *userspace* application written mostly in LuaJIt that's pushing network traffic on run-of-the-mill COTS x86_64 easily at 1Gbps, with a 100Gbps target in 2016...by bypassing the Linux kernel for faster performance...and you say you're getting comparable performance to C code from LuaJIt, without excessive scaffolding for performance testing.

KB: Yep.

SS: So, how big is this thing again?

KB: The source is less than 5MB; the resulting binary is something like 2MB before compression.

SS: Impressive. However, this isn't just some endpoint software, it's infrastructure—what does your team do to ensure code quality and security?

KB: For code quality, my team uses continuous integration, tests and code review. All code is reviewed by at least one person other than the author before being committed to the main development branch. We have a variety of end-to-end and unit tests. Pflua also has property-based tests, which compare its results with and without optimization to libpcap's results on randomly generated filters. We have plans to do more property-based and fuzz testing. Additionally, the small size of the system makes it easier to pay close attention to code quality. For security, all of the above, along with thinking hard and paying close attention to RFCs and their notes on security. Our fragmentation reassembly code rejects overlapping fragments as more recent RFCs recommend, for example.

SS: When you bring on new developers, how do you inculcate them into this kind of rigor given how little it's present in other programming shops, and how essential it is for developing infrastructure software?

KB: I introduced it to my team when I joined in 2014. We started using continuous integration software and property-based testing at that point, both of which significantly increased the reliability of pflua. Code reviews also help. We have had only one person join our team since; as we grow more, perhaps I will have a better answer for that. The code also had a lot of conceptual rigor to begin with. My colleague who shaped pflua most deeply, Andy Wingo, spends most of his time working on compilers, which also demands a fairly high level of rigor and coherent design.

SS: If you were to guess, about what percentage of your project's effort is spent on new code/features vs. testing/refining/optimizing, writing test code and scaffolding for fuzz testing vs. documenting and planning tasks?

KB: It depends on the phase of each project, I'd say. They tend to start off with a flurry of features and planning, and some testing is done in parallel. Later in each project, we optimize and test more. The efforts compliment each other well, as more extensive tests catch subtle mistakes attempts at optimization can introduce. Some of the features are specifically linked to performance goals, which makes them fall thoroughly into being both new code/features and testing/refining/optimizing at the same time. My colleague Andy Wingo's recent work with dynasm is an example. Earlier in the lwaftr project, almost all of the time was going to new code and features, with perhaps 20% going into testing them. Before the alpha release, the whole team focused almost entirely on performance for a few weeks. At the moment, I think it might be around 50/50.

SS: If you were to break it down to two to five principles the Snabb Switch toolkit is built on, what would they be?

KB: I would actually break it down to just one: systems should be small, and individuals should be able to hold the whole system in their head. Most of the other principles are consequences of this, and the high-speed software-defined networking/network function virtualization niche it is designed for.

SS: Why is this so important to Snabb Switch in particular?

KB: It is an important principle for software wherever it can reasonably be applied, and it is one thing that strongly drew me toward Snabb. I can only speculate that for Snabb in particular, it came from Luke's background with Smalltalk—where he took the quotes describing this principle from—and other minimal systems like Forth.

SS: What do you like best about working on Snabb Switch?

Between the design principles and being able not only to replace old network functions but also implement entirely new ones on commodity x86 hardware and network cards that cost a few hundred bucks, I think it has a really exciting future.

KB: Snabb is a really fun and interesting project to be involved with these days. Between the design principles and being able not only to replace old network functions but also implement entirely new ones on commodity x86 hardware and network cards that cost a few hundred bucks, I think it has a really exciting future.

SS: So, what got you interested in tweaking this type of networking code?

KB: This is my day job. I joined Igalia in 2014, when the networking team was working on pflua. Pflua is a library on top of Snabb that uses a subset of libpcap's filtering language. It is smaller, faster, and the subset that it supports is believed to be entirely compatible except for libpcap optimizer bugs.

I got into networking as a hobby back around 2000 though. I ran a small home network on Linux, various BSDs, read a lot of books, played around, etc. I have liked systems programming and dynamic languages for about as long, so a job that involves all of these elements is pretty fun.

SS: Did you make a specific effort to choose companies where you'd be doing open-source work, or was it luck?

KB: Igalia does only free software, and this was a major factor in my choosing to work for them. I did also interview at other companies that do a mixture of free and non-free software.

SS: Do you have any advice for coders interested in Snabb Switch in particular, or in moving from places higher up the stack into more infrastructure-y areas of programming work like what you're doing?

KB: For coders interested in Snabb

Switch in particular, it depends on their background; ones with more relevant experience can probably jump in and implement something Luke brain-dumps about on the mailing list, or something else entirely. For those newer to these kinds of programming, I would recommend starting by writing a Snabb app. These can be as simple as a packet blaster or an app that echoes packets between interfaces, or more complex than the l2vpn or lwAFTR, and can be built up incrementally. Some of the tests can be run, and app development can be done, on any modern x86 machine running Linux. I prototyped the lwAFTR on my development laptop, which has no Ethernet cards. Snabb has a mechanism for plugging apps into each other that makes running it on real hardware a simple matter of changing a couple lines of configuration and recompiling.

SS: Thanks again, Katerina, for sharing your time and expertise.■

Susan Sons serves as a Senior Systems Analyst at Indiana University's Center for Applied Cybersecurity Research (http://cacr.iu.edu), where she divides her time between helping NSF-funded science and infrastructure projects improve their security, helping secure a DHS-funded static analysis project, and various attempts to save the world from poor information security practices in general. Susan also volunteers as Director of the Internet Civil Engineering Institute (http://icei.org), a nonprofit dedicated to supporting and securing the common software infrastructure on which we all depend. In her free time, she raises an amazing mini-hacker, writes, codes, researches, practices martial arts, lifts heavy things and volunteers as a search-and-rescue and disaster relief worker.

Resources

Snabb Switch: https://github.com/SnabbCo/snabbswitch

"Snabb Switch Deep Dive on Software Gone Wild" by Ivan Pepeinjak: http://blog.ipspace.net/2014/09/snabb-switch-deep-dive-on-software-gone.html

"L2VPN over IPv6 with Snabb Switch on Software Gone Wild" by Ivan Pepeinjak: http://blog.ipspace.net/2014/12/l2vpn-over-ipv6-with-snabb-switch-on.html

"High-performance packet filtering with Pflua": https://archive.fosdem.org/2015/schedule/event/packet_filtering_pflua

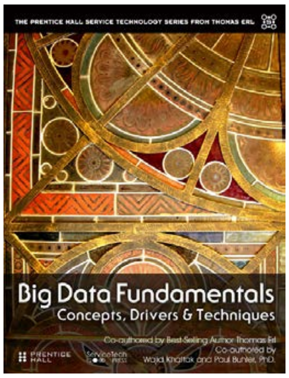
"Deutsche Telekom TeraStream: Designed for Simplicity" by Ivan Pepeinjak: http://blog.ipspace.net/2013/11/deutsche-telekom-terastream-designed.html

SoftMaker Office

Founded in 1987, the Germany-based software vendor SoftMaker has spent the modern PC era developing and improving its own office program, SoftMaker Office. SoftMaker Office 2016 for Linux is SoftMaker's latest release of the office suite, which consists of the word processor TextMaker, the spreadsheet program PlanMaker and the presentation program SoftMaker Presentations. SoftMaker's value proposition vs. other office suites includes a compact footprint, stability, availability in Standard and Professional versions, add-ons for the Thunderbird e-mail client, as well as compatibility with (and a five-fold speed advantage over) Microsoft Office. The new SoftMaker Office 2016 boasts more than 400 new features, such as optimized software architecture accelerated by the use of OpenGL for screen display and multiple CPU cores for graphic rendering, improved compatibility features with MS-Office, built-in version management for documents, support for 9,000 label formats, improved pivot-table functionality, smart guides for alignment of objects in presentations and more. The Professional edition adds four high-quality dictionaries by Berlitz for easy translation between English, Spanish, Italian, German and French. All modern Linux distributions are supported.
http://softmaker.com

Ziften ZFlow

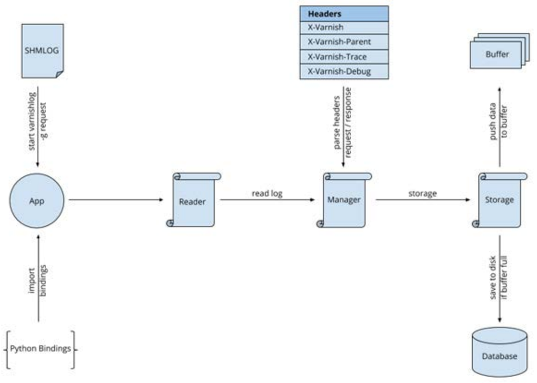
As enterprises have migrated to the cloud for economic and operational purposes, traditional network visibility has been lost due to a lack of access at the infrastructure layer. To help organizations avoid operating in the dark in the penguin zone, security specialist Ziften has released a new version of its flagship Ziften ZFlow for the Linux operating system. Ziften ZFlow delivers greater network visibility by providing full visibility, contextual intelligence, user behavioral analysis and integration into previously deployed security tools. This new integration with Linux provides previously non-existent visibility into the public cloud infrastructure and enables Ziften's new Cloud Visibility Initiative. The initiative helps secure cloud operations with the visibility that security professionals need to identify and respond to potential threats and attacks quickly. Ziften adds that ZFlow is lightweight, meets IPFIX standards and enables better east-west visibility to identify lateral movement of an attack within the data center.
http://ziften.com


THE PRENTICE HALL SERVICE TECHNOLOGY SERIES FROM THOMAS ERL
Big Data Fundamentals
Concepts, Drivers & Techniques
Co-authored by Best-Selling Author Thomas Erl
Co-authored by Wajid Khattak and Paul Buhler, PhD.
PRENTICE HALL
ServiceTech PRESS

Thomas Erl, Wajid Khattak and Paul Buhler's *Big Data Fundamentals* (Prentice-Hall)

Because Big Data will only get bigger, the case for reading *Big Data Fundamentals: Concepts, Drivers & Techniques* will only get stronger. Co-authored by best-selling IT author Thomas Erl and collaborators Wajid Khattak and Paul Buhler, *Big Data Fundamentals* is a pragmatic, no-nonsense introduction to Big Data for business and technology professionals. The authors clearly explain key Big Data concepts, theory and terminology, as well as fundamental technologies and techniques. All coverage is supported with case-study examples and numerous simple diagrams. The authors commence by explaining how Big Data can move an organization forward by solving a wide range of previously intractable business problems. Next, they demystify key analysis techniques and technologies and show how a Big Data solution environment can be built and integrated to offer competitive advantages. Other key topics include differences between Big Data and previous forms of data analysis and science, business motivations and drivers behind Big Data adoption and recognizing the five "V" characteristics of datasets in Big Data environments: volume, velocity, variety, veracity and value.
http://informit.com

Varnish Software's Zipnish



Gaining insights into how quickly services are running or whether they are adding latency is a difficult task in distributed architectures such as microservices. To simplify this task, Varnish Software launched Zipnish, a new open-source, architecture-agnostic tool that tracks performance and helps resolve latency issues in microservices architectures. Zipnish uses the Varnish logging API from Varnish Cache 4.0 to monitor transactions and uses Python and the event library Twisted to transport the data. MySQL is used as the database for storage. The presentation back end is done in Python, whereas a slightly modified version of Zipkin is used as front end.
http://varnish-software.com

Red Hat CloudForms

To meet customer demands in today's hybrid, heterogeneous world, Red Hat established an alliance with Microsoft to release Red Hat CloudForms 4, the latest version of the company's open-platform hybrid cloud management solution that now features Microsoft Azure support. Red Hat declares that Red Hat CloudForms 4 goes beyond self-service, offering a consistent experience and comprehensive lifecycle management across platforms, covering virtualization, private cloud, public cloud and containers. In addition to enabling Azure customers to manage those workloads and resources within CloudForms, release 4 adds management for container architectures and improved self-service dashboards and charts to better analyze the relationships between different cloud platforms and container hosts. Other platforms besides Azure into which Red Hat CloudForms 4 provides operational insight include Amazon Web Services, Hyper-V, OpenShift by Red Hat, OpenStack, Red Hat Enterprise Virtualization and VMware.
http://redhat.com

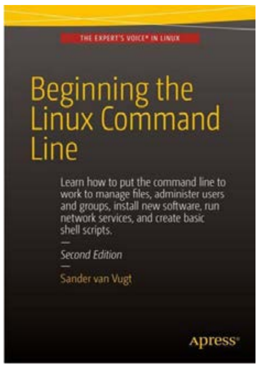
Imagination Technologies' Creator Ci40

"Where do I start?" is a common question for those of us exploring the Internet of Things (IoT) and embedded computing markets. Imagination Technologies answers that question in the form of a product, namely the firm's new Creator Ci40 IoT kit, which helps one make sense of the oh-so-many boards and accessories, hardware and software interfaces, connectivity standards and APIs in IoT. The Creator Ci40 includes not only the hardware building blocks needed to prototype a wireless IoT system from scratch quickly, but also the open-source software frameworks, the network stacks and the cloud connectivity capabilities required to connect and authenticate devices to the cloud securely. The included Ci40 microcomputer has been specifically designed for smart home, IoT and other connected devices (drones, robots and so on) and features the hardware requirements (802.11ac 2x2 Wi-Fi, an Ethernet port, SD and USB storage) for use as a high-speed wireless router. The board runs Linux (Debian, OpenWrt) and is part of Google's golden reference program for its Brillo OS for IoT. Rounding out the kit are add-ons from Mikroelectronica: two battery-powered Clicker board development kits and three Click boards for measuring temperature, detecting motion and controlling a relay.
http://imgtec.com

Opera Max

Cheapo data hogs (like you) with limited data plans can stretch their budgets and double their streaming of video—and now music—using Opera Max, Opera ASA's upgraded data-management and data-saving app for Android. The latest release adds streaming audio optimization to the existing video optimization functionality. Opera notes that nine hours of streaming music or on-line radio require 1GB of throughput, which easily can burn through a data plan when using mobile networks. Complementing Opera Max's existing ability to optimize streaming-video services like YouTube and Netflix comes new functionality that optimizes streaming music apps, such as YouTube Music, Pandora, Slacker Radio, Gaana and Saavn, with others on the way. Opera's optimization technique for audio involves applying technology from Rocket Optimizer for converting MP3 and MP4 audio streams to the more efficient AAC+ codec, which delivers high audio quality over a low bitrate connection to any compatible device.
http://opera.com

Sander van Vugt's *Beginning the Linux Command Line*, 2nd Edition (Apress)



IT author Sander van Vugt says that his book *Beginning the Linux Command Line*, now in it second edition, is for everyone who uses Linux—no exceptions. Van Vugt's book is a guide to Linux in which the user digs deeply into the system, wielding the shell as a shovel. Key overarching goals include understanding how Linux is organized and how to "think Linux". Other learning objectives include finding the right command for the task at hand, working with text editors and intelligent filters, shell programming, configuring access to hardware devices and more. *Beginning the Linux Command Line* is checked against all of the most important Linux distributions and follows a task-oriented approach that is distribution-agnostic. This new second edition of the book covers features of the very latest versions of the Linux operating system, including the new Btrfs filesystem, the systemd boot procedure and firewall management with firewalld.
http://www.apress

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.

TEMPERATURE CONTROL IN A HOMEBREWING TUN

Using a BeagleBone Black

By using a thermometer, a PID controller and a PWM output on the BeagleBone, you can control your brew tun when brewing your beer.

KLAUS KOLLE

I love beer—good beer, not industrial beers. The ones I brew myself, where I can control the ingredients and the process, are very good.

Beer brewing is a simple process—it used to be a duty of housewives on the farms here in my country long ago. The process I and my two brew mates use is a bit more complicated than the process used in ancient country houses though. One thing we can do nowadays is mash the malt in several temperature stages. This results in a more complex beer with a more malty body. And, we now have better efficiency—that is, more sugars for the yeast to produce alcohol from, meaning less malt will provide the same amount of alcohol in the beer if our efficiency is higher than normal.

But, some problems arise when trying to brew using our simple brew tun. The tun is a big pot with a 29-liter capacity. The pot has a built-in heating element with a rated power

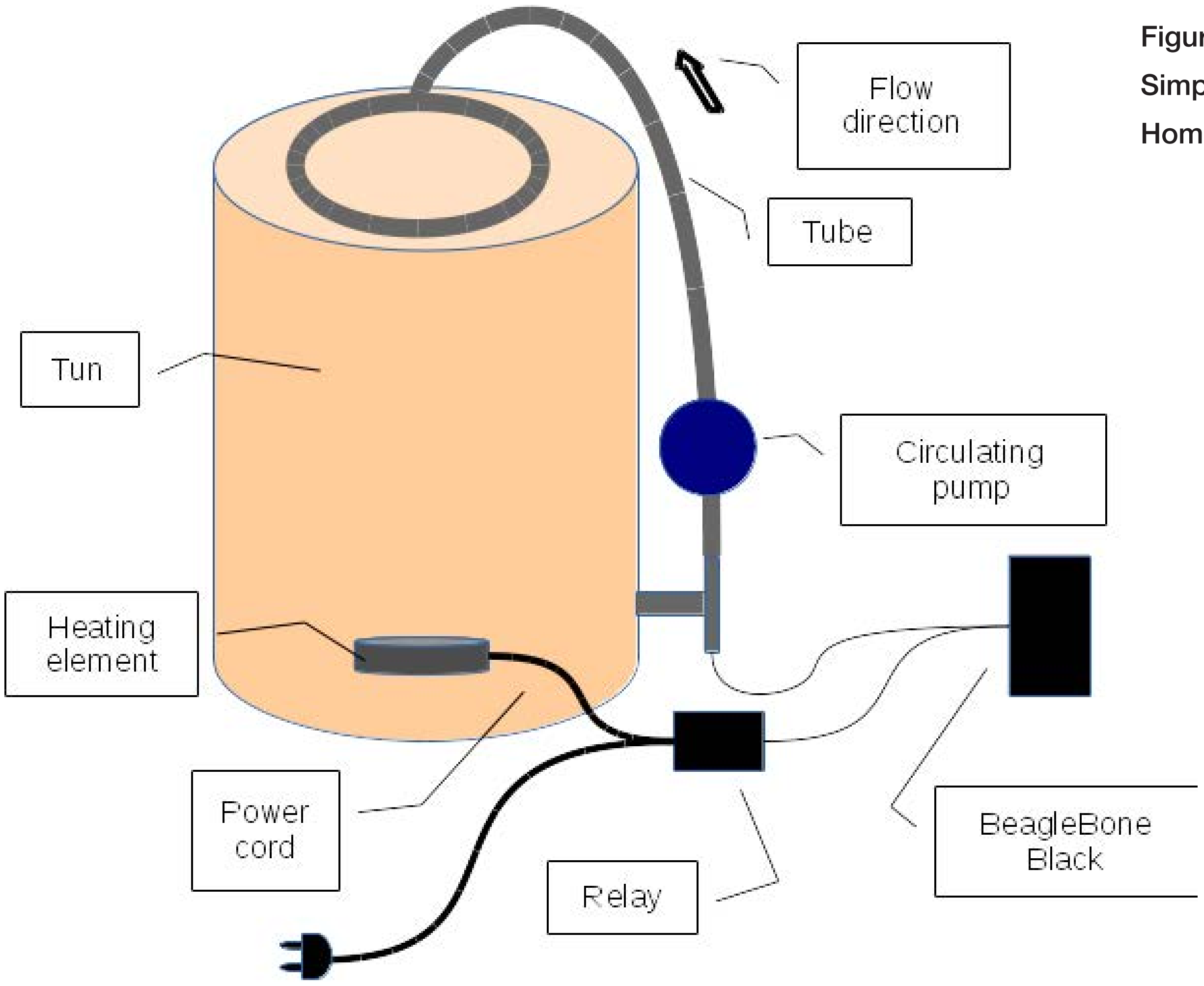


Figure 1. Simplified Homebrew Tun

of 1800 W (Figure 1). Helped by a circulating pump, the mash (fluid) circulates from the bottom of the tun to the top overflowing the malt.

Many of our beers have a starting point between 45°C–55°C. In this temperature range, we are preparing the proteins in the malt for the later process of extracting sugars out of the malt. The optimal temperature for the beta and alpha enzymes that convert starch to sugars is between 63°C–68°C. So after the protein stop in the 50°C range, we have to raise the temperature in the mash. But unfortunately, too much heat will burn some of the particles floating in the mash onto the heater in the bottom of the tun, resulting in a bad taste and difficulty getting the heat from the heating element into the mash. We have determined empirically that from 59°C–60°C we can apply full power. To finish the process, the temperature is raised to 78°C to stop the enzymes from working.

So, we need a way to control the amount of power fed into the heating element. What is more natural than to use a BeagleBone Black to control the rate of energy burned in the heater through a Pulse Width Modulated (PWM) controller already built in to the BeagleBone? The PWM will produce a pulse of varying width, turning a relay on and off in controlled intervals, thereby controlling the amount of power fed into the heater. If the relay is closed for 2 seconds and off for 8 seconds, 2/8 * 1800 W = 450W is fed into the heater. Well, to be honest, this problem could be solved using a simple 555 chip and a potentiometer, but since we are programmers, we tend to pull out the tools we love, even when it's overkill.

A side benefit from setting up a BeagleBone is the ability to log temperatures during the whole brew process. This is handy for evaluating the beer when adjustments to the recipe, which include the processing, are discussed. Previously, we noted temperatures and times on a piece of paper a few times during the brew process—when we remembered to do it. Another benefit is that the whole thing can be controlled from a Web page.

System Overview

I've already mentioned the requirements, and from there, you can extract that we need a controlling functionality that will read the temperature in the mash when it leaves the bottom of the tun. Based on the temperature read, it will set the period time of the output PWM. Also, we need a database enabling us to log the collected data during the brew process. Now, a database also could archive the recipes or at least the part of the recipes that is about temperatures and time. Finally, I also mentioned having a Web page. Data comes from the database, but how do we serve the data to the Web page? Because I also wanted to show the state of the controller on the Web page, I embedded a Websocket server into the application. This led (after a few iterations) to the software architecture diagram shown in Figure 2. Add to the architecture diagram a Web client program—for example, Firefox that loads and executes the JavaScript code embedded in the Web page. However, the database and Web page are beyond the scope of this article; I may

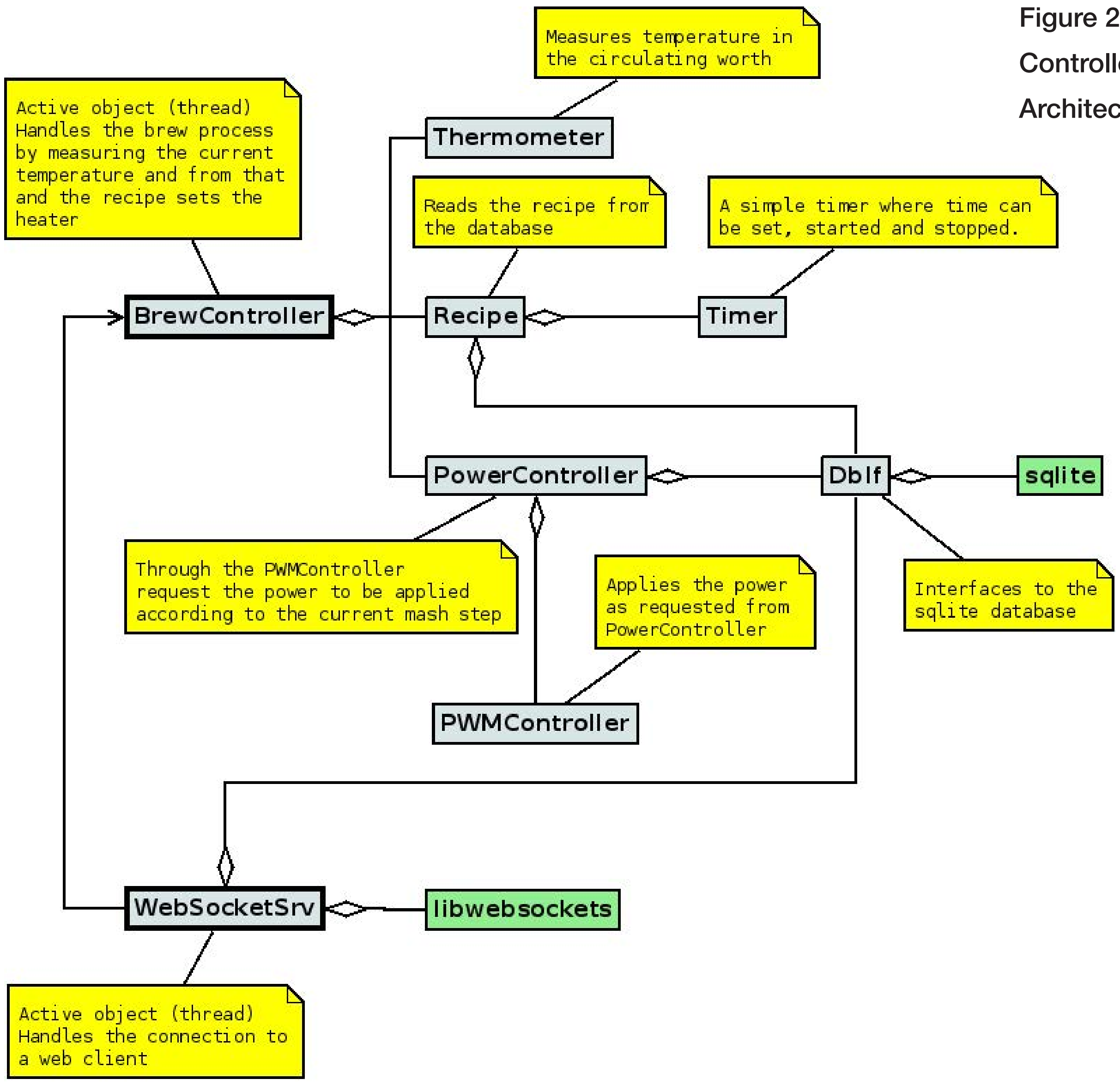

Measures temperature in the circulating worth
Active object (thread) Handles the brew process by measuring the current temperature and from that and the recipe sets the heater
Thermometer
Reads the recipe from the database
A simple timer where time can be set, started and stopped.
BrewController
Recipe
Timer
PowerController
DbIf
sqlite
Through the PWMController request the power to be applied according to the current mash step
Applies the power as requested from PowerController
Interfaces to the sqlite database
PWMController
WebSocketSrv
libwebsockets
Active object (thread) Handles the connection to a web client

Figure 2. Brew Controller Software Architecture

cover them in a future article.

I decided to develop in C++, but I could have used any programming language. In fact, I did start developing a prototype in BoneScript, a dialect of JavaScript that builds on Node.js, which is programmed in JavaScript. It worked quite well, but I was unsure whether I could trust it to be stable in the long run. I noticed during my tests that the BoneScript program had some delays a few arbitrary times and hiccups I couldn't explain. A program, where I know all details and that is compiled, is by far the most efficient, and once tested and debugged, I can trust this one to run for long time. And, now I also have several different examples of the same solution to show my students and discuss with them.

Preparing for Development

In order to prepare for developing the software that fulfills the requirements, a few things needed to be set up and configured:

- Cross compilation.
- Remote debugging.
- Setting up a temperature sensor.
- Setting up the PWM output.
- A log utility.
- Threading.

Cross Compilation: I prefer to develop using Eclipse when it is a larger project that runs over some time. There is no problem when developing small and quick solutions directly on the BeagleBone, but when complexity increases, the help that an integrated development environment provides levels out some of the complexity. Can you install Eclipse on the BeagleBone? Yes, it should be possible, but since the storage on the BeagleBone is Flash-based, I prefer to run Eclipse on my laptop. A lot of writing to files can wear out the Flash memory on the BeagleBone.

The BeagleBone is built using an ARM CPU architecture, and most desktops and laptops are Intel CPU architectures. This means we need to install a cross compiler on the laptop. A cross compiler is a compiler that produces executable code for another CPU architecture than what it executes on. In this case, the compiler executes on an Intel CPU but produces code for an ARM CPU.

I run Fedora Linux and have been doing so ever since the Fedora Project was launched. I couldn't find a suitable cross compiler in the repositories for Fedora, however. So, after some searching, eventually I found that Linaro.org maintains a suitable compiler suite that I could install. If you're following along, you should download a stable release from the repository. This is the one I downloaded: http://releases.linaro.org/14.11/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf.tar.xz. There might be fresher copies by the time you read this. Take a look at http://releases.linaro.org/14.11/components/toolchain/binaries, and be sure to download a gnueabihf version—the compiler that produces code for a hard floating-point unit.

I keep my downloaded tools, like compilers and Eclipses, in the /opt directory, so I unpacked the tarball in a directory I named toolchains. In this directory, you can make a symbolic link to the very long directory name you will get out of the tarball. This makes it easier when configuring a project in Eclipse. I made a symbolic link called gnueabihf, which is far easier to remember than gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf. And if I update the compiler at a later time, I can just remove the link and create a new one pointing to the newer compiler. I don't have to change anything in Eclipse.

In Eclipse, you have to select the Cross GCC choice in the toolchain section of the New Project dialog. In the next dialog, enter "arm-linux-gnueabihf-" in the Cross Compiler Prefix entry. If you look into the directory where you stored the cross compiler, you will find that *almost* all binaries are prefixed with this. So the gcc is really arm-linux-gnueabihf-gcc. In the cross compiler path, enter the path to where you installed the cross compiler—for example, /opt/toolchains/gnueabihf/bin. Now Eclipse knows how to produce executables that will run on the BBB.

Remote Debugging: Regardless of how experienced we are, bugs tend to creep into our code making programs not work as expected. So the ability to debug is mandatory—well, for me at least, but I also do have only 30+ years of programming experience. Luckily, it is possible to set up Eclipse so you can execute the freshly compiled program over on the BeagleBone under debugger control.

Included with the gnueabihf package is a gdbserver that can cooperate with Eclipse or, more precisely, with gdb the debugger. The gdbserver that comes with the BeagleBone by default does not cooperate well with Eclipse. So, you have to copy the gdbserver to the BeagleBone:

```
scp /opt/gnueabihf/bin/gdbserver <username>@<your BBB IP>:~/
```

If you just use the USB-created network interface, the address is 192.168.7.2. I have set up a Wi-Fi dongle on my BBB, so my address is different.

The next task is to configure Eclipse to perform remote debugging. Because this is a rather long procedure, I have collected it on a page at http://klaus.ede.hih.au.dk/index.php/BBB_Remote_Debugging. I made it for my students to follow, so readers of *Linux Journal* also should be able to follow these instructions.

Measuring and Controlling

Now we're ready to start programming. My typical practise, when attacking something I don't know anything or at least not much about, is to create small projects where I isolate the particular problem I am dealing with. One of the first things was measuring the temperature. I had a DS18B20 1-Wire temperature sensor. The sensor is embedded in a metal cap, so it is fine in food production environments. You easily can find them on eBay or similar sites.

Getting the Temperature: I've never worked with 1-Wire devices (I just knew they existed), so I had to do some reading to understand the device, but it is rather simple. Each device on the 1-Wire bus has a unique address. Now for the purpose of this article and the project, you don't need to have a full understanding of the 1-Wire devices, because the Debian Linux that comes with the BeagleBone has drivers for 1-Wire devices. So, there's no need to develop a device driver ourselves.

The BeagleBone uses the Flattened Device Tree (FDT), like many modern Linux distros for embedded systems running on an ARM architecture. The ARM CPU that is on the BeagleBone has an electronic pin multiplexer (a kind of switchboard) built in, enabling you to connect pins from the outside world to a specific internal device—for example, GPIO or PWM. The FDT will help you set this up, and it tells the kernel which type of interface it is enabling, so the kernel can use a suitable driver to service the pin. For more information on the FDT, see http://elinux.org/Device_Tree.

Basically, the FDT works by having a text file with the specification of the connection you want to enable. This specification file is compiled into binary format. You may be interested in a generator for DTS files, because it takes a lot of research and thought to put together a DTS. I stumbled upon this site: http://kilobaser.com/blog/2014-07-28-beaglebone-black-devicetreeoverlay-generator#dtogenerator, which I find helpful in cooperation with the technical specification of the BeagleBone (which is at https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true).

By looking in the technical specification in table 13, I could see that pin 12 on the P9 header could be connected to a GPIO pin. After some testing, I managed to put together a device tree file that enabled me to connect the DS18B20 to P9 pin 12:

```
/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";
    part-number = "DS1820";
    version = "00A0";

    exclusive-use = "P9.12";

    fragment@0 {
        target = <&am33xx_pinmux>;
        __overlay__ {
            ds1820_pins: pinmux_ds1820_pins {
                pinctrl-single,pins =  <0x78 0x37>;
            };
        };
    };

    fragment@1 {
        target = <&ocp>;
        __overlay__ {
            onewire@0 {
                status          = "okay";
                compatible      = "w1-gpio";
                pinctrl-names   = "default";
                pinctrl-0       = <&ds1820_pins>;
                gpios           = <&gpio2 28 0>;
            };
        };
    };
};
```

Save the specification in a file called DS18B20-00A0.dts, and compile it into the binary format using this command:

```
root@beaglebone:~# dtc -O dtb -o /lib/firmware/DS1820-00A0.dtbo
➥-b 0 -@ DS1820-00A0.dts
```

`dtc` is a Device Tree Compiler. In the above command, it's instructed to output in the dtb format (`-O`) into a file in the /lib/firmware directory (`-o`). The `-b` sets the boot CPU (here 0) and the `-@` means to use symbols, and finally, the input file is specified.

To enable this part of the device tree, issue this command:

```
root@beaglebone:~# echo DS1820 >
 ➥/sys/devices/bone_capemgr.*/slots
```

If you don't get any errors, run `ls -la` in the /sys/bus/w1/devices directory:

```
root@beaglebone:~# ls -la /sys/bus/w1/devices
total 0
drwxr-xr-x 2 root root 0 Jan  1  2000 .
drwxr-xr-x 4 root root 0 Jan  1  2000 ..
lrwxrwxrwx 1 root root 0 Aug 30 17:32 28-000005a7ce64 ->
 ➥../../../devices/w1_bus_master1/28-000005a7ce64
lrwxrwxrwx 1 root root 0 Aug 30 17:32 w1_bus_master1 ->
 ➥../../../devices/w1_bus_master1
root@beaglebone:/lib/firmware#
```

The 28-000005a7ce64 file (a symbolic link) is the thermometer. Each 1-Wire device has a unique ID, so it will be identified by a pattern like 28-00000nnnnnnn, where nnnnnnn is the unique address.

A quick test to see if the DS18B20 is working correctly can be done by changing the device address in this Python script to suit your configuration:

```
import time

w1="/sys/bus/w1/devices/28-000005a7ce64/w1_slave"

while True:
    raw = open(w1, "r").read()
    print "Temperature is "+str(float(raw.split("t=")
➥[-1])/1000)+" degrees"
    time.sleep(1)
```

If you want to have the DS18B20 enabled at every boot, prepare a file in /etc/init.d, and call it enable-DS18B20 or something similar. Put this into the file:

```
#! /bin/sh

### BEGIN INIT INFO
# Provides: enable-DS18B20
# Required-Start: $all
# Required-Stop: $all
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Enables the DS18B20 1-wire
 ➥on P9 pin 12
# Description: Connecting the 1-wire driver to
 ➥the P9 pin 12
### END INIT INFO
case "$1" in
        start)
            echo "Enabling 1-Wire DS18B20 on P9 Pin 12"
            echo DS1820 > /sys/devices/bone_capemgr.9/slots
        ;;
        stop)
            #no-op
        ;;
        *)
```

```
                #no-op
            ;;
esac

exit 0
```

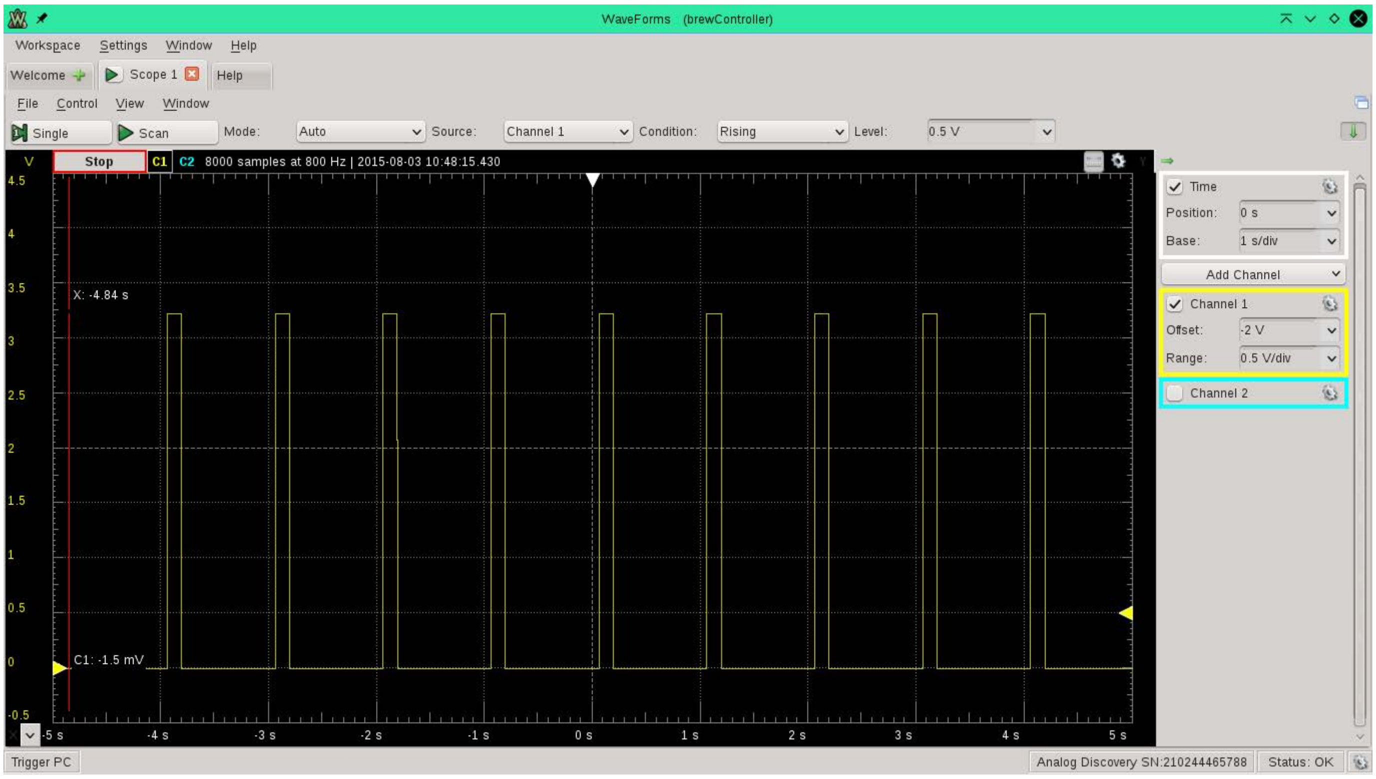
After creating the file, execute:

```
root@beaglebone:/etc/init.d# chmod 755 enable-DS18B20
root@beaglebone:/etc/init.d# update-rc.d enable-DS18B20
 ➥defaults
```

Try to reboot your system and check that the device is enabled as expected.

Setting Up the PWM Controller: As I mentioned earlier, a way of controlling the amount of power supplied to the heater is to use a PWM signal to control a relay. If you turn it on for one-third of the time and off for two-thirds, the applied heat is only one-third of the usual amount of heat when running switched on all the time. This can prevent having particles in the brew burn onto the heater. See Figure 3 for an example PWM signal output. Here I have set the period time to be approximately one second and the "on" time to be approximately 20% of the period time.

As for the 1-Wire, you also need a configuration on the device tree in order to gain access to the PWM



Mode: Auto Source: Channel 1 Condition: Rising Level: 0.5 V

8000 samples at 800 Hz | 2015-08-03 10:48:15.430

Time
Position: 0 s
Base: 1 s/div

Channel 1
Offset: -2 V
Range: 0.5 V/div

Analog Discovery SN:210244465788 Status: OK

Figure 3. PWM Signal Output

controllers on the board and to connect it to a suitable pin. If you consult the BeagleBone's technical specification, you will find that P9,14 connects to a PWM controller. So, jump to the on-line device tree generator and select P9_14 in the "Select Pin" box. Then select "Fast Slew" in the "Slew" box, "Pullup" in "Pullup/Down", and finally, in MuxMode "Mode6: ehrpwm1A", and you will get a device tree file like this:

```
*
 * Copyright (C) 2013 CircuitCo
 * Copyright (C) 2013 Texas Instruments
 *
 * This program is free software; you can redistribute
 * it and/or modify it under the terms of the GNU
 * General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This is a template-generated file from BoneScript
 */
/dts-v1/;
/plugin/;

/ {
   compatible = "ti,beaglebone", "ti,beaglebone-black";

  /* identification */
  part-number = "BS_PWM_P9_14_0x16";

  /* state the resources this cape uses */
  exclusive-use =
            /* the pin header uses */
            "P9.14",
            /* the hardware IP uses */
            "ehrpwm1A";

    fragment@0 {
      target = <&am33xx_pinmux>;
      __overlay__ {
        bs_pwm_P9_14_0x16: pinmux_bs_pwm_P9_14_0x16 {
                pinctrl-single,pins = <0x048 0x16>;
        };
      };
    };

    fragment@1 {
      target = <&ocp>;
      __overlay__ {
        bs_pwm_test_P9_14 {
                compatible      = "pwm_test";
                pwms            = <&ehrpwm1 0 1000000000 1>;
                pwm-names       = "PWM_P9_14";

                pinctrl-names   = "default";
                pinctrl-0       = <&bs_pwm_P9_14_0x16>;

                enabled         = <1>;
                duty            = <0>;
                status          = "okay";
        };
    };
    };
  };
```

If you want the polarity initially set to 0, change the 1 to 0 in the `pwms` line so it looks like this:

```
pwms = <&ehrpwm1 0 500000 0>;
```

Again, you need to compile it from text format to the binary format readable by the kernel:

```
root@beaglebone:~# dtc -O dtb -o
 ➥/lib/firmware/bspwm_P9_14_16-00A0.dtbo
 ➥-b 0 -@ bspwm_P9_14_16-00A0.dts
```

To check that you can create a PWM device file, execute these commands:

```
root@beaglebone~# echo "am33xx_pwm" >
 ➥/sys/devices/bone_capemgr.9/slots
root@beaglebone~# echo bspwm_P9_14_16 >
 ➥/sys/devices/bone_capemgr.9/slots
```

If no errors are shown, take a look in the device directory:

```
root@beaglebone:~# ls -al
 ➥/sys/devices/ocp.3/bs_pwm_test_P9_14.16/
total 0
drwxr-xr-x  3 root root    0 Jan  1  2000 .
drwxr-xr-x 42 root root    0 Jan  1  2000 ..
lrwxrwxrwx  1 root root    0 Sep 19 11:40 driver ->
 ➥../../../bus/platform/drivers/pwm_test
-rw-rw-rw-  1 root root 4096 Mar  1  2015 duty
-r--r--r--  1 root root 4096 Sep 19 11:40 modalias
-rw-rw-rw-  1 root root 4096 Mar  1  2015 period
-rw-rw-rw-  1 root root 4096 Mar  1  2015 polarity
drwxr-xr-x  2 root root    0 Sep 19 11:40 power
-rw-------  1 root root 4096 Sep 19 11:40 run
lrwxrwxrwx  1 root root    0 Jan  1  2000 subsystem ->
 ➥../../../bus/platform
-rw-r--r--  1 root root 4096 Jan  1  2000 uevent
```

The files of interest are "polarity", "period" and "duty". The polarity controls whether the controller outputs 0 or 1 when turned off. The period, as the name indicates, controls the period time—that is, for what length of time is a complete cycle in the controller. The period time is set in nanoseconds. Finally, the duty controls the amount of time that the signal from the PWM controller is active. The duty is set in percentages—for example, for 50%, you would write 0.5 to the duty file.

And as for the temperature sensor, an automatic start at boot would be handy, so create this file in /etc/init.d/enable-pwm with this content:

```
#! /bin/sh

### BEGIN INIT INFO
# Provides: enable-pwm
# Required-Start: $all
# Required-Stop: $all
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Enables the PWM chips and
# connects it through the pinmux
# Description: Connecting the pwm output through
# the pinmux and enables the PWM chip on board
### END INIT INFO
case "$1" in
  start)
      echo "Enabling PWM on P9 Pin 14"
      grep -q am33xx_pwm /sys/devices/bone_capemgr.9/slots ||
       ➥echo "am33xx_pwm" \
          >  /sys/devices/bone_capemgr.9/slots
          echo bspwm_P9_14_16 > /sys/devices/bone_capemgr.9/slots
  ;;
  stop)
          #no-op
  ;;
  *)
          #no-op
  ;;
esac

exit 0
```

Reboot your BeagleBone and check that you get the PWM device files created as shown above.

Now you're ready to develop a program that will measure the temperature, and based on a recipe, calculate the necessary power to apply to the heater.

A Simple Log Utility: When developing a program, it's often necessary to log different values from the program, especially when you have several threads running. Nothing happens at the time you would expect it to—this is the nature of threading. Therefore, I often print out to the console or log to a file, which lets me follow the progress of the program.

In this project, I started printing to the console but switched to logging to a logfile. I searched the Internet and ran across this site: http://www.infernodevelopment.com/c-log-file-class-forget-debuggers.

The header looks like this:

```
#include <fstream>

using namespace std;

class Log
{
  public:
    Log (const char* filename);
    ~Log ( );
    void Write (const char* logline, ...);
    private:
    ofstream m_stream;
};
```

And the implementation is simple:

```
#include "Log.h"
#include <stdarg.h>

Log::Log (const char* filename)
{
  m_stream.open (filename);
}
```

```
Log::~Log ( )
{
  m_stream.close ();
}

void Log::Write (const char* logline, ...)
{
  va_list argList;
  char cbuffer[1024];
  va_start(argList, logline);
  vsnprintf (cbuffer, 1024, logline, argList);
  va_end(argList);
  m_stream << cbuffer << endl;
}
```

This version allows me to log using the sprintf conversion specifiers—for example, `%f` for floats. I use it like this:

```
log->Write("PowerController: INFUSION POWER: \
  PWM set to      %f\n", maxPWM);
```

Notice here, that I use `%f` and pass a float value, which will be logged just like (s)printf would convert the float to a string of digits. This is due to the varadic declaration of the `Write` function.

When you want to see the log while the program runs, open another SSH connection to the BeagleBone and run:

```
root@beaglebone:~# tail -f <logfilename>
```

where you, of course, need to change <logfilename> to match your current log file.

Threading: I know that C++ ISO standard 2011 (or just C++11) allows you to create threads in C++ directly, but prior to that, I needed a way to make classes (or a function in a class) into a thread.

I found an example class on StackOverflow that I have been using ever since. So why learn something new, when you know how to do it? We are all a bit lazy now and then, aren't we? Here it is:

```
#include <pthread.h>
#include <cstdlib>

class Threadable
{
public:
  Threadable()
  {
    _thread = (pthread_t)NULL;
  }
  virtual ~Threadable()
  {/* empty */
  }

  /** Returns true if the thread was successfully
   started, false if there was an error starting
   the thread */
  bool StartInternalThread()
  {
    return (pthread_create(&_thread, NULL,
     ➥InternalThreadEntryFunc, this) == 0);
  }

  /** Will not return until the internal thread has exited. */
  void WaitForInternalThreadToExit()
  {
    (void) pthread_join(_thread, NULL);
  }
  pthread_t GetThreadID () { return _thread; }

protected:
  /** Implement this method in your subclass with
  the code you want your thread to run. */
  virtual void InternalThreadEntry() = 0;

private:
  static void * InternalThreadEntryFunc(void * This)
  {
    ((Threadable *) This)->InternalThreadEntry();
    return NULL;
  }

  pthread_t _thread;
};
```

This is a virtual base class that you can't create instances of directly, but you will inherit from it in a new sub-class.

So, for instance, my BrewController class begins like this:

```
class BrewController: public Threadable
{
  public:
    BrewController ( );
    virtual ~BrewController ( );
    void InternalThreadEntry ( );
...
```

The `InternalThreadEntry ( )` is the thread function for this class. So you will just have to fill in the code that composes the thread. The rest of the methods in the class are helper functions in one way or another.

Controlling the Heater

In order to control the heater, we need to measure the temperature in the mash. Having a representation of the temperature, it is a matter of comparing the current temperature with the desired temperature for the current step in the recipe. We use a PID controller to calculate the amount of power needed, but due to our experience with particles in the mash burning onto the heater, it may be necessary to limit the power. So the PID controller may call for full power, but the applied power is limited.

Measuring the Temperature: The first thing to do in the program is to locate the thermometer or the temperature sensor. I want the program to locate any DS18B20 attached to the BeagleBone. As mentioned earlier, the 1-Wire devices all have unique addresses. I currently have five or six DS18B20 thermometers on hand, so I designed the software so that I can attach any of my thermometers to the BeagleBone, and the program will locate it and start using it.

This code snippet does the trick (it is from a class I call Thermometer):

```
int Thermometer::locateThermometer()
{
  string initialDir = "/sys/bus/w1/devices/";
  string regExpr = "28-00000";
  string dir;

  // Open directory
  DIR *dp = opendir(initialDir.c_str());
  if (!dp)
  {
    exit (EXIT_FAILURE);
  }
  struct dirent *dirp;

  // Loop through the directory entries
  while ((dirp = readdir(dp)) != NULL)
  {
    std::size_t found = \
      string(dirp->d_name).find(regExpr);
    if (found != std::string::npos)
    {
      // We found one entry that matches
      oneWireDir = initialDir + string(dirp->d_name)\
        + string("/w1_slave");
      // Nicely close the directory again
      (void) closedir(dp);
      return 0;
    }
  }
  exit(EXIT_FAILURE);
}
```

`oneWireDir` is a class variable where I keep the directory and file from which I can read the temperature sensor.

The output from a DS18B20 is made up of different information. Take a look at the output below:

```
root@beaglebone:/sys/bus/w1/devices/28-000005a7ce64# cat w1_slave
85 01 4b 46 7f ff 0b 10 5f : crc=5f YES
85 01 4b 46 7f ff 0b 10 5f t=24312
root@beaglebone:/sys/bus/w1/devices/28-000005a7ce64#
```

The first line is of no interest to us. The second line shows that the temperature t is 24.312°C. From this, we learn that it is just a matter of reading the w1_slave file and grabbing the temperature from the output. I have designed this function to do the task:

```
int Thermometer::readTemperature()
{
  int fd;
  int res;

  // Open the OneWire thermometer file
  fd = open(oneWireDir.c_str(), O_RDONLY);
  if(!fd)
  {
    return -1;
  }
  char buf[256];
  // Read from it
  res = read(fd, buf, sizeof(buf));
  if (res < 0)
  {
    close (fd);
    return -1;
  }
  // Retrieve current temperature from device
  if (res > 0)
  {
    std::size_t found = string(buf).rfind('=');
    if (found != std::string::npos)
    {
      curTemp = atof((const char *) \
                &buf[found+1]) / 1000;
    }
    else
    {
      close(fd);
      return -1;
    }
  }
  close (fd);
  // Call calc average temperature
  return calcAvgTemp();
}
```

The file is opened and read, and because there are two = characters in the buffer, I perform a reverse search (from the end of the buffer rather than from the beginning) in the buffer for =. When the = is located, I convert the ASCII text to a float using `atof` and scale it with 1,000 since there are no decimal points in the readout from the sensor.

The next thing to do is to calculate the average temperature. I keep the last five samples in an array in order to calculate an average temperature. The reason for this is that I occasionally saw some spurious measurements where the temperature was a bit off. The averaging will smooth this out:

```
int Thermometer::calcAvgTemp()
{
  float t = avgTemp - curTemp;
  if ((t < -20 || t > 20) && tempSamples.size() > 4)
  {
    // Skip this measurement - it's way out of range.
    log->Write("Skipped this temperature: %f, \
               t = %d\n", curTemp, t);
    return -1;
  }
  if (tempSamples.size() > maxSamples)
  {
    // Get rid of first element
    tempSamples.erase(tempSamples.begin());
  }
  tempSamples.push_back(curTemp);
  t = 0;
  for (int i = 0; i < (int)tempSamples.size(); i++)
  {
    t += tempSamples[i];
  }
```

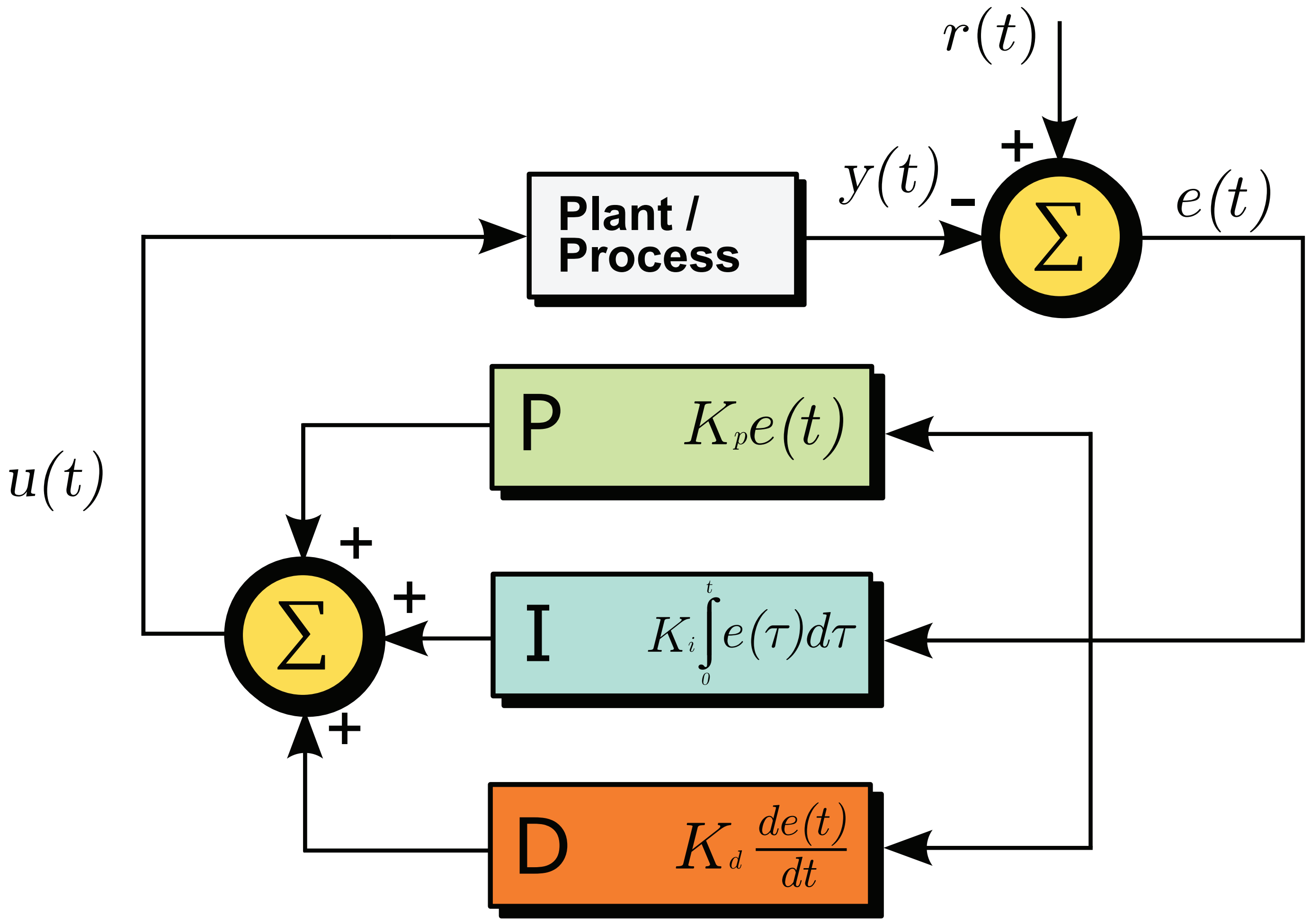


Figure 4. PID en updated feedback by TravTigerEE—Own work. Licensed under CC BY-SA 3.0 via Commons.

```
  avgTemp = t / tempSamples.size();
  return 0;
}
```

`maxSamples` currently is set to 5. As you can see, if the vector that holds the samples is filled, I drop measurements that are 20°C off the average. If the vector is not filled, I add the sample to be included—it is needed when starting the measurements.

From this point, we just need to set up, somewhere else, a timed job that calls the `readTemperature()` function. I have set it up to read every tenth second. With a mass of approximately 30kg, it is a relatively slow process.

A PID Controller: What is a PID controller? It is a proportional-integral-derivative controller. From that, you can see that it is a controller that reacts proportionally on the measured input compared to the set point (the desired value), and it accumulates the historic development of the controlled using an integral function, and finally, it tries to predict the future in the derivative part. That was a big mouthful. Let's take this one step at a time.

See the diagram of a PID controller in Figure 4 or expressed as a formula in Figure 5.

Our "plant" (as shown in Figure 4) is the brew tun. We measure, like in this project, a temperature, and elsewhere, we have a desired temperature we want to keep for a while. The difference between the measured and desired temperature is called the error—e(t) in the Figure.

The error signal is fed into the proportional, integral and derivative part of the controller.

The Proportional Controller: If the measured temperature is lower than the desired, we can fire up the heater to raise the temperature. If it is higher, we can do nothing in this setup but wait for it to cool down a little.

The amount of power that we'll apply to the heater is controlled by the error signal. If the error is small, a small amount of power is applied, and if it is large, we apply more power to the heater. By having a factor (Kp) to multiply with the error, we can amplify the error, or if the factor is below 1, attenuate the impact of the error. The proportional part of the PID controller tends to overreact—that is, the temperature overshoots the desired temperature by quite an amount. This is not desired in a process where we are seeking to give our enzymes ideal temperatures to work in. And if enzymes get too warm, they will denature—that is, die! The derivative part will compensate for this to some degree.

The Integral Controller: The integral part of the controller sums or accumulates the error over time. This part can suppress the proportional part, but with the cost of being slower to reach the desired set point.

The Derivative Controller: This part looks into the future, so to speak. It compares the previous measurement with the current one

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de}{dt}$$

Figure 5. PID Formula

in order to predict the future state of the plant. This results in a quicker raise to the desired set point, but it also will back off when nearing the set point, leaving the fine-tuning to the P and I parts.

For further details, take a look at this fine article by Wess Scott: http://m.eet.com/media/1112634/f-wescot.pdf, or see this page on Wikipedia: https://en.wikipedia.org/wiki/PID_controller.

The PID Code: It's rather easy to implement a PID controller in software:

```
// The PID Controller
float processVal = curTemp;

// The Proportional part
float error = setPoint - processVal;
float pTerm = pGain * error;

// The Integral part
iState = iState + error;
float iTerm = iState * iGain;

// The Derivative part
float dTerm = (dState - processVal) * dGain;
dState = processVal;

// Scaling to fit
float pwr = (pTerm + iTerm + dTerm) / 10;
```

The factors (Kp, Ki and Kd) that I currently use are set to:

```
pGain = 5.0;   // Kp
iGain = 0.025; // Ki
dGain = 30.0;  // Kd
```

I let the proportional and derivative parts have quite a large impact, and the integral part is dampened quite a bit.

So after getting the PID controller set and working as desired (although some fine-tuning can be done on the Kn parameters in order to find the optimal for my set up), it's time to turn to the `PowerController`.

Controlling the Heater: The output from the Thermometer class is fed into the `PowerController`. This class uses the PID controller to calculate the amount of power to apply.

As mentioned before, I had to limit the amount of power applied even if the PID controller called for more. The reason is that below 57°C–59°C, we have determined that if we apply one-third of the available power, we will not cause anything to burn onto the heater. Therefore, I have set some limits, regardless of how much power the PID controller calls for.

But hey, what about infusion? Infusion is the point in time where we add the malt to the preheated water. It's also called the strike temperature. The colder malt mixes with the water and cools it a little. Therefore, the infusion temperature can be calculated from the mass of water

$$T_{strike} = \frac{T_{Mash} * \left(V_{water} + (0.4 * W_{Malt})\right) - (0.4 * W_{Malt} * T_{Malt})}{V_{water}}$$

T_{strike} : strike temperature in Celsius [°C]

T_{Mash}: desired mash temperature in Celsius [°C]

T_{Malt}: The temperature of the malt in Celsius [°C]

V_{Water}: volume of water in litres [L]

W_{Malt} : Weight of malt in kilos [kg]

Figure 6. Strike Temperature Calculation

and malt using the amount of them both and the temperature of the malt (Figure 6). So if the desired infusion (or strike) temperature is 55°C, the amount of water is 19 liters, and the 7kg of malt is 14°C, we can calculate that the water should be preheated to 59.8°C before adding the malt. After a short while, it'll find its rest at 55°C.

In order to get ready for brewing as quickly as possible, we can heat the water with full power until we add the malt. So we need to know the different steps of the mashing process, which is kept in a database (more about that later). From that, we get the set point.

I have put the maximum temperature ranges into a two-dimensional array of floats:

```
tempPwrRange[0][0] = 57;     // Below this temp. use
tempPwrRange[0][1] = 0.333;  // this amount of pwr
tempPwrRange[1][0] = 60;     // Below this temp. use
tempPwrRange[1][1] = 0.666;  // this amount of pwr
tempPwrRange[2][0] = 102;    // Below this temp. use
tempPwrRange[2][1] = 1.0;    // this amount of pwr
```

Then, it's simple to compare against the current temperature and find the maximum amount of power that may be applied:

```
// Locate the temperature range to operate within
// Use the current temperature measured to retrieve
// the maxPwr
for (int i = 0; i < 3; i++)
{
  limitTemp = tempPwrRange[i][0];
  maxPwr = tempPwrRange[i][1];
  if (curTemp > limitTemp)
  {
    continue;
  }
}
```

After having the PID controller calculate the desired power, it is left to the `setPwrLvl` function to find what can be set according to the current step in the recipe from where we have derived the `maxPwr`:

```
float PowerController::setPwrLvl (float pwr,
                                  float maxPwr)
{
  // The PWM operates with a high granularity,
  // hence we have to scale the pwr

  // Corrects the duty cycle when applied to the PWM
  const long long Factor = 999000000;
  log->Write("PowerController: maxPwr is set to %f",\
           maxPwr);

  // If the requested pwr is less that 0
  // we set it to 0
  if (pwr < 0)
  {
    // Set PWM to 0
    if (pwm.setDutyCycle (0) == 0)
    {
      log->Write("PowerController: PWM set to 0\n");
    }
    else
    {
      log->Write("PowerController: ERROR: \
               PWM not set! (pwr<0)\n" );
    }
    return 0.0;
  }

// If we are in the infusion step it is allowed to use max pwr
if (infusion)
{
  if (pwm.setDutyCycle (maxPWM) == 0)
  {
    log->Write("PowerController: INFUSION POWER:\
               PWM set to     %f\n", maxPWM);
  }
  else
  {
    log->Write("PowerController: ERROR: PWM not\
               set! value: %f\n", maxPWM);
  }
  return 1.0;
}

// If the pwr is less than the maxPwr allowed
// use the requested pwr
if (pwr < maxPwr)
{
  // Set PWM to pwr
  if (pwm.setDutyCycle (pwr * Factor) == 0)
  {
    log->Write("PowerController: PWM set to \
               %f\n", pwr * Factor);
  }
  else
  {
    log->Write("PowerController: ERROR: PWM not \ set!\n");
  }
  return pwr;
}

// Otherwise use the maximum power, i.e. 100%
else
```

```
  {
    // Set PWM to the maxPwr
    if (pwm.setDutyCycle (maxPwr * Factor) == 0)
    {
      log->Write("PowerController: PWM set to \
                 %f\b", maxPwr * Factor);
    }
    else
    {
      log->Write("PowerController: ERROR: PWM not \
                 set!\n");
    }
    return maxPwr;
  }
  return 0.0;
}
```

The `setDutyCycle` function actually writes to the PWM file that represents the PWM controller on board:

```
int PWMController::setDutyCycle(long long duty)
{
  string dir;

  dir = initialDir + "/duty";

  // Open the file representing the PWM controller
  int f = open(dir.c_str(), O_WRONLY);
  if (f < 0)
  {
    log->Write("ERROR Opening file %s with this \
                error: %s\n",
                dir.c_str(),
                strerror( errno ));
    return -1;
  }

  // Write the requested duty cycle to the file
  char str[100];
  sprintf(str, "%lld", duty);
  size_t res = write (f, str, strlen(str));
  close (f);

  // Handle errors
  if (res != strlen (str))
  {
    log->Write("ERROR Opening file %s with this \
                error: %s\n",
                dir.c_str(),
                strerror( errno ));
    return -1;
  }
  return 0;
}
```

Yes, I know—I am a C hacker (mis-)using the fine C++ language with file operations in plain C code. But, so it is! What comes easiest to the fingers over the keyboard is what goes in the code—sometimes I think there may not be a brain involved at all. But otherwise, it's straightforward code.

This concludes the measuring of the temperature and controlling the power output. In a future article, I'll look into adding an SQLite database to the brewController program. In order to service a Web page, we also need a WebSocket server, so I will introduce that as well.

The database has two purposes: storing recipe details for the mash process and continuous logging of data produced during a brew.

The WebSocket server is the interface to the Web page from where we can monitor and control the brewing process. So I also will dive into a relatively complex Web page with a lot of JavaScript code to control the graphs and retrieval of data from the WebSocket server, which will look up in the database for the correct data to serve.

Finally, I'll write about how to dæmonise the brewController and make it start automatically when the BeagleBone is powered.

All the code will be open-sourced when it has proven to control a few brews this autumn. If you are in a hurry, send me an e-mail and I will send you the code.■

Klaus Kolle is currently teaching electronic engineering students software development and device driver development in the Linux kernel at Aarhus University. Klaus has been working and developing on UNIX and Linux since 1988. Klaus loves well brewed beer, especially if it is home-brewed.

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

Resources

You can retrieve a cross compiler for the ARM processor at http://releases.linaro.org/14.11/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2014.11-x86_64_arm-linux-gnueabihf.tar.xz.

My instructions to set up the remote debugging in Eclipse: http://klaus.ede.hih.au.dk/index.php/BBB_Remote_Debugging

About the Flattened Device Tree (FDT): http://elinux.org/Device_Tree

A Web page for setting up the Device Tree specification files: http://kilobaser.com/blog/2014-07-28-beaglebone-black-devicetreeoverlay-generator#dtogenerator

A simple log class: http://www.infernodevelopment.com/c-log-file-class-forget-debuggers

The technical specifications for the BeagleBone Black: https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf?raw=true

PID without a PhD: http://m.eet.com/media/1112634/f-wescot.pdf

Wikipedia on PID controllers: https://en.wikipedia.org/wiki/PID_controller

Command-Line Tutorial:

Does Every Year Have a Friday the 13th?

Can you write a one-liner that answers this question for a given year? Can you write a short script around your one-liner and solve this puzzle for all time?

SOL LEDERMAN

Here's a fun little math problem that a handful of Linux commands, a bit of thought and a short shell script will help solve: does every year have a Friday the 13th? `cal`, `cut`, `grep`, `sed` and `cksum` will be your friends in this exploration.

Before engaging the computer, let's consider how you might solve this problem without one; you could examine some number of calendars. If you discover a year's calendar with no Friday the 13th, you're done. Otherwise, you eventually will come to believe that every year does have a Friday the 13th. "Eventually" is a long time away though. So, how many calendars do you need to look at? You may want to pause to consider this question before reading further. And, to support your exploration, you may want to use the Linux `cal` command to display calendars to your terminal.

Welcome back! Did you count 14 different calendars that you need to consider? Did you count seven? Both answers are correct, depending on your approach. (If you don't believe that looking at seven calendars is enough, see the suggested explorations at the end of the article.) Let's also consider leap years for this exploration. And, there's another consideration: is it enough to inspect the calendars for 14 consecutive years? Ponder that for a bit.

Let's dive in. How many unique calendars are there? Two calendars are "unique" if they look different—that is, if some months start on different days when you compare the two. Two calendars are the same if you could hang either one on the wall, and except for the year printed at the top of the calendar, you couldn't tell the difference. 2014 started on a Wednesday, and 2015 started on a Thursday. Those are two unique calendars. 2006 and 2012 both started on a Sunday. Are they the same calendar? Could you swap one for the other? No, because 2012 was a leap year, but 2006 wasn't. So, although January of both calendars was the same for those years, the other months weren't.

Considering that some years are leap years and that others aren't leads to one approach to counting calendars. Every year starts on one of the seven days. If you don't consider leap years (and there is an approach where you don't need to consider them), then there are seven different calendars. If you also consider leap years, you have seven more calendars to consider. So, if you examine 14 unique calendars, you will have the answer. You either will find a year that doesn't have a Friday the 13th or you won't.

Here is the part of the analysis where the computer is going to help. You need to look at 14 unique calendars. Will it be enough to look at the calendars for 2000 through 2013? That's a 14-year span. It turns out that that span does not include 14 unique calendars. In particular, 2001 and 2007 are the same calendar. Rather than trying to think through how many calendars you need to look at, let's take a different approach. Let's write a shell script that will review calendars until it has looked at 14 unique ones.

How can you make the computer tell you if calendars for two years are the same? Let's take a detour away from that question for a short while and consider this command: `cksum`. `cksum` displays the checksum of a file (or of stdin). What's a checksum? A checksum is a number associated with a file that is commonly used to detect transmission errors.

Imagine that you send a file from one computer to another via a noisy phone line. You want some assurance that the file wasn't corrupted during the transmission. What could you do? You could compare the sizes of the files on both ends. If the sizes are different, you would know the file copy is corrupt. But, if the sizes are the same, you certainly can't be confident that the content of the files is the same. You could transmit the file twice and compare the two copies byte by byte, but that would double the transmission time.

Checksums solve the problem of verifying that two files are identical without incurring a large computational or transmission cost.

Checksums solve the problem of verifying that two files are identical without incurring a large computational or transmission cost. The Linux `cksum` utility computes a single number—the checksum, for a file. The process for using the checksum is this: compute the checksum, transmit the file, compute the checksum for the received file and compare the checksums. If the checksums are identical, there is a near-100% probability that the two files are identical. Near-100% is quite good enough for most purposes.

Now, let's return to the calendar problem. You want to find 14 unique calendars and check each to see if they have a Friday the 13th. Here's some pseudo-code that illustrates this approach:

```
Year = 2000         # Year increases until we
                    # have seen 14 unique
                    # calendars
ChecksumCount = 0   # This counts how many unique
                    # calendars we've seen
While ChecksumCount < 14 {
     Compute the checksum for Year
     If we have not seen this checksum yet {
          Add this checksum to the list of
             checksums we've seen
          Add 1 to ChecksumCount
     }
     Add 1 to Year
}
```

Why do you use checksums to compare calendars? Why don't you just create a file for each unique calendar and compare each calendar to every saved file? You could do that. But, once you work through the checksum approach, I think it will become clear that comparing files, with the `diff` command or some other approach, is clunkier. You decide.

It's time to dive in to some shell commands. I mentioned earlier that 2001 and 2007 are the same calendar. Compare the output of these two commands:

```
cal 2001
cal 2007
```

Now, compute the checksum of each calendar by piping the output of `cal` to `cksum`:

```
cal 2001 | cksum
3673415557 2014

cal 2007 | cksum
2655244645 2014
```

Hmmm. The two checksums are not the same. Why is that? (Note that "2014" is the number of characters in the output of `cal`. Ignore that and just look at the first number.) It's the title in the output of `cal` (the year) that makes the two calendars appear to be different. All you need to do is remove the first line, and you'll be good. `sed` is one of a number of Linux tools that easily can remove the first line of the output:

```
cal 2001 | sed '1d' | cksum
2408573533 1980
cal 2007 | sed '1d' | cksum
2408573533 1980
```

Note that `cksum` may generate a different checksum for your `cal` output. That's because `cal` has slightly different layouts on some flavors of Linux. As long as you're comparing `cal` output on the same machine, everything will work correctly.

`cksum` has confirmed that 2001 and 2007 are the same calendar. The next task is to build a list of the unique checksums that you've seen so that you can compare new calendar checksums to those on the list. Let's use a string to hold that list and initialize it to be empty:

```
checksums=""
```

Then, add the checksum for 2001 to the list:

```
checksums="$checksums 2408573533"
```

In order to add just the checksum to the list and not the number of characters in the output of `cal`, you will need to keep just the first part of `cksum`'s output and discard the second part. There are many ways to do this in bash. Here's one way:

```
cal 2001 | sed '1d' | cksum | awk '{print $1}'
2408573533
```

Awk is a very powerful text and data processing Linux tool. If you've never used awk, it's worth reading a tutorial about it and exploring it. For the purposes of this article, all you're doing with awk is telling it to print the first (whitespace-separated) field of the input ($1) from the pipe.

With the new checksum, 2408573533, use `grep` to search the list of checksums you've seen for this new one:

```
echo "$checksums" | grep -w 2408573533
```

Note that it's a good idea to use the `-w` flag with `grep` to match only 2408573533 as a word, with spaces, tabs and punctuation separating words. You don't, for example, want to match 2408573533123456789.

You also will want to use `grep` with the `-q` option when you write your shell script to count Friday the 13ths. The option tells `grep` to search quietly and not to display any output. You then will use the return status from `grep` to know whether it matched anything:

```
echo $checksums | grep $newChecksum
if [ $? -eq 0 ] # if return value is 0
then
   # we matched something
fi
```

Whenever you see a unique

calendar, increment a counter:

```
numUniqueCalendars=$((numUniqueCalendars+1))
```

`$((..))` is a way you can do simple arithmetic in bash. Note that inside the `$((..))`, you don't put dollar signs in front of variable names.

When the counter reaches 14, you've seen all unique calendars.

Now that you have a general idea about how to use checksums, strings and `grep` to determine whether a calendar is unique and to count unique calendars, there is one more significant task. Given a calendar, how do you know if it has a Friday the 13th? Let's look at the output from `cal` for just one month:

```
cal 4 2001
     April 2001
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

April 13, 2001, is a Friday. It's easy enough for us humans to scan the Friday column looking for 13. How might a computer do the same? If you're not familiar with the Linux `cut` command, take a look at its man page and play with it a little before reading further.

`cut` allows you to select one or more columns. Sunday dates are in columns 1 and 2; column 3 is blank; Monday dates are in columns 4 and 5; and, if you keep counting columns, you see that Friday dates are in columns 16 and 17. Pipe the output of `cal` to `cut`, and you get all the Friday dates:

```
cal 4 2001 | cut -c16-17
Fr
 6
13
20
27
```

All you need now is to `grep` these Friday dates for the number 13:

```
cal 4 2001 | cut -c16-17 | grep 13
13
```

So, you now have a Linux pipeline that will tell you if a given year and month have a Friday the 13th. How can you generalize this approach to scan an entire year? Note that `cal` displays entire years in a 3x4 layout of three months across and four months down. You can use `cut` again and have it display three columns of Fridays. Note that you will need to include a column with a space, either before or after the dates, otherwise the column values will run together:

```
cal 2001 | cut -c16-18,38-40,60-62
```

Note that the column ranges may be different in your version of Linux. I had to change column numbers to get the output to be correct on one of my Linux machines. Here is some of the output:

```
Fr Fr Fr
 6  4  1
13 11  8
20 18 15
27 25 22
      29


Fr Fr Fr
 6  3
13 10  7
20 17 14
27 24 21
   31 28
```

Notice that there are two Friday the 13ths in 2001. Let's add `grep` to the pipeline to filter out just the lines with 13 in them and use `wc` to count those lines:

```
cal 2001 | cut -c16-18,38-40,60-62 | grep -o 13 | wc -l
2
```

You may be wondering why you need the `-o` option to `grep`. Consider this:

```
echo 12 13 14 13 | grep 13
12 13 14 13
```

The number 13 appears twice in the list that you `echo`, but `grep` is matching the entire line, and when you pipe the output of `grep` to `wc -l` to count matches, you get this:

```
echo 12 13 14 13 | grep 13 | wc -l
       1
```

It would be nice to know how many Friday the 13ths there are in a year. This is where the `-o` option is useful:

```
echo 12 13 14 13 | grep -o 13
13
13
```

Adding `-o` tells `grep` to print only the matching part of lines and to print matches on multiple lines. Now you easily can count Friday the 13ths for any year. How many Friday the 13ths were there in 2014? Let's see:

```
cal 2014 | cut -c16-18,38-40,60-62 | grep -o 13 | wc -l
1
```

Congratulations! You've laid all of

the groundwork and should now be able to write a shell script to answer this article's burning question!

Let's summarize the programming approach used here. This should look very similar to the pseudo-code earlier in this article with one addition: you are going to count the number of Friday the 13ths in each unique calendar.

1. Pick a starting year. Does it matter what that year is?

2. For this year, compute its calendar checksum.

3. If the checksum is not on the unique checksum list, then you are looking at a calendar that you've not looked at previously. Add its checksum to your unique checksum list, and add one to the number of unique calendars you've seen. Count the number of times Friday the 13th appears in this year.

4. If you've seen 14 unique calendars, you're done. If not, add one to the year and loop back to step 2.

Here is a bash script that puts all of these ideas together. Note that the lines ending in \ need to not have a space, tab or any other character after them.

```
#!/bin/bash
# Starting with the year 2000, we
# compute how many times
# Friday the 13th occurs in every
# year. We stop when we have
# looked at 14 different calendars.
year=1999
checksums=""
numUniqueCalendars=0
echo "Year     Number of Friday the 13ths"
while [ $numUniqueCalendars -lt 14 ]
do
   year=$((year+1))
   sum=$(cal ${year} | sed '1d' | cksum |\
      awk '{print $1}')
   echo "$checksums" | grep -q $sum
   if [ $? -ne 0 ]
   then # It's a new calendar
      checksums="$checksums $sum"
      numUniqueCalendars=$((numUniqueCalendars+1))
      echo -n "${year} "
      cal ${year} | cut -c16-18,38-40,60-62 |\
         grep -o 13 | wc -l
   fi
done
```

Here's the output:

```
Year     Number of Friday the 13ths
2000         1
2001         2
2002         2
```

```
2003        1
2004        2
2005        1
2006        2
2008        1
2009        3
2010        1
2012        3
2016        1
2020        2
2024        2
```

Notice that you had to examine 25 consecutive calendars (between 2000 and 2024) to find 14 unique ones. And, notice...drum roll... Friday the 13th occurs once, twice or three times in every year.

I chose to start checking calendars starting in 2000. What would have happened if I had chosen some other year?

I hope you appreciate the power of a handful of Linux commands, some logical thinking and a short shell script to solve a fun math challenge. If you enjoyed this problem, consider exploring these related problems:

1. How would using `ncal` instead of `cal`, if it's available on your system, simplify your code?

2. Does every year have a Monday the 13th? What's the most any year can have? For this exploration (and for #6), you'll need to generalize the concept of pulling out one or more columns from a calendar to be able to select any day of the week.

3. If you exclude January and February, does every year still have a Friday the 13th?

4. 2015 has three Friday the 13ths. When will that happen next?

5. How likely is the 13th of a month to fall on a Friday vs. the other days of the week? Have the computer scan every month for 100 years of calendars to find out.

6. Which day/date pairs occur every year? For example, does every year have a Monday the 31st? This exploration generalizes #2.■

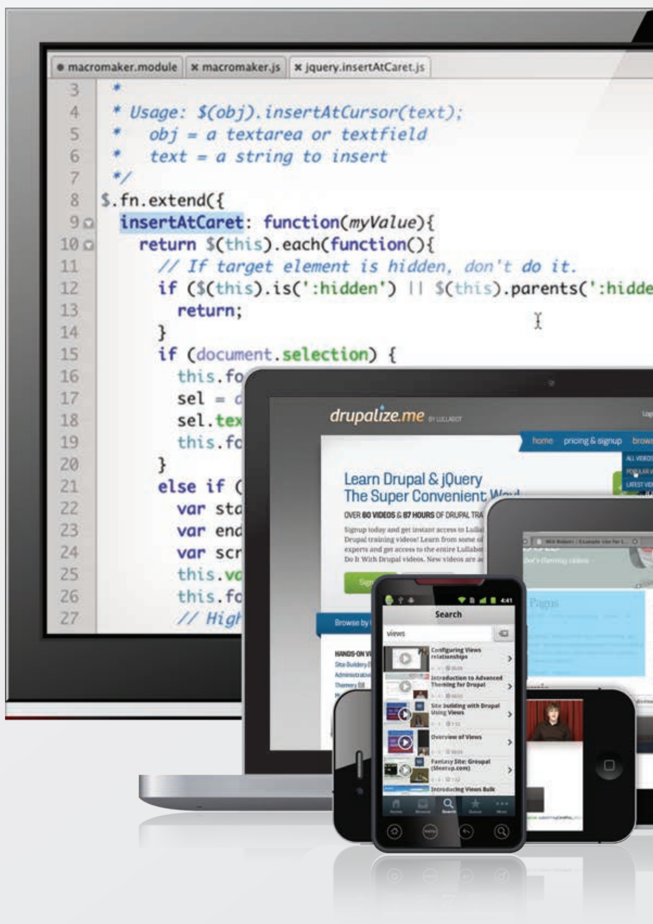
Sol Lederman is a math geek and general techie. As a writer and programmer, he especially loves to communicate technical things to a broad audience. Read more about Sol at http://sollederman.com.

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

drupalize.me

Instant Access to Premium Online Drupal Training

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



```
  *
  * 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 = d
        sel.tex
        this.fo
      }
      else if (
        var sta
        var end
        var scr
        this.va
        this.fo
        // High
```

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!

WEBCASTS

Maximizing NoSQL Clusters for Large Data Sets

Sponsor: IBM

This follow-on webcast to Reuven M. Lerner's well-received and widely acclaimed Geek Guide, "Take Control of Growing Redis NoSQL Server Clusters", will extend the discussion and get into the nuts and bolts of optimally maximizing your NoSQL clusters working with large data sets. Reuven's deep knowledge of development and NoSQL clusters will combine with Brad Brech's intimate understanding of the intricacies of IBM's Power Systems and large data sets in a free-wheeling discussion that will answer all your questions on this complex subject.

> **http://geekguide.linuxjournal.com/content/maximizing-nosql-clusters-large-data-sets**

How to Build High-Performing IT Teams — Including New Data on IT Performance from Puppet Labs 2015 State of DevOps Report

Sponsor: Puppet Labs

DevOps represents a profound change from the way most IT departments have traditionally worked: from siloed teams and high-anxiety releases to everyone collaborating on uneventful and more frequent releases of higher-quality code. It doesn't matter how large or small an organization is, or even whether it's historically slow moving or risk averse — there are ways to adopt DevOps sanely, and get measurable results in just weeks.

> **http://geekguide.linuxjournal.com/content/how-build-high-performing-it-teams-including-new-data-it-performance-puppet-labs-2015-state**

WHITE PAPERS

Comparing NoSQL Solutions In a Real-World Scenario

Sponsor: RedisLabs | Topic: Web Development | Author: Avalon Consulting

Specializing in cloud architecture, Emind Cloud Experts is an AWS Advanced Consulting Partner and a Google Cloud Platform Premier Partner that assists enterprises and startups in establishing secure and scalable IT operations. The following benchmark employed a real-world use case from an Emind customer. The Emind team was tasked with the following high-level requirements:

- Support a real-time voting process during massive live events (e.g., televised election surveys or "America Votes" type game shows).
- Keep voters' data anonymous but unique.
- Ensure scalability to support surges in requests.

> **http://geekguide.linuxjournal.com/content/comparing-nosql-solutions-real-world-scenario**

NEW Forrester Study!

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

Achieving Application Delivery Velocity with a 482% ROI

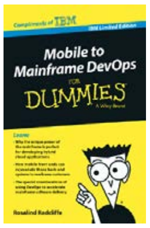
IBM commissioned Forrester Consulting to conduct its Total Economic Impact™ (TEI) study that examines and quantifies potential return on investment (ROI) for IBM UrbanCode Deploy within an enterprise DevOps environment. The study determined that a composite organization, based on the customers interviewed, experienced an ROI of 482%!

Read the Forrester Consulting study and learn learn how these enterprise organizations achieved:

- 97% reduction in the cost of releases.
- Reduction in the risk of failed deployments.
- 75% faster deployment times.

See how IBM UrbanCode brings deployment velocity while reducing release costs.

> http://devops.linuxjournal.com/devops/total-economic-impacttm-ibm-urbancode



Mobile to Mainframe DevOps for Dummies

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

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

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

BRAND-NEW EDITION!

DevOps For Dummies – New Edition with SAFe®

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

Download the E-book to learn about:

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

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

DOC SEARLS

Giving Silos Their Due

We're not wrong in our principles or our work. But, we may have to admit that silos have proven right in some big ways.

Two things I got way wrong, way back.

One was Linux on the Desktop (LOTD, http://www.linuxjournal.com/googlesearch?s=lotd). Around the turn of the Millennium, I predicted big successes for LOTD and Linux on the Laptop (LOTL, http://www.linuxjournal.com/article/7464)—and continued to do the same, annually, until I gave up. Here's how my optimism looked in January 2003 (http://www.linuxjournal.com/article/6548):

> Is 2003 the year we see big-name hardware companies selling Linux as aggressively as they sell Windows—by which I mean Linux PCs show up in stores and on the front pages of Web sites?
>
> Paul Saffo famously said we overestimate in the short term and underestimate in the long. But I'm going to go out on a limb and say the long term is mostly behind us. (How many years have we been waiting for LOTD to take off?) Somebody is going to break the ice this year. My own instinct says it'll be IBM, mostly because there must be serious demand for LOTD (as well as improvised solutions with it) inside the company. You can't have that many engineers walking around with Linuxified ThinkPads and not get around to selling them at some point.
>
> I'm betting Dell will be next. Then Gateway. Then HP....

Didn't happen. Sure, some of the bigs sold personal Linux

Which brings me to the second thing I got wrong: expecting the world to prefer flat, distributed, open and free technology ecosystems over silo'd centralized, closed and proprietary ones.

hardware (and still do), but not "as aggressively as they sell Windows". If you're not a Linux geek today, your choices of desktops and laptops are contained by two silo'd operating systems from just two companies: Apple and Microsoft. Hardware choices are also narrower since Microsoft decided to follow Apple into the hardware business.

The center of device use gravity has also moved over to mobile, where most of us are using gear running just two operating system platforms: Apple's iOS and Google's Android. True, Android is based on Linux (a victory we gladly claim), and the range of devices running Android is huge (http://techcrunch.com/2014/08/21/opensignal-2014-android-ecosystem-report), but Android is still Google's show. The network frontier is also mostly fenced off by mobile carriers and device suppliers who believe, almost across the board, that captive customers and users are more valuable than free ones—and everybody agrees, at least tacitly, that "free market" means "your choice of captor".

Which brings me to the second thing I got wrong: expecting the world to prefer flat, distributed, open and free technology ecosystems over silo'd centralized, closed and proprietary ones. One example is XMPP (https://en.wikipedia.org/wiki/XMPP), originally called Jabber. It was meant to bridge or replace all the competing proprietary instant-messaging systems in use at that time: AOL's AIM and ICQ, Microsoft's MSN, Yahoo's Messenger, Apple's whatever-it-was (now called iMessage) and so on.

Didn't happen. Look up "Jabber" today at LinuxJournal.com and you'll get a *lot* of results, also from back around the turn of the Millennium, mostly written by me. In those days, many of us had full confidence that Jabber/XMPP would

Wikipedia currently lists 55 sites, services, protocols and programs that do chat. There is no underlying protocol for interoperation among them.

do for instant messaging (aka chat) what SMTP/POP3/IMAP did for e-mail and HTTP/HTML and its successors did for publishing and all the other things one can do on the World Wide Web. We would have a nice flat, distributed and universal standard that people could employ any way they wanted, including on their own personal hardware and software, with countless interoperable systems and no natural barriers to moving data easily from any one system to any other.

Didn't happen. Yes, XMPP turned into a widely used standard and either supplied or inspired lots of messaging systems. But today, instant messaging is almost entirely silo'd, and it's more popular than ever. Facebook (including WhatsApp), Twitter, Google and LinkedIn have their own messaging systems inside their own "social" silos. So do Apple and Microsoft (including Skype). Tencent Holdings, the giant Web portal company in China, has Tencent QQ, WeChat and Qzone (a social site with messaging built in). Some interoperate, but most don't, since that would free their users from captivity.

Wikipedia currently lists 55 sites, services, protocols and programs that do chat. There is no underlying protocol for interoperation among them. Nor do any makers seem especially interested. Google Talk (https://developers.google.com/talk) interoperated with other XMPP systems, but then it was replaced with Google Hangouts (http://windowspbx.blogspot.com/2013/05/hangouts-wont-hangout-with-other.html), which lacks support for XMPP. So, while XMPP.org says "it's not the end of XMPP for Google Talk" (https://xmpp.org/2015/03/no-its-not-the-end-of-xmpp-for-google-talk), it hardly matters if Google Talk is itself dead.

The last nail in XMPP's coffin, perhaps, is this from Facebook (https://developers.facebook.com/docs/chat):

> On April 30, 2014, we announced the deprecation of the XMPP Chat API as part of the release of Platform API v2.0.
>
> After April 30th, 2015 apps will no longer be able to access the service or API. This includes both access to chat.facebook.com and the xmpp_login permission.
>
> We recommend people access Facebook Messages on the desktop via Facebook.com or Messenger.com.

On the other hand (or on another tentacle), the number and variety of chat platforms and apps is huge, and growing. Apple's App Store currently lists 171 chat apps, while Google's Play Store for Android lists 165 (https://play.google.com/store/search?q=chat&hl=en). Nearly all of them are proprietary, and many also depend on (or work inside) large silo'd proprietary services. Listen for complaints about all this and you'll hear crickets.

APIs are also silos. As more and more of the connected world comes to depend on Whatever as a Service, "the cloud" and other remote and centralized services, the very idea of distributed capacities

ADVERTISER INDEX

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

ADVERTISER	URL	PAGE #
Drupalize.me	http://drupalize.me	97
Great Wide Open	http://www.greatwideopen.org/	31
HPC Wallstreet	http://www.flaggmgmt.com/linux	45
O'Reilly Fluent	http://conferences.oreilly.com/fluent/javascript-html-us	39
Peer 1 Hosting	http://go.peer1.com/linux	7

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.

So now the chat ecosystem is a forest of silos standing on free and open-source geology, but not built with the same rock.

on standalone open-source (or open-source-based) hardware and software that individuals or organizations fully control as independent and sovereign entities in the world, seems terribly retro, utopian, or both.

Brian Behlendorf says, wisely, that the ideal is "minimum viable centralization". I think that should be a transcendent design principle. But the centralized thing is too often the easiest to make or to rely on (as is the case with APIs and big silo'd platforms). So now the chat ecosystem is a forest of silos standing on free and open-source geology, but not built with the same rock.

What may be hardest for old-skool types like me to admit is that silos are now the preferred method for inventing and promulgating technologies supporting uses that scale and invite incompatible competitors to do the same, while the marketplace as a whole doesn't have a big problem with any of it.

I'll never believe silos are the best way to make the world work in the long run. And I'll always believe that the flat distributed world built on free and open stuff is the most supportive and fertile base on which to build the best and broadest range of goods and services. But I'm past believing that this will ever be obvious to the world of developers, businesses and governing bodies. Muggles have always outnumbered wizards and always will.

Being out of sight and mind for most of the world doesn't make us wrong. In fact, it might just make us right in ways most will never see, no matter how much they depend on what we do.■

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.