jQuery | Gauger | Moose | Qt4 Designer | GNU Awk | jEdit

LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community
SEPTEMBER 2011 | ISSUE 209 | www.linuxjournal.com

Make Utility Primer

What's New in **GNU Awk 4.0**

WaveMaker for Rapid Application Development

Modern Development with **Perl** and **Moose**

PROGRAMMING

DEVELOP GUIs with Qt4 Designer and Eclipse

MULTIPLATFORM DEVELOPMENT Using GNU Libraries and Tools

USE GAUGER for Performance Regression Testing

PLUS:

GETTING STARTED WITH **jEDIT**

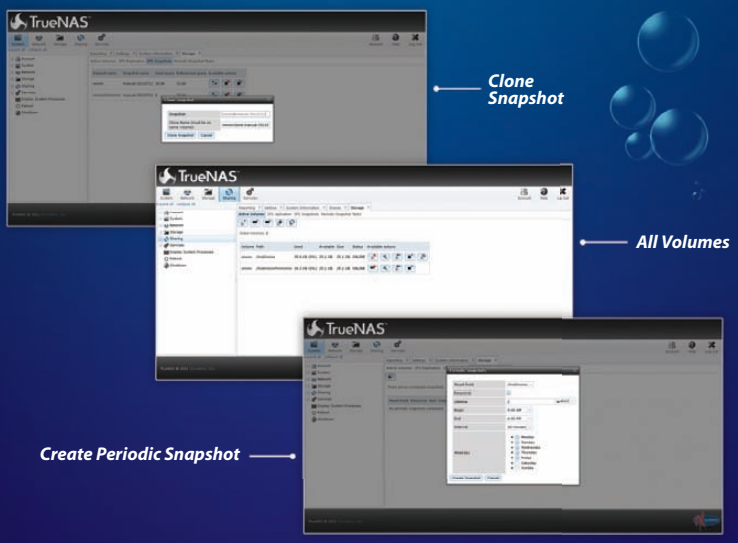
TrueNAS™ Pro 2U Appliance:
You Are the Cloud

FreeNAS POWERED

Storage. Speed. Stability.

With a rock-solid FreeBSD® base, Zettabyte File System (ZFS) support, and a powerful Web GUI, TrueNAS™ Pro pairs easy-to-manage FreeNAS™ software with world-class hardware and support for an unbeatable storage solution. In order to achieve maximum performance, the TrueNAS™ Pro 2U System, equipped with the Intel® Xeon® Processor 5600 Series, supports Fusion-io's Flash Memory cards and 10 GbE Network Cards. Titan TrueNAS™ Pro 2U Appliances are an excellent storage solution for video streaming, file hosting, virtualization, and more. Paired with optional JBOD expansion units, the TrueNAS™ Pro System offers excellent capacity at an affordable price.

For more information on the **TrueNAS™ Pro 2U System**, or to request a quote, visit: **http://www.iXsystems.com/TrueNAS**.



KEY FEATURES:

- Supports One or Two Quad-Core or Six-Core, Intel® Xeon® Processor 5600 Series
- 12 Hot-Swap Drive Bays - Up to 36TB of Data Storage Capacity*
- Periodic Snapshots Feature Allows You to Restore Data from a Previously Generated Snapshot
- Remote Replication Allows You to Copy a Snapshot to an Offsite Server, for Maximum Data Security
- Up to 4.48TB of Fusion-io Flash Memory
- 2 x 1GbE Network interface (Onboard) + Up to 4 Additional 1GbE Ports or Single/ Dual Port 10 GbE Network Cards

JBOD expansion is available on the 2U Pro System

* 2.5" drive options available; please consult with your Account Manager

iXsystems®

Call iXsystems toll free or visit our website today!

1-855-GREP-4-IX | www.iXsystems.com

intel Xeon® inside™

Powerful. Intelligent.

1&1 DUAL HOSTING

THE NEW STANDARD IN WEB HOSTING

No other web host offers more expertise, know-how and quality service than 1&1.

- ✓ **Double Security:** Your website is simultaneously hosted in 2 locations in our high tech data center!
- ✓ **High-speed Global Network:** 210 GBit/s Connectivity
- ✓ **Environmentally Responsible:** 100% Renewable Energy
- ✓ **Solid Technical Foundation:** 1,000 In-house Developers

SUMMER SPECIAL: 1&1 DUAL ADVANCED PACKAGE

1 YEAR FREE!*

- 2 FREE Domains
- FREE Private Domain Registration
- DNS Management
- 500 E-mail Accounts
- 150 GB Web Space
- DNS Management
- 50 FTP Accounts
- 1&1 SiteAnalytics
- ASP, .NET, AJAX, LINQ, PHP5, Perl, SSI
- 5 Microsoft® SQL Databases
- Mobile Website Optimization Software
- 24/7 Toll-free Customer Support

OFFER ENDS 08/31/11

Need more domains?
.com with FREE Private Registration just $4.99/first year.*

MEMBER OF united internet

1-877-GO-1AND1 www.1and1.com
1-855-CA-1AND1 www.1and1.ca

1&1®

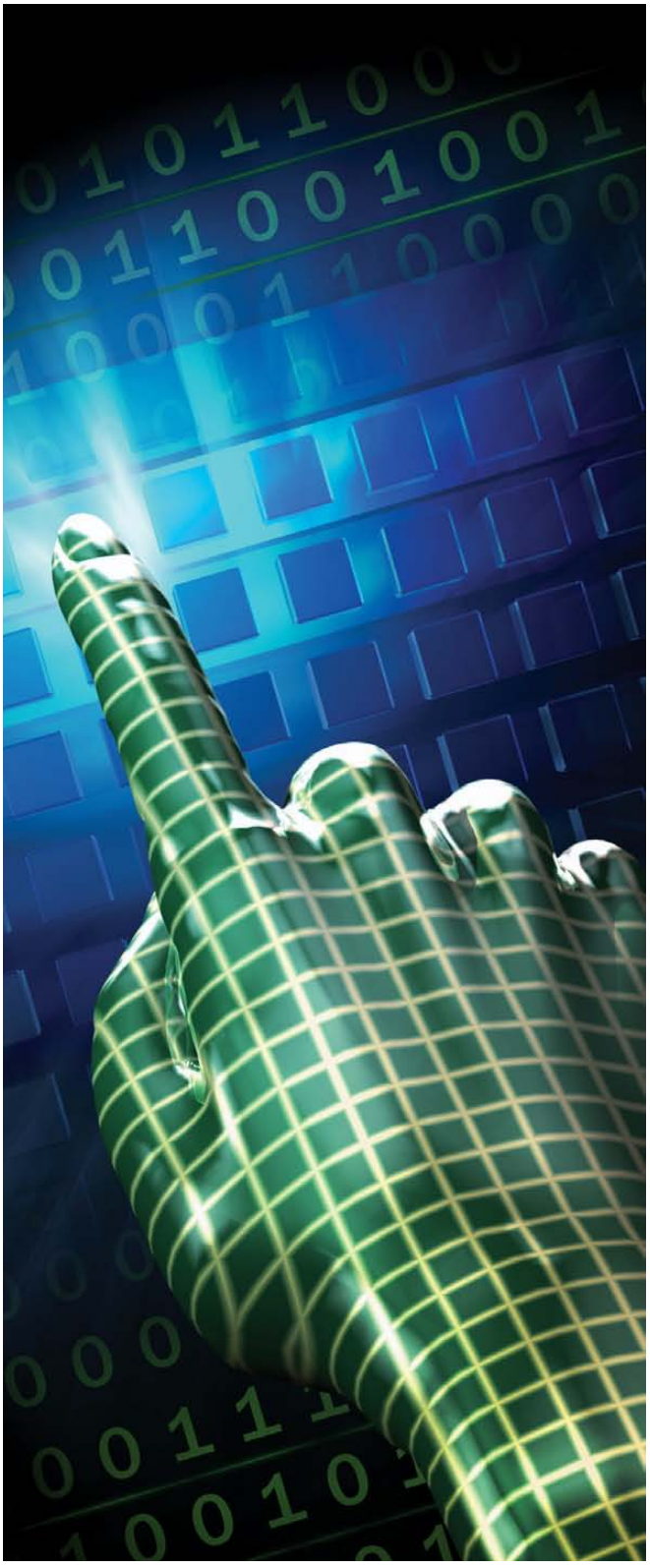
* Offers valid through August 31, 2011. 24 month minimum contract term required for Dual Advanced offer. Set-up fee and other terms and conditions may apply. .com price valid first year only. After first year, standard pricing applies. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet AG, all other trademarks are the property of their respective owners.

CONTENTS

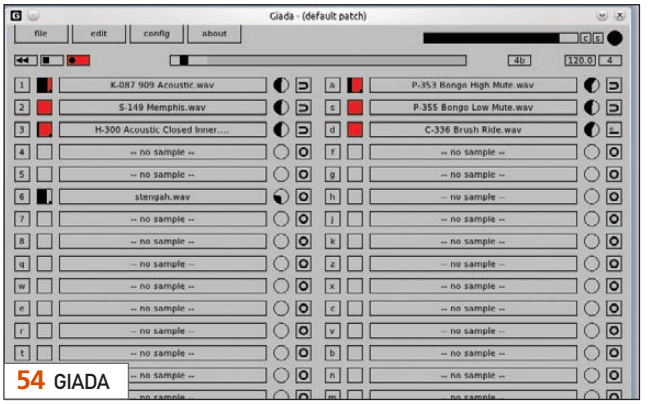
SEPTEMBER 2011
Issue 209

PROGRAMMING

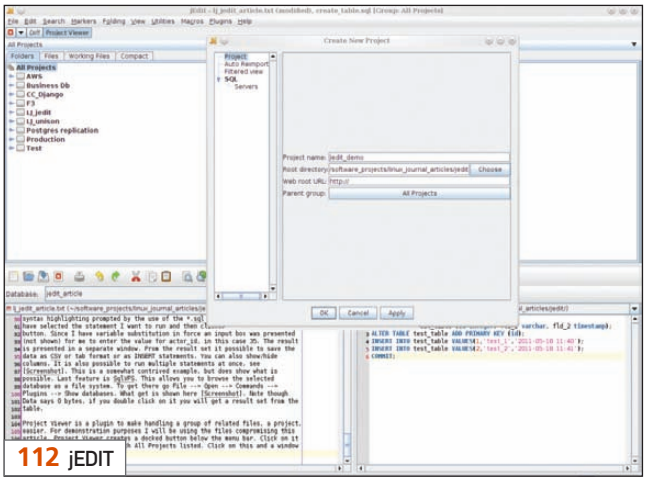
FEATURES



COLUMNS



54 GIADA


112 jEDIT

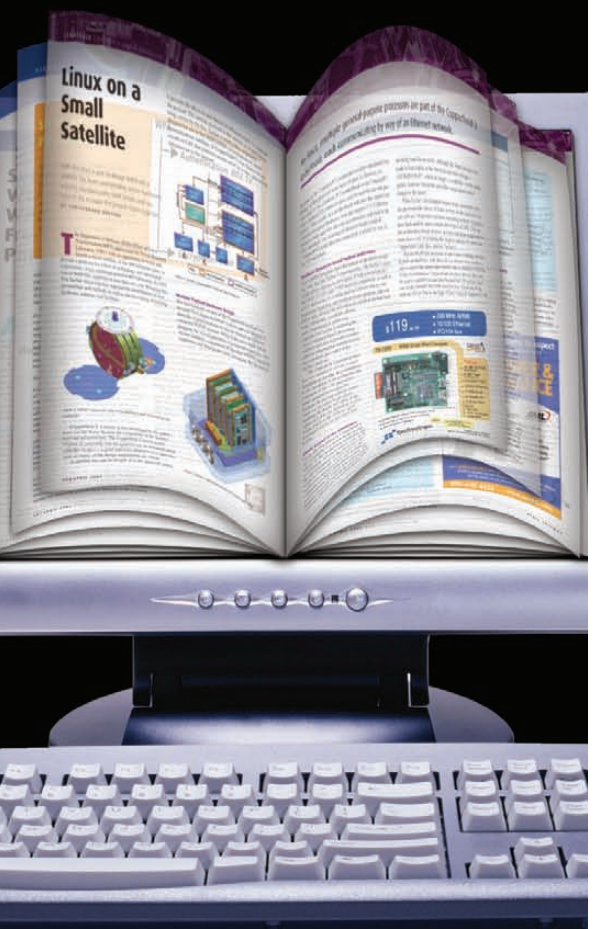
INDEPTH

IN EVERY ISSUE

ON THE COVER

- Make Utility Primer, p. 76
- What's New in GNU Awk 4, p. 94
- WaveMaker for Rapid Application Development, p. 102
- Modern Development with Perl and Moose, p. 124
- Develop GUIs with Qt4 Designer and Eclipse, p. 84
- Multiplatform Development Using GNU Libraries and Tools, p. 60
- Use Gauger for Performance Regression Testing, p. 68
- Getting Started with jEdit, p. 112

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Belltown Media, Inc., 2121 Sage Road, Ste. 310, Houston, TX 77056 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

Proofreader Geri Gale

Publisher	Carlie Fairchild publisher@linuxjournal.com
General Manager	Rebecca Cassity rebecca@linuxjournal.com
Advertising Sales Representative	Joseph Torres ads@linuxjournal.com
Associate Publisher	Mark Irgang mark@linuxjournal.com
Webmistress	Katherine Druckman webmistress@linuxjournal.com
Accountant	Candy Beauchamp acct@linuxjournal.com

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

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

Subscriptions
E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
PHONE: +1 818-487-2089
FAX: +1 818-487-4550
TOLL-FREE: 1-888-66-LINUX
MAIL: PO Box 16476, North Hollywood, CA 91615-9911 USA

They say work smarter, not harder. They must be using our processor.

intel
Sponsors of Tomorrow.™

The next generation of intelligent server processors
The Intel® Xeon® processor 5600 series automatically regulates power consumption to combine industry-leading energy efficiency with intelligent performance that adapts to your workload. Check out the new intelligent features of the Xeon® 5600 at intel.com/itcenter.

intel Xeon inside™

SEE WHAT INTELLIGENCE CAN DO

Servers from iXsystems feature the Intel® Xeon® processor 5600 series.

iXsystems®

Enterprise Servers
for Open Source
www.iXsystems.com
1-855-GREP-4-IX

Intel is not responsible for and has not verified any statements or computer system product-specific claims contained herein.

The **iX2216-10G** features dual on-board Intel® 82599EB 10 Gigabit SFP+ Ports, dual on-board Intel® 82576 Gigabit Ports, and 18 DIMM slots supporting up to 192GB of DDR3 ECC Registered memory. Ideal for HPC, Data Center, Virtualization, Clustering, and Cloud Computing applications.

The **iX1204-10G** features the latest Intel® CPUs based on the 32nm and 45nm next-generation microarchitecture. The next step in intelligent performance, automated energy efficiency, and flexible virtualization.

iX2216-10G

- Dual Intel® Xeon® 5600 Series Processors
- 2U Form Factor with sixteen 2.5" SAS/SATA Hot-Swap Drive Bays
- On-Board Dual Port Intel® 82599EB 10 Gigabit SFP+

iX1204-10G

- Dual Intel® Xeon® 5600 Series Processors
- 1U Form Factor with 4 Hot-Swap SAS/SATA Drive Bays
- On-Board Dual Port Intel® 82599EB 10 Gigabit SFP+

Off the Press

DOC SEARLS

Why all-digital is more liberating than some-digital and some-print.

Ever since I discovered HTML, it's been my preferred format for writing. Every word of mine that's gone into *Linux Journal*, since I started in 1996, has been written and delivered in HTML. That's because my writing has been normalized to hypertext, and to pixels rather than print.

What's different for me this time is that I'm not paying attention to my monthly 900-word limit (or less if images are involved). While a word limit does impose the discipline of brevity, the fact remains that brevity is not the only virtue of good writing. Yes, it's a good one to have when your column appears on the last page of a print magazine. But when that magazine is no longer confined by the dimensions of printed pages, you're free to go longer—or shorter, as the case may be.

My case this month is for the all-digital version of *Linux Journal*. Yes, we lose a lot, but we stand to gain much more. Let me explain.

We've fought to stay in print ever since the dot-com crash nearly killed us, 11 years ago. Before that crash, we were fat with ads from well-funded startups. When the bust hit, many advertisers vanished without a trace, owing us literally $millions we never collected. After that crash, getting and keeping advertisers for a print trade publication was much harder. The costs of printing and mailing also went up, and continued to go up. Meanwhile, Linux succeeded in the marketplace and is now the most widely used operating system.

Yet, while Linux continues to spread, the population of pure-Linux geeks—the kind who subscribe to *Linux Journal*—has remained a core that has grown very little. We continue to serve that core. That's our mission, and we're sticking to it. The question is, what's the best way? Today, it's hard to say print is that best way, especially with more and more people spending more and more time reading glowing rectangles rather than paper.

But, we are by nature and practice a print magazine, and we have done our best to remain one, even as the world has changed around us. So I want to congratulate the publishing side of our house for keeping our print operation going, against stupendous odds, and for never selling out. (And believe me, there were many offers, mostly from entities that are now gone.) Our team did the

impossible for as long as it could.

Yet, consider this. We also always have been a digital publication, starting with the first CD digest of issues in 1994. And, digital publishing has done nothing but grow from the beginning. So has advertising in the digital realm, which is inherently limitless.

Something else also has started to happen in digital publishing. It has become easier, and more acceptable, for people to pay for goods that also are available for free. There has been much experimentation here, and we are among the many doing the experimenting. One advantage for us is that we've always had paying subscribers. Maybe it's crazy to think they'll stick with us after we go all-digital. But, I don't think so. I'm a big believer in the willingness of people to pay for value, provided the means are there. We have some means today, and we will have better ones tomorrow, especially if you help us think those through—while also helping us improve our editorial methods and materials.

Every magazine has a periodical heartbeat. Ours always has been monthly. That won't change. What will change is how much time passes between what we write and when it appears. A production cycle that took several months will now take just weeks. (So for this issue, I am writing this on August 1st for a September publication date.) Much more of our stuff will be current, or as close to *now* as we can get.

We always will remain a print publication at heart (and in that respect, we will be no different from the rest of journalism), but we won't remain contained by the print medium. That medium, where nearly all of our contributors grew up, has legacy values (fairness, transparency, credit to sources and so on) that are important to bring to a vast new world that has too little of them. Again, we expect you to help us with that.

Linux Journal always has been a publication for the Linux Community. *Linux Journal* will now be a publication by the Linux Community as well. This is a very good thing. Here in the digital world, connection between people and ideas are much more direct and immediate. Understandings are also easily iterated and improved. Just like code.

Maybe Linux doesn't need *Linux Journal*—or anything, other than continued constructive hacking in kernel space. But I do believe Linux has been better with *Linux Journal* around than it would have been without it. Therefore, with *Linux Journal* in a much more improve-able place, we can't help but make Linux better in the process.■

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.

SHAWN POWERS

My Language: Dork++

If I've learned nothing else from American politics, it's that it doesn't take knowledge or insight on a topic to have lots to say on the matter. Thankfully, although this issue's Programming focus isn't even close to my area of expertise, our authors don't have that shortcoming. The worst you should have to put up with is me trying to explain what this issue contains. Feel free to point and laugh.

Kyle Rankin, a fellow sysadmin, works through an interesting conundrum this month. You're all familiar with programs like DBAN for wiping sensitive data, but what if you need to delete information securely on a server thousands of miles away? (Or, in the next room if you're lazy like me.) Kyle shows how to go about taking care of a seemingly difficult chicken/egg scenario. Kyle also shares a "Tale from the Server Room" with Bill Childers and talks about the joy of UPS delivery—more specifically, when servers are unboxed, sometimes things don't go quite as planned.

If you're beginning to worry our Programming issue doesn't contain articles about programming, fear not. Yes, we try to include a little something for everyone, but this issue focuses on programming, and we've got tons of useful stuff for you. Nathanael Anderson starts out with an appealing way to learn multiplatform GNU development: getting a guitar synth to work with *Rock Band 3*. Unfortunately, there's no programming that can make me any better at *Rock Band*, but using a real guitar is a step in the right direction!

My friend Adrian Hannah is back this month with a primer on the Make utility. For most users, prepackaged applications are how programs are installed. For programmers, or people on the bleeding edge, it's necessary to compile programs themselves. Adrian shows how to "make" programs from their source code. Sometimes when you are on the bleeding edge, you'll notice that a newer version of an application isn't always better than the previous version. Programmers need to be aware of such things, and Bart Polot and Christian Grothoff show

us Gauger, a tool that monitors performance regression. Sometimes an application is slower because it has more features, but sometimes it's just slower because of an erroneous source code change. Gauger helps determine when new versions go bad.

When I took programming in college, I started out learning to program command-line utilities that did little more than solve the problem presented in the curriculum. If programming was a little more interesting back then, I might have stuck with it for longer than the single semester it was required. My problem was that I wanted to make GUI programs. PJ Radcliffe shows how to develop GUI interfaces with Qt4 Designer and Eclipse. Granted, a fancy GUI controlling a cruddy underlying program isn't very useful, but at least for me, a cool-looking program is something I'd like to spend time perfecting. PJ shows how easy it can be to include GUI controls.

If GUI programs aren't for you, that's fine too. Adrian Klaver explores jEdit, which is a very powerful and cross-platform text editor. jEdit has features that make programming much easier, and its cross-platform nature means you can use a consistent interface regardless of the computer you're stuck using. Arnold Robbins is a fan of text as well, and he presents GNU Awk version 4. Awk has been around forever, and although it's still as useful as it's ever been, version 4 offers a few new tricks as well.

Of course, we have our regular columnists teaching about programming this month as well. Reuven M. Lerner discusses CoffeeScript, a different way to program JavaScript. Dave Taylor finishes his series on determining the day of the week in a script. Plus, we have many other programming-related articles as well! Henry Van Styn describes how to write object-oriented code in Perl, Donald Emmack teaches how to use WaveMaker, and we've even included the results of a LinuxJournal.com programming survey so you can see what your fellow Linux programmers are up to.

If you're a programmer, this issue likely will be one of your favorites of the year. If you're not a programmer, there still are exciting things to read, and you might find that programming is more interesting than you originally thought. I know I learned a lot this month, and I might have to dust off my old C++ course book and figure out how to make a GUI version of "Hello World". Either way, we hope you enjoy reading this issue as much as we enjoyed putting it together. ■

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.

DO IT WITH

DRUPAL

2011

New York City

Oct 12, 13, & 14

Oct. 12, 13, & 14

Techniques & Technologies

for BUILDING SUCCESSFUL WEBSITES

REGISTER TODAY!

doitwithdrupal.com

"Attending Do It With Drupal was absolutely worth it. I learned more about Drupal in a few days than I would have in months on my own. Meeting so many talented and passionate web professionals was exhilarating."

Scott Phillips, Web Developer, Drake University

Lullabot™
powered

★★★ Featuring Awesome Speakers! ★★★

Jeffrey
ZELDMAN

Josh
CLARK

Karen
MCGRANE

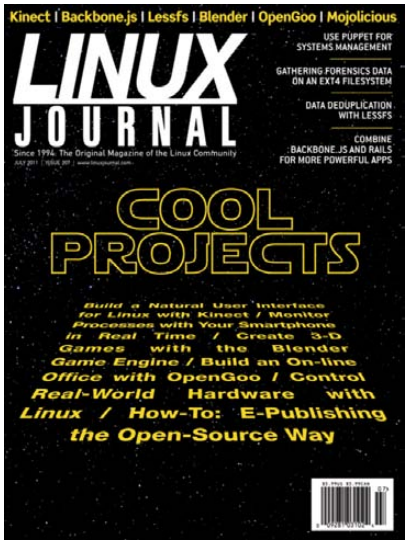
Also Jeff Robbins, Angela Byron, Karen Stevenson, Jeff Eaton, Ryan Szrama & more!

Do it with Drupal sessions are designed for you, covering the latest Drupal information and best practices. But you'll learn about more than just Drupal -- choose from sessions on UX, designing mobile apps, web typography, and more. And the speakers who present are not just knowledgeable, they're fun and engaging. Do it with Drupal provides practical information that you can start using right away!

Use coupon code LJ50 when you register & save $50!

** coupon code expires October 10th*

letters



Aussie Admirer

I had a couple hours to kill at the shops while I was waiting for new tyres on my car and found an April edition of your mag for $13.95 AUD (today's date is June 6). The people at the news agency stated it cost that much because they had to get it from America, and it was two months old because it would cost even more to send via air freight! A quick check of exchange rates shows the Aussie dollar paying $1.07 USD, making the price in USD close enough to $14.95. Would you sell many copies for this price, I wonder? It's a crying shame that Australian retailers charge this absolutely stupid markup. The reasoning I was given is utter garbage—a hundred issues and I could fly them on a seat first-class with free champagne and still show a profit!

Enough with the whining. I purchased the mag anyway and had a very interesting read while I waited. Compliments to your team. This won't be my last purchase; however, I likely won't be purchasing retail again any time soon.

Word of mouth spread of Linux in this country can get only so far, and the lack of reasonably priced journals on the shelves of our shops sparking the interests of new users is quite an impediment to our plans of world domination.
—Scott K.

Thank you for picking up an issue, even if the price was painfully high! The cost of international shipping and printing is one of the reasons we switched (starting with this issue!) to a digital-only format. For the same subscription price as anyone here in the US, you get the same experience, regardless of your location. We think this recent change will really level the playing field for international subscribers. Hopefully you agree!—Ed.

Magazine Locations

I'm sitting in a hospital room watching my father recover from liver cancer surgery and a fall on the way home. I recently discovered that my subscription expired, and I'm going to renew it next week. In the meantime, I thought I'd pick up a copy locally, and this is why I'm contacting you. I'd like you to consider an application for your Web site that would display nearby magazine/bookstore locations that carry your fine magazine. Perhaps

an Android app would be nice too. Yes, I can read the on-line version, but I'd really prefer a hard copy. Perhaps Shawn Powers could assign this little Google smashup to someone? Thank you for your consideration.

—Michael Soibelman

As someone who lives in an area with no local retailers stocking Linux Journal*, I feel your pain. I'm not sure how to create an app like that myself, but hopefully, your letter will spark someone's interest in doing such a thing. As far as assigning it goes, I always could pick you if you like. Hope your dad is doing well.—Ed.*

Google Maps

I enjoyed the mapping article by Mike Diehl in the April 2011 issue ("Find Yourself with the Google Maps API"). Like Mike, I would not be without Google Maps. You may not be interested in the content (unless you are a train buff), but take a look at these sites. I think they are truly awesome and are done by an "amateur" at that. It just goes to show what skills are out there, and it makes me very envious: **traintimes.org.uk/map** and **traintimes.org.uk/map/tube**.

—Roy Read

What Day Is It?

I'm sending a little feedback to Dave Taylor's "parsing the cal" output (see Dave's column in the June–September 2011 issues). There's no need to use regular expressions in the awk script at all, because you can compare numbers directly. Below, you'll find the script you can call by the following command line:

```
$ cal | awk -f day.awk 25

# day.awk
BEGIN{
 ARGC=1;
 getline;getline;
 for(i=1;i<=NF;i++) wd[i]=$i
}
{
 for(i=1;i<=NF;i++)
   if($i==ARGV[1])
     print wd[i+($0~/^ /?(7-NF):0)]
}
```

In the `BEGIN` block, `ARGC=1` prevents it from taking the last argument (25 in this case) as an input file. Then, the script fetches the first two lines and stores the weekday names in an array.

The rest of the script compares the argument number with every field in every line. On a match, the day name is output, `wd[i]`. The month does not always start on a Sunday, so the script has to fix the index for lines starting with a space (condition `$0~/^ /`). For those lines, the first item starts with `index 7-NF`. Note that this fix also works fine for the second line of numbers (which also starts with a space), since 7-7 equals 0.

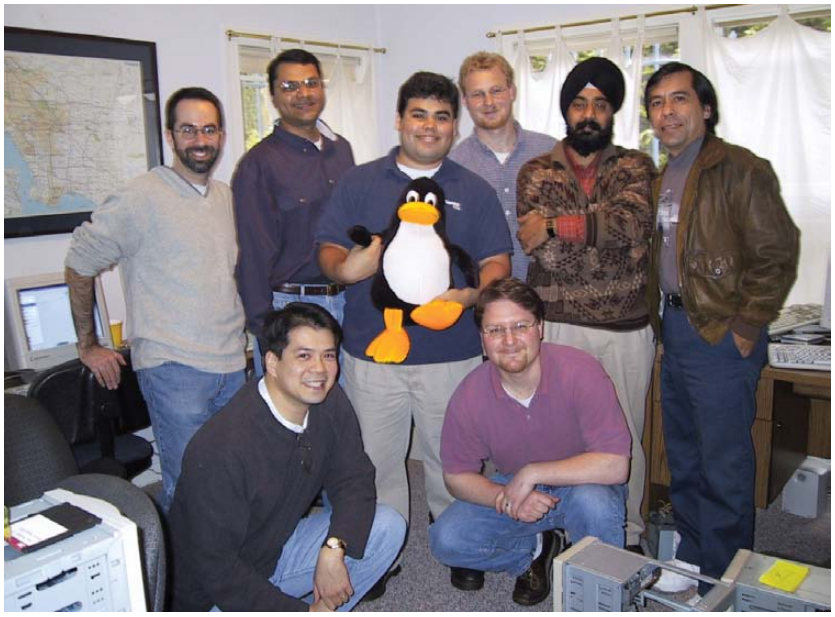
You can make the script a one-liner if you like. It was written in multiple lines for readability reasons. And, last but not least: great OS, great magazine, keep going.
—Eric Miller

Dave Taylor replies: *Thanks for your note. I realized that there was a way to break down the input and process it with a multiline awk script (just as I could do much of the task more easily in Perl or, for that matter, a short C program), but my goal with the Work the Shell column is to force myself to stick with standard Linux shell commands and capabilities as much as possible and see what I can accomplish. Sometimes the result is a bit, um, Byzantine and unquestionably inefficient, but the upside is that it's always interesting and, I hope, informative and entertaining reading.*

Installfest 2001

Installfest goal: adapt seven older desktop computers for use by fourth-grade teacher Mike Steins at Shenandoah St. Elementary School and learn how to install/configure Linux. (We had two Linux experts in the group.)

Outcome: we got four computers working with Linux by scavenging parts from various other machines.



Linux installfest at 419 N Vista St., Los Angeles (circa 2001). Left to right, top row: Danny Olster, Abhijeet Chavan, Christian Peralta Madera, Mike Steins, Charanjeet Singh, Christian's Dad. Bottom row: Chun Wong, Chris Steins.

Although the Linux installfest event ten years ago may not have been hugely productive (eight people @ six hours = four working Linux computers), it got the ball rolling to get computers at the school. Since then, every teacher has received a laptop, projector and document camera. The school has multiple interactive whiteboards, a fully functioning computer lab (actually two—one is made up of aging computers), digital microscopes, cameras and video cameras, a completely wireless network with networked printing and storage capabilities, an in-house server, student e-mail accounts, and we're slowly looking to integrate tablet devices during the next few years as the technology becomes more inexpensive and funding levels rise (if that ever happens). So, Linux (I believe it was the Red Hat distribution) laid the red carpet for technology at our school. Thanks.

—Mike Steins

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

diff -u

WHAT'S NEW IN KERNEL DEVELOPMENT

Google's Mike Waychison has posted some patches to organize Google-specific firmware used in its servers. These patches were met with approval by **Alan Cox** and **Greg Kroah-Hartman**, although Google's servers remain unavailable for purchase by the public. Perhaps that will change soon.

Chris Wright officially is no longer on the stable kernel team, and he has not been involved in that project for some time. Greg Kroah-Hartman will continue his effort to release several short-term stable versions of each kernel, during the intervals between official releases from **Linus Torvalds**. Long-term stable trees in the 2.6 series are maintained by **Andi Kleen**, **Willy Tarreau**, **Paul Gortmaker** and others.

The ancient versioning system, **CVS**, still is being used by some kernel developers in spite of the advent of **git**, and **Sebastian Andrzej Siewior** recently posted a patch supporting those developers. His patch took out all the stale CVS Id tags sprinkled throughout the kernel that were confusing CVS. Apparently, some developers are syncing the kernel sources from git and then feeding the whole tree into a CVS repository before working on it—bizarre. But, it's a great testament to the hardiness of CVS after so many years and so many attempts to find something better.

A recent bug in **SysFS** allowed regular users to overwrite **NVRAM**. **Vasiliy Kulikov**'s patch to close the security hole had taken more than a month to be incorporated into the kernel. In light of that, he's posted a patch to give the system administrator the power to mount the SysFS directory as read-only. This blanket protection would not be in effect at all times, but it would give administrators the ability to lock down that part of the system in the event that a similar bug were discovered. The problem, as Greg Kroah-Hartman points out, is that locking down SysFS may produce other unanticipated problems, and he feels the right approach simply would be to fix SysFS bugs as they occur, rather than add a blanket layer of security over it. The debate is unresolved at the moment and could play out either way.

Huang Ying recently posted some code to cause unknown **non-masking interrupts** (NMIs) to crash the kernel and produce a panic report. But, part of the problem is that a wide array of

systems produce unknown NMIs for no reason at all. Huang's solution was to create a whitelist of systems that were known not to do this, and his patch would work only on that whitelisted set of systems. But, **Ingo Molnar** suggested using **active event filters** to allow unknown NMIs to go through a localized policy decision-making process first, so the decision to panic the system could be made on a per-system basis.

Active event filters, as Ingo pointed out, would allow a certain portion of the decision-making process to occur while still in kernel space, without having to return to userland. This is key, because when the system is crashing, it often is not feasible to pass control over to a user-space program. But in the case where the active event filters determined that a crash probably was not occurring, they could hand control to a user-space dæmon that would make additional decisions about how to handle the unknown NMI.

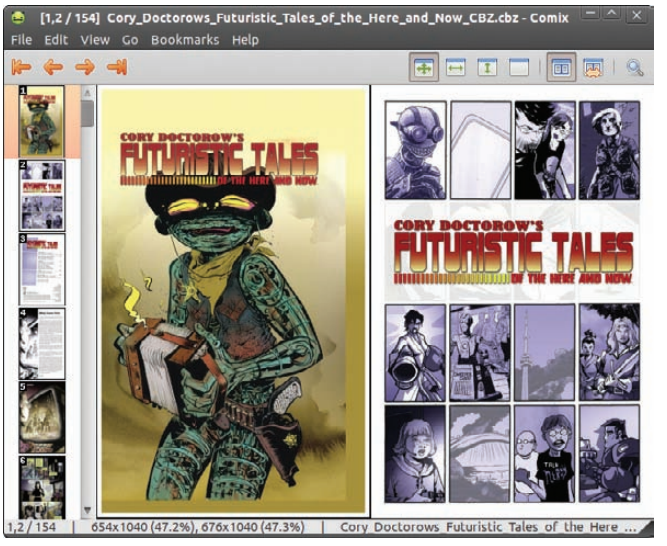
Active event filters apparently are tremendously powerful and soon may be seen in use throughout the kernel as a way of standardizing a number of disparate behaviors that currently are handled in an ad hoc manner.

—ZACK BROWN

CBZ, the MP3 of Comics

Digital music and, more recently, digital video and digital books, have changed the way we consume media. Comic books are no different, and with the advent of tablet computers, digital comics are becoming more and more popular. If you don't have a tablet computer, however, viewing CBR (or their compressed version, CBZ) files is as simple as installing a CBR reader and downloading your favorite comic.

Many comic book readers are available for Linux. A quick Google search will turn up programs like Comix, ComicMaster and Comical, all of which display digital comics quite well. Another search likely

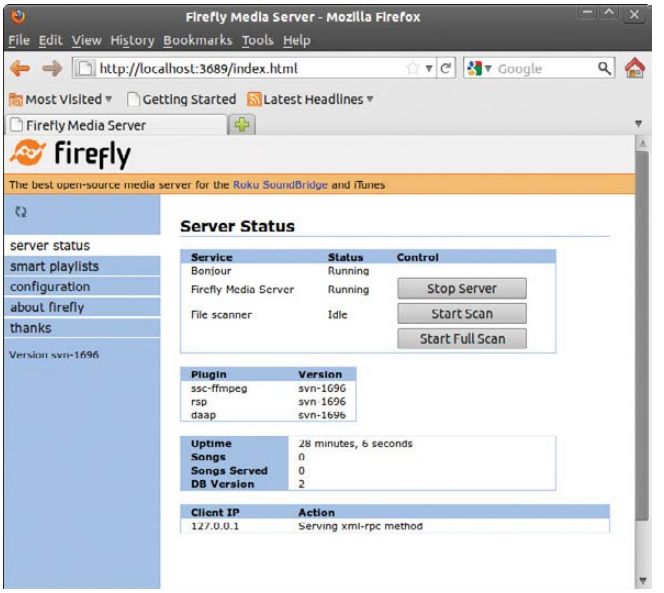


will turn up some free comic resources, like the one shown here: Cory Doctorow's *Futuristic Takes of the Here and Now*. If you miss the comic books of your youth, or if you still enjoy them on a regular basis, you owe it to yourself to check out CBR/CBZ files.**—SHAWN POWERS**

ROCK YOUR WORLD WITH FIREFLY

No, I'm not talking browncoats and spaceships. Unfortunately, that ship has sailed. If you're the musical type, however, installing a Firefly Media Server is fairly simple. It was renamed from mt-daapd, so your distribution still might call it that. After a quick install, visit the Web configuration, usually at http://localhost:3689 with the default login mt-daapd and password mt-daapd. You can configure your music location, smart playlists and just about every other aspect of the media server. Then comes the cool part.

On any software or hardware on your network that supports daap (often known as the iTunes protocol), you should be able to play your music remotely. Firefly does a decent job of scanning your music collection and updating the clients on the network.



You can add m3u playlists, and Firefly will serve up playlists as well.

I find the best way to listen to music on XBMC is via daap. It makes configuring playlists and adding media simple. It's also cross-platform, so those folks using actual iTunes can listen to their tunes as well.

—SHAWN POWERS

Non-Linux FOSS

Sometimes, you just want a simple word processor. Yes, there are many options for word processing, from the awesome OpenOffice.org to the awesome-for-other-reasons vi. If you're looking for a happy medium, however, it's hard to do better than AbiWord. When you add the free on-line component, AbiCollab.net, you even can collaborate with other AbiWord users on documents.

AbiWord is available for Linux, Windows and even OS X. You need version 2.8 or higher to use AbiCollab.net, but most distributions include at least version 2.8. Check out the Web site at **www.abisource.com**.—SHAWN POWERS

They Said It

"The question of whether a computer can think is no more interesting than the question of whether a submarine can swim."—E. W. Dijkstra

---

"One thing a computer can do that most humans can't is be sealed up in a cardboard box and sit in a warehouse."—Jack Handey (from "Deep Thoughts", *Saturday Night Live*)

"If you have any trouble sounding condescending, find a UNIX user to show you how it's done."—Scott Adams

---

"Isn't it interesting that the same people who laugh at science fiction listen to weather forecasts and economists?"
—Kelvin Throop III

"Computer Science is no more about computers than astronomy is about telescopes."—E. W. Dijkstra

---

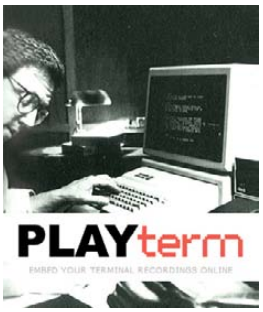
"Even he, to whom most things that most people would think were pretty smart were pretty dumb, thought it was pretty smart."—Douglas Adams

---

"It is bad luck to be superstitious."
—Andrew W. Mathis

"The problem with quotes on the Internet is that it is often difficult to verify their authenticity."—Abraham Lincoln

Playterm, Platform of the Gurus



Did you learn all your Linux console skills from books or from forums? Or, did you peek over someone's shoulder to see the real action? Once in a while, we stumble upon new projects that deserve some attention, like Playterm (**www.playterm.org**). What's the reason for this command-line "peep show"? To spread GNU Linux command-line knowledge.

You will see a fair amount of on-line terminal recordings when you enter this site. The recordings cover several topics performed in the shell: tricks, one-liners, guided tutorials and handy utilities.

Personally, I found them quite entertaining to watch, and it brought me back to the BBS days. It can be educational, and also quite hilarious to see people making typos and mistakes.

Another interesting Playterm feature is the embed facility. You can upload terminal recordings on this site, which you then can embed and play on your blog or Web site. Optionally, you can allow commenting on your recordings, which, of course, will provide interesting hints and tips and other improvements.

The Coder of Salvation (Leon van Kammen) created Playterm because he was just too curious about what people were doing in their terminals. He used to work for a company where he did extreme programming sessions with his colleagues through the GNU `screen -x` utility. In his experience, it is extremely educational when you work together in one terminal (and also entertaining). In his opinion, console-related books and articles are great, but sometimes it can be more helpful to see gurus at work. If it were up to him, more command-line projects should feature a terminal player on their sites: "Why not? Why have only a tar archive on a site? Developers should make more demos to show the world how cool their utilities are! It hurts me to see so many great utilities being overseen by the masses." Obviously, these are the words of a true terminal evangelist.

Before the big Internet boom, people used BBSes a lot (**en.wikipedia.org/wiki/Bulletin_board_system**). People called to other people's BBSes via their phone line. The cool thing about running your own BBS was that you could create an console "intervention". By doing this, you could "take over" the terminal session of a given user. In those days, a lot of teaching and

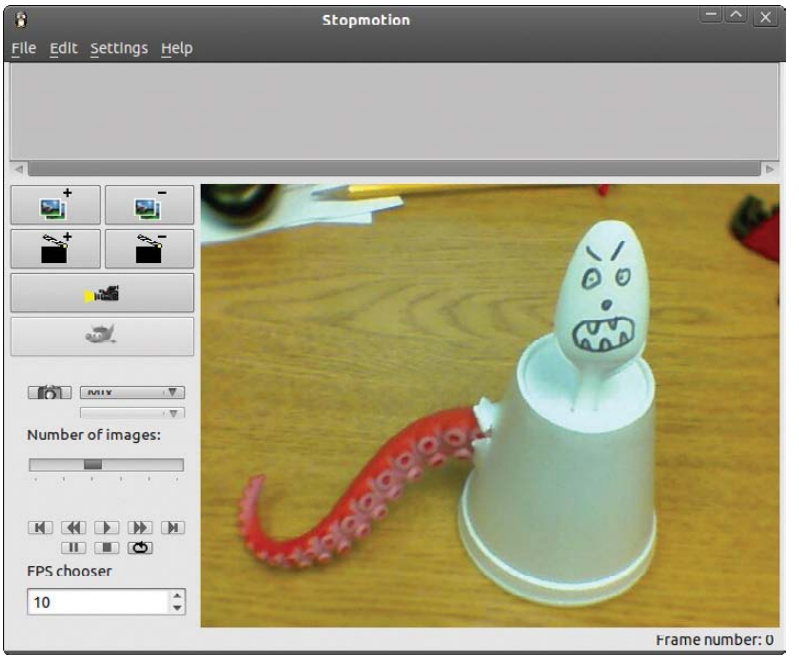
cooperation was done this way.

Of course, the Playterm Web site would not be possible without the GNU and Open Source movement. Thanks also to the developers of ttyrec (**0xcc.net/ttyrec**) and jsttyplay (**encryptio.com**).

Playterm is still beta, but it's already fully functional. Currently, we are curious about how many users we can serve, but in terms of global Linux knowledge, we are very excited. At this point, Playterm.org is a nonprofit project to serve the community and spread GNU Linux knowledge. Hopefully, it will inspire youngsters to use the shell more often.—**LEON VAN KAMMEN**

Roll Your Own Cthulhu Flick

I may not be Steven Spielberg, but every time I see a rerun of *Gumby*, I'm convinced I could be a famous producer. With Linux, I don't even have to get a fancy movie set. I can make my own science-fiction adventure film with nothing more than a Webcam and a streak of bizarre creativity.



Stopmotion is a Linux program designed for creating stop-motion films. It's available for most distributions and easily compilable for the rest. Stopmotion is simple in its design, and it allows you either to import a series of pre-taken photos or take live stop-action with a Webcam. I find the latter to be slightly easier, as you can see a ghost image of the last shot you took, making the slight changes you need easy to spot.

Recording stop-motion films is tedious work, but the end result can be pretty cool. Check out the homepage, short-linked here: **is.gd/stopmotion**. If you create an interesting video, send a link to ljeditor@linuxjournal.com. If we get enough, we'll post them on our Web site.—**SHAWN POWERS**

Computer-Aided Engineering in Linux

Engineers are some of the heaviest number-crunchers around. If you are a grad student, post doc or undergrad, you usually get whatever is lying around as your work machine. Also, depending on how inflexible your local IT department is, you may be forced to use one of the commercial operating systems around these days. What are lowly students to do when they need to do heavy computational work? You may be interested in looking at CAELinux (Computer Assisted Engineering, **www.caelinux.com**). This project provides a live CD that gives you all the open-source tools you might need for your engineering work. And, because it is a live CD, you can use it without touching the local drive of the machine you are using.

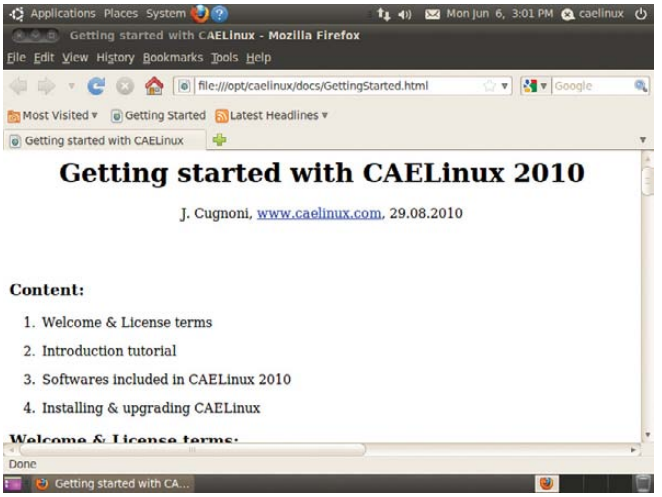
Like all live CDs, it has all the standard Linux desktop tools you should be familiar with, including Firefox for Web browsing, Evolution for e-mail, and OpenOffice.org for word processing, spreadsheets and presentations. Along with these applications, there are dozens of others to help with all your number-crunching work. The most recent versions are based on Ubuntu, so it should be a fairly comfortable environment for most people. Be aware, however, that you can't use the usual software update mechanism in Ubuntu. Many of the packages in CAELinux are compiled from source and optimized, so you don't want them being overwritten accidentally by any packages provided by Ubuntu.

Welcome to CAELinux

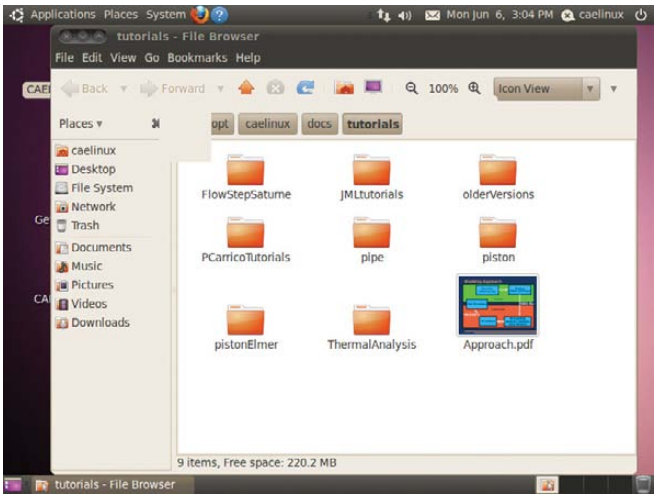
A really well written introduction to CAELinux is available right on the desktop, called "Getting Started". You should start here if this is your first step into the world of CAELinux.

Last month, I looked at OpenFOAM in this space. CAELinux includes a full install of OpenFOAM. It also includes another fluid dynamics program called SALOME. This program provides a full graphical interface that takes you from forming your problem, to modeling, to calculation and through to analyzing your results. This might be a good choice for those who are more comfortable with a GUI. A series of examples on the desktop are available that provide a walk-through of the program, showing each of the steps as you go through.

As you can see, the tutorials walk through several common simulations, like modeling flow through a pipe.



Getting Started



CAELinux Tutorials

These can provide great starting points for many people.

If your work leans more toward data analysis, several popular packages are available. For all of you Matlab addicts, there is Scilab. Scilab provides the same types of functions in an environment familiar to Matlab users. There also is Maxima, which provides tools more from a mathematical background (for example, analyzing functions and doing calculus), as compared to Scilab's approach of working from a matrix background (such as looking at data analysis). Maxima has several front ends available. The default one in CAELinux is wxMaxima. If you are doing really heavy statistical analysis, there is R. The real power of R is the CRAN repository, and a fair amount is available out of the box. R also has several graphical front ends. CAELinux provides two: R Commander and RKWard. If you are doing work more along the lines of pure mathematical analysis, there also is Octave. The default GUI available within CAELinux is QtOctave. In all of these cases, text-based interfaces also are available, if you are an old-style computer user who prefers

that kind of thing.

Several software packages exist for applications other than CFD or statistics. If you need to do finite element analysis, there's elmer. It provides both a text-based and GUI interface. There also is JavaFoil, available for doing analysis on airfoils and wings. If you are designing electrical circuits, two packages are available. Electric is a CAD program that helps you lay out a circuit. And, once it is all laid out, you can use PCB Designer to get it set up so you can etch a board to make it real.

This is all fine and good if you can use a standard toolset in your work. But, what if you need computing power for really cutting-edge research? CAELinux provides the entire GNU toolset. This means you have everything you need to go ahead and start developing your own code. All of the most common scientific and engineering libraries, like gsl and LAPACK, are available. If you are working on really large problems, MPI and openMP also are available. This way, you can develop a parallel programming solution if that is what your problem needs.

Once you have finished all your calculations, an important part of data analysis is graphical analysis. There is something visceral and instinctive about actually seeing your data represented. To this end, CAELinux provides several packages. If you simply want to plot your data, you can use programs like grace and LabPlot. If you want to do more complicated data analysis, you have programs like G3Data and OpenDX Data Explorer. These programs have lots of functionality that can be used to look at your data graphically. If you are doing CFD work, several programs for visualizing your meshes are available. So, you have your choice based on what features you need.

The last option to look at this month is using CAELinux in "the cloud". Cloud computing is one of those sexy terms that gets used a lot in marketing, but it sometimes doesn't really give you anything useful. In this case, there really is something substantial being offered. CAELinux now can be run as an application under Amazon Elastic Cloud Computing. You can now run, on demand, as many nodes as you like, each having eight cores and 64GB of RAM. For people who don't have the resources to run their own clusters, but need more than what a desktop can handle, this can be a very attractive choice. It definitely is worth looking into as a possible option. You can find more information about EC2 at **aws.amazon.com/ec2**, and the CAELinux Web site has a very good set of instructions to get you up and running.

As you can see, CAELinux provides a lot of power and functionality for doing computational science. Because it is a live CD, you can run it on essentially any 64-bit machine without touching the hard drive. But, you also have the option of installing it on the machine if you are allowed. Download an ISO and start playing with it to see just how much work you can do with it.—**JOEY BERNARD**

More TFLOPS, Fewer WATTS

Microway delivers the fastest and greenest floating point throughput in history

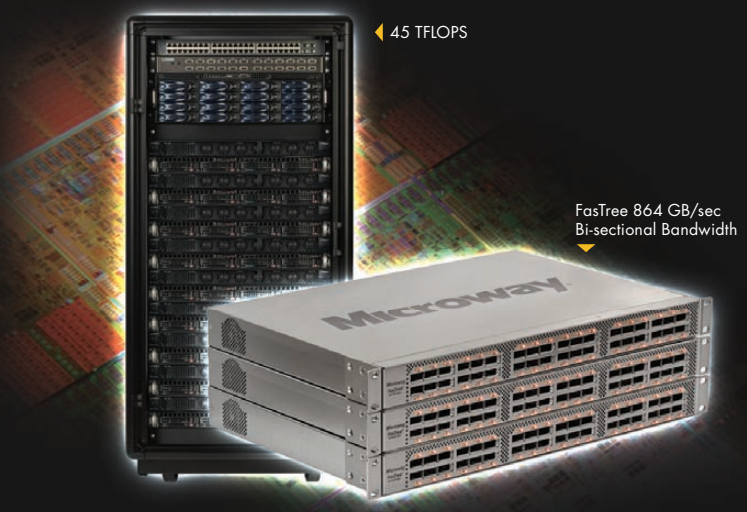
Enhanced GPU Computing with Tesla Fermi

- 480 Core NVIDIA® Tesla™ Fermi GPUs deliver 1.2 TFLOP single precision & 600 GFLOP double precision performance!
- New Tesla C2050 adds 3GB ECC protected memory
- New Tesla C2070 adds 6GB ECC protected memory
- Tesla Pre-Configured Clusters with S2070 4 GPU servers
- WhisperStation - PSC with up to 4 Fermi GPUs
- OctoPuter™ with up to 8 Fermi GPUs and 144GB memory

New Processors

- 12 Core AMD Opterons with quad channel DDR3 memory
- 8 Core Intel Xeons with quad channel DDR3 memory
- Superior bandwidth with faster, wider CPU memory busses
- Increased efficiency for memory-bound floating point algorithms

Configure your next Cluster today!
www.microway.com/quickquote
508-746-7341



FasTree™ QDR InfiniBand Switches and HCAs

- 36 Port, 40 Gb/s, Low Cost Fabrics
- Compact, Scalable, Modular Architecture
- Ideal for Building Expandable Clusters and Fabrics
- MPI Link-Checker™ and InfiniScope™ Network Diagnostics

Achieve the Optimal Fabric Design for your Specific MPI Application with ProSim™ Fabric Simulator

Now you can observe the real time communication coherency of your algorithms. Use this information to evaluate whether your codes have the potential to suffer from congestion. Feeding observed data into our IB fabric queuing-theory simulator lets you examine latency and bi-sectional bandwidth tradeoffs in fabric topologies.

NVIDIA Tesla CUDA | Intel Channel Partner Premier 2010 | AMD Fusion Partner Premier | GSA

GSA Schedule Contract Number: GS-35F-0431N

Microway®
Technology you can count on℠

LinuxJournal.com Programming Survey

One of our favorite things to do over at LinuxJournal.com is to check the pulse of the Linux community and our readership. We do this fairly regularly with simple polls on our site. These give us valuable insight into your interests, and they give us a fun way to get feedback on a specific question. Sometimes a simple question generates a tremendous amount of discussion, and even a little controversy. We recently asked readers to choose their favorite programming language, and we received a lot of great answers. Check it out at **www.linuxjournal.com/content/whats-your-favorite-programming-language**.

We know that for some of you, your favorite programming language is such an important part of your existence that you are understandably quite opinionated on the subject. So, as you might expect, there were clear leaders and underdogs, as well as passionate supporters of each.

Python tends to be the favorite among our readers, and it has won more than one Readers' Choice Award for favorite programming language. Indeed, Python has staying power, as it once again was the leading choice among LinuxJournal.com readers with 24% of the votes. Not surprisingly, there were many skeptics among the commenters,

but Python fans did their best to set everyone straight about Python's virtues, and one reader offered this sage advice:

> In my opinion, a large Python system is well organized (like any language), sticks to standards, has docs (particularly module docs) and has tests. Consistent, well-written Python code makes it fairly obvious what objects/types functions accept and return, and having useful standard types (lists, dics, sets, etc.) encourages people not to make exotic variations often. Having module docs that document what your function does and its inputs and outputs clarify any questions.

In a close second and third place were C and C++, respectively. Both had enthusiastic support in the comments section, and we can infer from some comments that "all of the above" (in the case of C, C++ and Python) might have been a popular answer as well. Many indicated a preference for different languages for different tasks, and we applaud their versatility and open-mindedness!

Java trailed C++ in fourth place, and it was cited more than once as a preferred learning and teaching language. It probably also is safe to bet that some of Java's popularity is due to the increasing

demand for Android applications written in Java. One anonymous commenter gave us a detailed breakdown of how Java fits the bill versus other languages:

- For fun: Forth.
- For teaching (elementary): Logo.
- For teaching (secondary+): Java.
- For desktop applications: C++ (with Qt).
- For Web services: Java (SE or SE + Servlet only).
- For enterprise internal: Java (EE with or without Spring).
- For enterprise external: Java (SE or SE + Servlet and JSP, no Spring).
- For cloud: C++, Java, Scala and Python together.
- For mobile: Java (Android) C++ and Qt (native).
- For embedded systems: C++ or Java (real time).
- For device drivers: C++ or C.
- For deep embedded (no MMU): C, Forth or Assembly, as needed.

Although these may not suit everyone, I have to give credit for such a detailed response.

Getting a little less love were Perl, C# and Ruby. Although these all have devoted followings, their fans were largely outnumbered in this poll. I was a little surprised to see Ruby score only 4% of the votes, as I personally know so many enthusiastic Ruby coders.

Haskell and OCaml each got 1% of the vote, while the catchall "other" made up the remaining 8%. Most interesting were the comments describing the variety of languages our readers use regularly as well as dabble in.

PHP always has a few fans, and although we can argue that PHP belongs in a separate "scripting languages" poll, its fans still showed support. The same can be said for JavaScript and Bash, both of which got a little love from our readers. Perhaps we'll do a favorite scripting language poll soon, but in the meantime, since PHP, JavaScript and Bash tend to be part of my daily life, it is nice to see I am not alone.

There were quite a few mentions of Scala, LISP and Erlang, as well as oldies but goodies, FORTRAN and Cobol. The latter occasionally were mentioned in the context of "getting old", but frankly, do the classics ever go out of style?

It is always fun to read the nostalgia posts that inevitably appear on these sorts of comment threads. When programmers share their stories, there is usually a mention of the language they started with,

and when our readers share stories of the language they were using in 1976 or even 1968, it gives us all a glimpse at where we've been, and how we all got to where we are today. Whether you started as a mainframe pioneer or a geeky kid typing out rudimentary BASIC on your TI-99/4A (What? Bill Cosby said it was cool!), you've likely had a somewhat meandering journey made up of more than a few languages to get to the code you write today. In my humble opinion, sharing these stories is one of the best parts of LinuxJournal.com. I hope you'll all join in the fun and check out the current poll next time you visit LinuxJournal.com. Don't be shy! Tell us your stories and opinions in the comments. You never know who you may inspire. I suspect it might be me!—**KATHERINE DRUCKMAN**

Programming Language Survey Results

C	**19% (1,661 votes)**
C++	**17% (1,488 votes)**
C#	**5% (399 votes)**
Haskell	**1% (126 votes)**
Java	**13% (1,118 votes)**
OCaml	**1% (47 votes)**
Perl	**8% (674 votes)**
Python	**24% (2,025 votes)**
Ruby	**4% (361 votes)**
Other	**8% (670 votes)**
TOTAL VOTES	**8,569**

REUVEN M. LERNER

CoffeeScript and jQuery

CoffeeScript is a better way to write JavaScript, but it integrates just fine with libraries like jQuery.

Last month, I wrote about CoffeeScript, a new, small language that compiles into JavaScript. CoffeeScript has generated a great deal of buzz and excitement among Web developers, including no less than Brendan Eich, Mozilla's CTO and the inventor of JavaScript. Ruby on Rails 3.1, which presumably will be released by the time this column sees print, includes CoffeeScript, and other frameworks might follow suit in the future.

Even if you're not interested in the future of JavaScript or in Ruby on Rails, you owe it to yourself to look at CoffeeScript. First, it's a new and interesting language, and I'm definitely a believer in learning new languages as part of my professional development. Second, CoffeeScript's syntax makes it easier to do many things that previously were difficult, long-winded or just plain ugly in JavaScript. Just as a number of languages have emerged that compile to the JVM, but that are easier to use in various ways, so too is CoffeeScript functionality equivalent at the end of the day to JavaScript, but with an easier syntax that's more appropriate for many modern applications.

But, perhaps the most interesting part of CoffeeScript is the fact that, ultimately, it's just another way of writing JavaScript, which means anything you can do in JavaScript, you also can do in CoffeeScript. CoffeeScript programs can run on the server, in such environments as node.js, but they also can run in the browser, working in conjunction with Web applications. Things become even more interesting if you use a JavaScript framework, such as jQuery, for developing Web applications—you can benefit from the best of both worlds, enjoying the power and expressiveness of jQuery, along with the terse and readable syntax of CoffeeScript.

This month, I describe some ways that CoffeeScript and jQuery can interact in a browser-based program.

Even if you don't decide to adopt CoffeeScript in your own programs, it's worth playing with the language to get the hang of things.

Starting Off

I'm going to assume you already have installed CoffeeScript, as well as any support files, such as a mode for your editor. Create a bare-bones HTML file, as shown in Listing 1, and a stylesheet (coffeescript.css), in the same directory, similar to what's shown in Listing 2. Notice how in the HTML file, I include two JavaScript files:

```
<script
src="http://ajax.googleapis.com/ajax/libs/
➥jquery/1.4.2/jquery.min.js"></script>
<script src="app.js"></script>
```

The first probably is recognizable as the Google-hosted version of a minified version of jQuery. But the second file, app.js, is the target of the CoffeeScript compilation—that is, you're not going to write app.js directly. Rather, you're going to write CoffeeScript that compiles into JavaScript.

You do this by creating (in the same directory as the HTML file, unless you want to change the paths in the <script> tag) a CoffeeScript program, named app.coffee. Just to test things, I created a very simple CoffeeScript program that uses the standard (and annoying!) alert dialog to say "hello":

```
alert "hello"
```

Save this file as app.coffee. On the command line, you then want to tell CoffeeScript to compile app.coffee into app.js. (Otherwise, it'll try to execute your program, which not only will mean that the resulting JavaScript isn't available

Listing 1. coffeescript.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="content-type"
 ➥content="text/html; charset=utf-8" />
<link rel="stylesheet" type="text/css"
 ➥href="coffeescript.css" />
<title>CoffeeScript</title>
<script src="http://ajax.googleapis.com/
➥ajax/libs/jquery/1.4.2/
➥jquery.min.js"></script>
<script src="app.js"></script>

</head>
<body>
<h1>Headline</h1>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
<p>Paragraph 3</p>
</body>
</html>
```

Listing 2. coffeescript.css

```
p.large {
    font-size: 30px;
}

p.medium {
    font-size: 20px;
}

p.small {
    font-size: 10px;
}
```

for your Web page, but it also will result in an error if you try to access the DOM, which isn't available outside a browser context.) You can do this with:

```
coffee --compile app.coffee
```

The problem with this approach is that you need to recompile your CoffeeScript program every time you change it. A better solution probably is to tell CoffeeScript to watch the file and compile it every time a change is detected:

```
coffee --compile --watch app.coffee
```

Just after running this, the compiler will run over app.coffee, producing app.js. When you load your Web page, app.js will run, and you should have an alert saying "hello".

Functions and Objects

It's important to remember that although CoffeeScript certainly is a different syntax from JavaScript, it is fundamentally the same language. This means any function or object that you can access from JavaScript can be accessed from CoffeeScript, with the same name. True, CoffeeScript does offer some shortcuts and syntactic sugar; those basic JavaScript objects are still around and kicking. That's why you could invoke the "alert" function in app.coffee—it's not that CoffeeScript has defined a new function, but rather that you're using the same built-in JavaScript function.

This means if you load jQuery in the same document as a program written in CoffeeScript, you can use jQuery from within CoffeeScript. What does that mean? Well, it means you can access the jQuery object directly, often abbreviated as $. For example, let's change app.coffee so that it tells you what version of jQuery you're using, normally available via `$().jquery`. You also can do this in CoffeeScript:

```
alert $().jquery
```

Let's do something a bit more exciting now, using jQuery's capabilities for easily changing elements in the DOM based on events that take place. For example, you can add the "large" class to all of the paragraph elements in your document.

In JavaScript, you would do this with:

```
$("p").addClass("large");
```

In CoffeeScript, you can use the same code as above. But, because CoffeeScript allows you (like in Ruby) to remove most of the parentheses, you end up with this:

```
$("p").addClass "large"
```

Notice how the original jQuery selector has remained the same, as has the method you're calling on each of the selected DOM elements. What has changed is the way you invoke the method; you no longer need to put parentheses around it.

There is a problem with this though. It will execute immediately upon being loaded. The problem is that just because the JavaScript is executing, it doesn't mean the HTML all has been loaded or rendered onto the screen. Now, you can get around this in traditional jQuery by putting all of your code inside a call to `$(document).ready()`, as in:

```
$(document).ready(
    function () {

    // Event handlers go here

    }
);
```

You can do the same thing, but in less space (of course) using CoffeeScript:

```
$(document).ready ->
  ($ "p").addClass "large"
```

As you can see, CoffeeScript's syntax is cleaner and trimmer, without nearly as many curly braces and parentheses. You start off with the same invocation of $ with the "document" parameter, and then invoke the "ready" method on that object. You then need to pass a function to "ready", which you do by defining a new, anonymous method with CoffeeScript's -> symbol, cleverly dubbed "dashrocket" in the PeepCode screencast about CoffeeScript.

In other words, you've wrapped your original invocation of "addClass" and friends inside a function that's invoked when the document is ready. But, you've cut the number of lines of code in half, without sacrificing readability. Now, let's do something a bit more exciting, namely change the size each time you click on a paragraph. In order to do that, you'll need to use one of jQuery's event handlers—specifically, you'll use the "click" handler, which you set by invoking a selector, the "click" method, and then passing the name of a function. For example, if all you want to do is display an alert dialog when

a paragraph is clicked, you can do it with the following CoffeeScript:

```
$(document).ready ->

  changeSize = ->
    alert("changing size!")

  $("p").addClass "large"

  $("p").click changeSize
```

Note how I've defined two functions here: an anonymous function for `$(document).ready` and another function to which I give the name `changeSize`. But, of course, you want to do something a bit more complex than display an alert dialog; you want to change the size. When changeSize is fired, you want to know which paragraph to change. An event handler always is passed "this", an all-too-common word in JavaScript that confuses many people.

One way to get the sizes to rotate is shown in Listing 3, app.coffee. Basically, your callback function starts off by assigning a local variable, "text". If this were JavaScript, "text" would not be a local variable, but rather a global one, because you used neither the "var" keyword nor another object (for example, myObject.text). In CoffeeScript, variables are local, which means you cannot pollute the global namespace accidentally.

Listing 3. app.coffee

```
$(document).ready ->

    changeSize = ->
        text = $(this)

        if text.hasClass "small"
            text.removeClass "small"
            text.addClass "medium"

        else if text.hasClass "medium"
            text.removeClass "medium"
            text.addClass "large"

        else if text.hasClass "large"
            text.removeClass "large"
            text.addClass "small"

        else
            text.addClass "large"

        true

    ($ "p").addClass "large"
    ($ "p").live 'click', changeSize
```

Listing 3 shows a basic use of if/then/else blocks. Notice there isn't any need for braces, begin/end statements or other markers. Python programmers will see this (rightly) as a vindication of semantically significant whitespace. I just like the fact that well-indented code is easy to read,

and that CoffeeScript enforces this on me.

You also can see that with rare exception, you've managed to get rid of the parentheses that JavaScript would require, in favor of terse, clean syntax. You're still using the same jQuery methods, but you're doing so in a way that I find easier to read.

You then take the changeSize function and attach it to an event:

```
($ "p").click changeSize
```

It might look a bit strange to have the parentheses around the call to `$ "p"`, which in standard jQuery would look like:

```
$("p")
```

CoffeeScript tries to get rid of as many parentheses as possible, but there are times when the ambiguity would makes things too difficult for its parser. In such circumstances, you can use parentheses to make things easier.

As you can see from the above example, CoffeeScript makes all of jQuery's functions available. No matter

what you might want to do to the text or HTML of your document, you can use CoffeeScript to do it—adding and removing (and querying) nodes, adding and removing (and querying) attributes, changing text, invoking menus or anything else you can do in JavaScript. Having jQuery around means you can make use of its syntax and abstractions, a potentially killer combination. Indeed, a number of blog postings (including several mentioned in the Resources section for this article) indicate that the combination of CoffeeScript and jQuery is a popular and effective one.

Conclusion

jQuery is a popular framework for client-side Web development, providing a large number of abstractions and convenience functions for querying and modifying the DOM. CoffeeScript is a language that makes it easier to write in JavaScript, by simplifying the syntax, removing some of the most common problems that people have with the language, and providing easier ways to work with strings, arrays and hashes. But at the end of the day, both jQuery and CoffeeScript are tools for working with JavaScript, which means there's full interoperability between them. Although the examples in this column are simple, they demonstrate that it's easy to get started with CoffeeScript and even to integrate it into an existing application. My guess is that CoffeeScript has a very bright future and, I should add, deservedly so.■

Reuven M. Lerner is a longtime Web developer, architect and trainer. He is a PhD candidate in learning sciences at Northwestern University, researching the design and analysis of collaborative on-line communities. Reuven lives with his wife and three children in Modi'in, Israel.

Resources

The home page for CoffeeScript, including documentation, quick references, FAQs and annotated source code, is at **jashkenas.github.com/coffee-script**. There is an active and growing community of CoffeeScript users, with an IRC channel (#coffeescript) and Wiki at GitHub.

A good introduction to CoffeeScript is this presentation written by Jacques Crocker: **coffeescript-seattlejs.heroku.com**.

PeepCode (**peepcode.com**), which makes excellent screencasts on a variety of subjects, has one about CoffeeScript that I learned from and enjoyed.

There are many blog postings about CoffeeScript and jQuery. Stefan Buhrmester wrote a nice description of using jQuery with CoffeeScript: **buhrmi.tumblr.com/post/5371876452/how-coffeescript-makes-jquery-more-fun-than-ever**. And, Aaron Russell describes his experience combining CoffeeScript with jQuery: **aaronrussell.co.uk/articles/using-coffeescript-with-jquery**.

Finally, the Pragmatic Programmers have released (at the time of this writing) an excellent pre-release "beta book", written by active CoffeeScript user Trevor Burnham. If you're interested in learning more about this interesting little language, I highly recommend this book. It's aimed mostly at beginners, but given the limited number of advanced CoffeeScript programmers out there, that shouldn't bother you.

GROOVE

- Solid outdoor design
- Quick and easy to mount
- 2 or 5GHz wireless module
- Up to 125Mbit aggregate speed
- 53000pps wireless throughput
- N-male connector
- Signal strength LED indicators
- One 10/100 Ethernet port w/PoE
- FCC and CE certified
- Voltage and temperature monitors
- 16kV ESD protection on RF port
- Integrated mounting bracket, mounting ties, power adapter and PoE injector
- **Just $69** for the complete kit *

The Groove is our smallest outdoor series model - a fully featured RouterBOARD powered by RouterOS. Weatherproof, durable and ready to use. It has one 10/100 Ethernet port with PoE support and a built-in 200mW 802.11a/n wireless radio. With the Nv2 TDMA technology, 125Mbit aggregate throughput is possible!

It has a built-in N-male connector, and pole attachment points. You can attach it to an antenna directly, or use a standard antenna cable and mount it on a pole or mast. LED signal indicators make it easy to install and align.

Currently two versions are available: Groove 5Hn and Groove A-5Hn. Both of them can be used as clients or for point-to-point links, but Groove A-5Hn can also be used as an Access Point. The Groove runs RouterOS with all it's features.

* - $69 for the CPE/PtP model, $89 for the AP base station model. Units don't include antennas

MikroTik routerboard

www.routerboard.com

DAVE TAYLOR

Calculating Day of the Week, Finally

Dave wraps up the script and leaves us with the problem of Leap Year.

As with many of the challenges we tackle in this column, the latest project has sprawled across more issues than I ever expected when I first received the query from a reader. The question seems reasonably simple: given a month, day number and day of the week, calculate the most recent year that matches those criteria.

There are some obscure and complex formulas for doing just this, but instead, I decided it'd be interesting basically to loop backward from the current year for the month in question, parsing and analyzing the output of the handy cal program.

The real challenge has been that the cal program never really was designed to produce easily parsed output, so figuring out the day of the week (DOW, as we've been abbreviating it) involves basically counting the number of leading spaces or otherwise compensating for an average month where the first day starts mid-week, not neatly on Sunday.

An algorithmic-friendly version of cal would have output where days prior to the first day of the month would be output optionally as zeros or underscores, making this oodles easier. But it isn't, so we have to compensate.

Figuring the Day of the Week

Last month, we wrapped up with a shell function that expected the day, month and year as arguments and returned the day of the week of that particular date in that month on that year. In other words, 16 May, 2011, occurs on a Monday:

```
      May 2011
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 31
```

The actual return value of the function in this instance is 2, so 1 = Sunday, 2 = Monday, and so on.

Given the desired day of the week that the user specifies and a simple way to decrement the year until we hit a match coupled with the function already shown,

it should be relatively easy to assemble all the pieces and create—finally—the script that details when a specific date was on a specific day of the week.

I won't republish all the code from previous months (the completed script is 83 lines long), but here's the most salient portion at the end, the section that steps back year by year to figure out which one has a matching calendar entry:

```
echo Looking for $weekday, $day, $month \($monthnum\) \
    starting in $mostrecent
# now we need to loop backwards through years until a match
year=$mostrecent
DOW=-1                    # start with a dead value
while [ $DOW -ne $desiredDOW ]
do
  figureDOW $day $monthnum $year
# echo "> $day $month occurred on a $DOW in $year"
  year=$(( $year - 1 ))
done
echo "Got it! $day $month occurred on a $weekday
 ➥most recently in ${year}:"
cal $month $year
```

Notice that when we find a match, we not only print out what year had that

date on the specified day of the week, but we also print out the calendar for that month as a visual confirmation.

A few sample runs illustrate:

```
$ whatyear Friday February 9
Got it! 9 feb occurred on a fri
 ➥most recently in 2006:

   February 2006
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

$ whatyear wed aug 3
Got it! 3 aug occurred on a wed
 ➥most recently in 2004:

    August 2004
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 31
```

Since we convert the day of the week name and the month name to all lowercase and then truncate anything after the first three letters, you can see that "Friday" and "wed" both work, which is a nice side benefit. Applications with more flexible input options obviously are greatly preferred and make everyone's life easier.

Something's Still Broken

One date breaks the script because it doesn't occur every year: February 29. Here's the problem in a nutshell:

```
$ cal feb 2010
   February 2010
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
```

When we try to find a match for "29" on this calendar, there's no matching output, and the conditional tests we have in the script can't handle the empty string.

It's not pretty:

```
$ whatyear mon feb 29
./whatyear.sh: line 21: [: -eq: unary
 ➥operator expected
./whatyear.sh: line 72: [: -ne: unary
 ➥operator expected
Got it! 29 feb occurred on a mon
 ➥most recently in 2010:

   February 2010
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
```

You know, if we had these ugly "[" test error messages but the end result was correct, I probably could live with it, but you can see that it's matched on a February that doesn't even have a 29th day—lame.

However, fixing it might be more trouble than it's worth, and it certainly will cause us to sprawl into a subsequent column. Instead, I encourage you to grab the entire source code library from **ftp.linuxjournal.com/pub/lj/listings/issue209/11090.tgz**, and explore how to fix it yourself. Yes, I am punting!

Next month, I'll start on a new shell scripting challenge, and as usual, I encourage you to send me a quick e-mail note with some ideas you have on what would be compelling for us to develop or any particularly interesting scripting problems you're facing.■

Dave Taylor has been hacking shell scripts for a really long time, 30 years. He's the author of the popular *Wicked Cool Shell Scripts* and can be found on Twitter as @DaveTaylor and more generally at www.DaveTaylorOnline.com.

KYLE RANKIN

Remotely Wipe a Server

What would you do if you had to erase all the files securely on a server thousands of miles away?

In many ways, I feel sorry for people stuck with proprietary operating systems. When something goes wrong or if they have a problem to solve, the solution either is obvious, requires buying special software or is impossible. With Linux, I've always felt that I was limited only by my own programming and problem-solving abilities, no matter what problem presented itself. Throughout the years that Linux has been my primary OS, I've run into quite a few challenging and strange problems, such as how to hot-migrate from a two-disk RAID 1 to a three-disk RAID 5, or more often, how to somehow repair a system I had horribly broken.

The Problem

Recently, I ran into an interesting challenge when I had to decommission an old server. The server had quite a bit of sensitive data on it, so I also had to erase everything on the machine securely. Finally, when I was done completely wiping away all traces of data, I had to power off the machine. This is a relatively simple request when the server is under your desk: boot a rescue disk, use a tool like shred to wipe the data on all the hard drives, then press the power button. When the server is in a remote data center, it's a little more challenging: use a remote console to reboot into a rescue disk, wipe the server, then remotely pull the power using some networked PDU. When, like me, you have to wipe a server thousands of miles away with no remote console, no remote power, no remote help and only an SSH connection, you start scratching your head.

Why Would You Ever Do This?

At this point, some of you might be asking: "Why would you ever need to do this?" It turns out there are a few different reasons both legitimate and shady:

1. You have broken hardware. This could be a server with a broken video card, a malfunctioning KVM or remote serial console, or some other problem where physical hardware

access just doesn't work.

2. You are locked out from your server. This could happen, for instance, if you colocate your server in a data center but stop paying your bills or somehow have a falling out with the provider. They revoke your physical access to your server, but you need to remove all the sensitive files while the machine is still available over the network.

3. You have a bad consulting client. Perhaps you are a responsible and talented sysadmin who sets up a server for a client in good faith only to have that client refuse to pay you once the server is on-line. You want to remove your work securely, the client won't return your calls, yet you still have SSH access to the machine.

4. You bought a cloud server with inadequate tools. It is very popular these days to host your server environment in the cloud; however, one downside is that many cloud providers cut costs by giving you limited access to management of your cloud instance. Do you really trust that when you terminate a server instance it is securely erased? Do you get access to tools that would let you boot a rescue disk on your cloud instance? In some cases, about the only remote management you have for a cloud server might be your SSH connection.

5. You are an evil, malicious hacker who wants to cover his tracks. Yes, this is the least legitimate and most shady reason to wipe a server remotely, but I figured I should mention it in the interest of completeness.

6. It's a challenge. Some people climb mountains, others run marathons, still others try to wipe servers remotely over SSH. You could just be a person who likes to push things to the limit, and this sounds like an interesting challenge.

How Would You Ever Do This?

Now that you have worked through the reasons you might need to know how to wipe a server remotely, let's talk about how you actually would do it. First, and most important, there are **no redos**! When you write random bits to a raw disk device, especially over SSH, you have only one shot to get it right. When I was preparing this process, I tested my procedure multiple times on virtual machines to make sure my steps were sound. I'm glad I did, as it took a few times to get all the steps right, confirm my assumptions and get the commands in the correct order.

What makes this challenge tricky is the fact that you will write randomly over

the very filesystem you are logged in to. What happens if you overwrite the sshd and shred files while you are running shred and logged in over SSH? More important, what happens when you overwrite the kernel? The main principle that will make this procedure work is the fact that Linux likes to cache files to RAM whenever it can. As long as you can make sure everything you need is stored in RAM, you can overwrite the filesystem as much as you want. The trick is just identifying everything you need to store in RAM.

Always Have a Plan B

So, I mentioned there was no redo to this procedure, but that doesn't mean you can't set up some sort of safety net for yourself. Although I knew that once I launched the shred command it would run completely from RAM, what I had to figure out was what commands I would need to run *after* shred. Even commands like `ls` won't work if there's no filesystem to read. So that I would have some sort of backup plan, I took advantage of the /dev/shm ramdisk that all modern Linux systems make available. This is a directory that any user on the system can write to, and all files will be stored completely in RAM.

Because I wasn't sure whether commands like echo (which I would need later) would work after I had shredded the hard drive, I copied it to /dev/shm along with any other files I thought I would need. If you have the space, why not copy all of /bin, /sbin and /lib if you can. Finally, I knew I would need access to the /proc filesystem to power off the server. I assumed I still would have access to /proc even if I had overwritten the root filesystem, but I wasn't 100% certain, so just to be safe, I became root (you can't assume sudo will work later) and mounted an extra copy of /proc under /dev/shm as the root user:

```
$ sudo -s
# mkdir /dev/shm/proc
# mount -t proc proc /dev/shm/proc
```

It turns out I ultimately didn't need any of these precautions, but it doesn't hurt to be prepared.

It's Clobbering Time

Now is the point of no return. Just to be safe, I changed to the /dev/shm directory so my current working directory would be on a ramdisk. Then, I unmounted any unnecessary mountpoints (like /home) and ran the shred command below on every nonroot drive on the system. In my case, I used software RAID, so I also took the extra step of hot-removing all but one drive from any RAID array and shredded them separately. Finally, I was left with just my root filesystem stored on /dev/sda, so I took a deep breath and typed the following command:

Connecting you to the right people for Linux based careers

LinuxCareer.com is a leading online IT employment solution for both employers and job seekers, bringing together the talents of high calibre candidates with employers looking for individuals to meet their required skill level. **LinuxCareer.com** specialises in a niche market to ensure that only the best opportunities are available.

Employers

- Search resumes
- Receive resumes via email
- Application tracking
- Screening questionnaire
- Various subscription options
- Login and import data using Linkedin and Facebook

and more...

Job Seekers

- Resume uploads
- Application tracking
- Job alerts
- Save searched jobs
- Job board mobile version
- Login and import data using Linkedin and Facebook

and more...

linuxcareer.com

LinuxCareer.com uses an advanced and dedicated portal with features such as a highly powerful search engine with multiple search options, different subscription options, world-wide job listings and SEO. All of this comes with the assurance of high security to protect your personal details.

The portal's connection also links to Facebook and Linkedin so you can use your social profile when registering, with Linkedin providing you any relevant links or information for the ad you are searching, while Facebook allows you to autofill resumes directly from your Facebook profile.

For the best in Linux careers visit LinuxCareer.com today. We'll help you connect.

LinuxCareer.com Connecting Talent

linuxcareer.com

```
# shred -n2 -z -v /dev/sda
```

This command writes random bits to /dev/sda for two complete passes (-n2) then does a final pass with zeros so the drive looks perfectly clean (-z) with verbose output so I can see what's going on (-v). Of course, adjust the -n argument to your particular level of paranoia—two passes was fine for me. I have to admit, there's something satisfying and strange about overwriting the root filesystem while you are still logged in.

Once the shred process completed, I had a completely empty filesystem. It was weird—commands like `ls` gave odd errors, and I knew I was isolated in my /dev/shm jail. All that was left was to shut down the server, but how do you do that when /sbin/shutdown is erased? No problem, you might say, just kill PID 1, since if you kill init, it will halt the system. That would work if, say, the kill program still were around. In this case, the only way I had left to shut down the system was via the /proc interface. The /proc directory is a special filesystem that allows you access to processes and kernel information, and it resides entirely in RAM, so my little shred stunt didn't wipe it out. To halt the machine, just enable the sysrq interface in the kernel, and then send the right command to sysrq:

```
# echo 1 > /proc/sys/kernel/sysrq
```

```
# echo o > /proc/sysrq-trigger
```

If the halt command doesn't work, or if you just want to reboot the machine instead, you would type:

```
# echo b > /proc/sysrq-trigger
```

Now you might be asking yourself, didn't I overwrite the echo command? After all, /bin/echo is on the root filesystem. It turns out I didn't even have to rely on my copy of the command under /dev/shm—echo is one of the programs that are built in to the bash shell. When you execute echo, bash executes the version that is built in to itself, and because I already was inside a bash shell, the executable ran from RAM. Once you run the last echo command, the kernel instantly will halt. Any remote pings or other commands will stop, and the system will be powered off.

As a final note, I want to say that even if you don't think you'll ever need to go to such lengths to wipe a server, I think this procedure is such fun that you should at least try it in a disposable virtual machine. Shred the system and see which commands still work and which ones don't. As an extra challenge, see if you can get commands to run within /dev/shm.■

Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

8th Annual

High Performance Computing on Wall Street Show and Conference

September 19, 2011 (Monday) **Roosevelt Hotel, NYC**

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

SAVE THE DATE

Register today online. See HPC, Low Latency, Networks, Data Centers, Speed, Cost Savings. Wall Street markets will assemble at the 2011 HPC Sept. 19 show to see these new systems live on the show floor.

www.flaggmgmt.com/hpc

High Performance Computing, Low Latency, Networks, Data Centers, Cost Savings – the largest meeting of High Performance Computing in New York in 2011.

This HPC networking opportunity will assemble 800 Wall Street IT professionals at one time and one place in New York in September 2011.

This show will cover High Performance Computing, High Frequency Trading, Low Latency, Networks and Switch Solutions, Data Centers, Virtualization, Grid, Blade, Cluster, overcoming Legacy systems.

Our Show is an efficient one-day showcase and networking opportunity.

Register in advance for the full conference program which includes general sessions, drill down sessions, an industry luncheon, coffee breaks, exclusive viewing times in the exhibits, and more. Save $100. $295 in advance. $395 on site.

Don't have time for the full Conference? Attend the free Show. Register in advance at: www.flaggmgmt.com/hpc

Wall Street IT speakers and Gold Sponsors will lead drill-down sessions in the Grand Ballroom program.

This Show is a networking opportunity for the entire IT community.

Sponsors

Microsoft · IBM · intel · hp · BLADE Network Technologies · redhat · CISCO · FORSYTHE Realize the business value of IT.SM · ARISTA · SUPERMICRO · AMD · DataDirect Networks · Corvil · FULCRUM microsystems · Solace Systems · Myricom · vmware · TIBCO · Syncfusion · LINUX Pro Magazine · ADMIN Network & Security · UBUNTU user · LINUX JOURNAL · HPC wire · HPC In the Cloud · SECURITIES TECHNOLOGY MONITOR · TRADERS Magazine · SD Times · Global Investment Technology · Inside Market Data · waters · DWT · FINANCE ON WINDOWS · WINDOWS IN FINANCIAL SERVICES · WSTA Wall Street Technology Association

Show Hours: Mon, Sept 19 8:00 - 4:00
Conference Hours: 8:30 - 4:50

Show & Conference:
Flagg Management Inc
353 Lexington Ave, NY10016
(212) 286 0333 flaggmgmt@msn.com

Visit: www.flaggmgmt.com/hpc

DeLorme's inReach

We'll be expecting letters to the editor from Antarctica once the more intrepid among you get DeLorme's new inReach, a personal communicator that delivers two-way communication beyond the reach of cell-phone signals and one-way satellite systems. The Iridium Communications-based inReach offers "pole to pole" two-way satellite text messaging, delivery confirmations, SOS capabilities, remote tracking and an Android smartphone interface. The device's core communications component is the Iridium 9602 short-burst data transceiver, which utilizes the company's far-reaching satellite network. The GPS-enabled inReach can be used by itself or paired with either an Android smartphone or a DeLorme Earthmate PN-60w. With the standalone inReach, users can send pre-loaded text messages to designated recipients and activate remote tracking, allowing others to follow one's travels on-line via a "bread crumb" trail. When paired with an Android or the DeLorme Earthmate PN-60w, users enjoy full-featured, two-way text messaging to and from e-mail addresses and cell phones, as well as the ability to post to Facebook and Twitter.

www.delorme.com

Mellanox's FDR 56Gb/s InfiniBand Solutions

Mellanox recently unveiled a complete end-to-end solution for FDR 56Gb/s InfiniBand consisting of adapter cards, switch systems, software and cables, a feat that the firm calls an industry first. The solution consists of Mellanox's ConnectX-3 FDR 56Gb/s InfiniBand adapters, SX-6000 series switch systems, Unified Fabric Manager (UFM), Mellanox OS, software accelerators and FDR copper and fiber cables. As a package, it delivers "the highest level of networking performance while reducing system power", according to the company. The combination enables cost-effective networking topologies for HPC, financial services, database, Web 2.0, virtualized data centers and cloud computing.

www.mellanox.com

Compact SATA Modules - S3 Series

Emphase's S3 Series Compact SATA modules

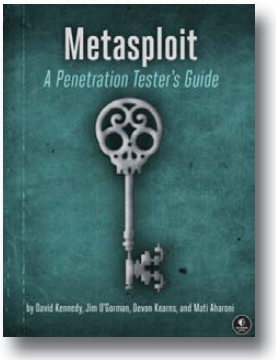
Emphase's new S3 Series line of compact SATA modules—that is, the CFast, SATA Flash Module, and mSATA—offers big performance in a small package, combining performance, dependability and longevity with extremely low power consumption and the most compact footprint. When space is an issue, this team of SLC NAND solid-state devices is an ideal solution. Transfer speeds have been increased to 120R/100W; capacities range from 1GB–32GB, with a 64GB capacity in the pipeline. The quick-and-rugged CFast S3 is the solution for data mobility or where hot-swap functionality is a must. The SATA Flash Module S3 weighs in at less than 10 grams and plugs directly, vertically or horizontally, in to a board's SATA port. The mSATA S3 is ideal for a low-profile embedded system, integrating high performance and capacity at less than 4mm thick. All modules now integrate TransferSAFE technology to weather-inconsistent power scenarios and operate off as little as 0.5 Watts, well below the average of 3 Watts.

www.emphase.com

Acquia's Commons

Here's a recipe for a kick-booty social software solution: download Drupal, stir in Acquia's updated open-source Commons 2.0, mold carefully with your creative vision and skill and *voilà*—you've got yourself an engaging community Web site. Commons melds rapidly evolving social Web features, including activity streams, social networking, blogs, wikis, badges and events together with enterprise-class analytics, support and management services. The result is a "prepackaged open-source alternative to proprietary social business solutions" that gives companies extensive freedom to extend and adapt their community sites to meet unique business needs. Version 2.0 adds features such as increased style flexibility, access to Acquia Cloud, increased configuration options, more support resources and complete design and theming control, including prebuilt Commons themes. A number of service packages are available from Acquia.

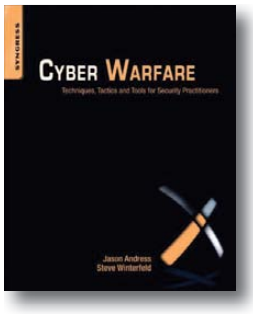
www.acquia.com



David Kennedy, Jim O'Gorman, Devon Kearns and Mati Aharoni's *Metasploit: A Penetration Tester's Guide* (No Starch)

You may be familiar with the the Metasploit Framework, a toolset that makes discovering, exploiting and sharing vulnerabilities quick and relatively painless. The challenge for security professionals is that the documentation is lacking, and it can be hard to grasp for first-time users. *Metasploit: A Penetration Tester's Guide* from No Starch Press intends to fill this gap by teaching how to harness the Framework, use its many features and interact with the vibrant community of Metasploit contributors. Readers will learn to find and exploit unmaintained, misconfigured and unpatched systems; perform reconnaissance and find valuable information about targets; bypass antivirus technologies and circumvent security controls; integrate Nmap, NeXpose and Nessus with Metasploit to automate discovery; use the Meterpreter shell to launch further attacks from inside the network; harness standalone Metasploit utilities, third-party tools and plugins; and learn how to write Meterpreter post-exploitation modules and scripts.

www.nostarch.com



Jason Andress and Steve Winterfeld's *Cyber Warfare: Techniques, Tactics and Tools for Security Practitioners* (Syngress)

Jason Andress and Steve Winterfeld's new book *Cyber Warfare: Techniques, Tactics and Tools for Security Practitioners* explores the battlefields, participants and the tools and techniques used during today's digital conflicts. The concepts discussed in this book will give those involved in information security at all levels a better idea of how cyber conflicts are carried out now, how they will change in the future and how to detect and defend against espionage, hacktivism, insider threats and non-state actors like organized criminals and terrorists. The authors provide concrete examples and real-world guidance on how to identify and defend a network against malicious attacks, dive deeply into relevant technical and factual information from an insider's point of view, and outline the ethics, laws and consequences of cyber war and how computer criminal law may change as a result.

www.syngress.com

Likewise Open

With Likewise Open, recently upgraded to version 6.1, admins can standardize on Microsoft Active Directory in their enterprise networks without losing the flexibility to choose non-Microsoft operating systems. The open-source Likewise Open allows single sign-on for critical enterprise applications, such as Apache Tomcat, IBM WebSphere, Oracle WebLogic and JBoss Application Server. Newly open-sourced integrations include the Kerberos/NTLM JAAS login module and SPNEGO Kerberos/NTLM servlet filter, which can be integrated with any servlet spec2.3-compliant application server. Customers wishing to extend event management, auditing and reporting for compliance with SOX, PCI-DSS, HIPPA and other industry standards to these integrations can do so by upgrading to Likewise Enterprise.

www.likewise.com

QLogic's InfiniBand Fabric Suite

The new 7.0 release of QLogic's InfiniBand Fabric Suite (IFS)—a fabric management software package that enables users to optimize fabric performance and communications efficiency for HPC clusters—is now available. IFS 7.0's leading new feature is the integration of vFabric QoS (Quality of Service) with work-flow schedulers from Adaptive Computing and Platform Computing, making it possible for HPC users to set a priority and QoS level as the message-passing interface application is being scheduled. This reduces management overhead, simplifies cluster scheduling and optimizes use of the fabric. Other noteworthy features include an enhanced Fabric Viewer with Fabric Dashboard, improved static routing performance, an improved Congestion Control Algorithm and support for both NVIDIA GPUs and Red Hat Enterprise Linux 5.6/6.1. The result of this feature set, says QLogic, is an ability for HPC customers to obtain maximum performance and efficiency from their cluster investments while simplifying management.

www.qlogic.com

Please send information about releases of Linux-related products to newproducts@linuxjournal.com or New Products c/o *Linux Journal*, PO Box 980985, Houston, TX 77098. Submissions are edited for length and content.

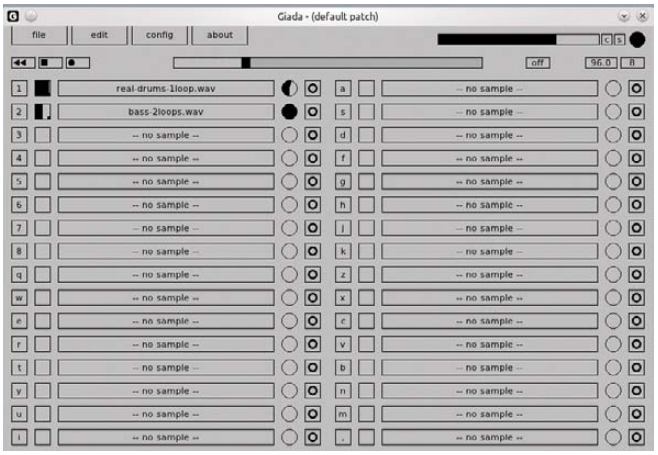
Fresh from the Labs

Giada—Hard-Core Live Looping

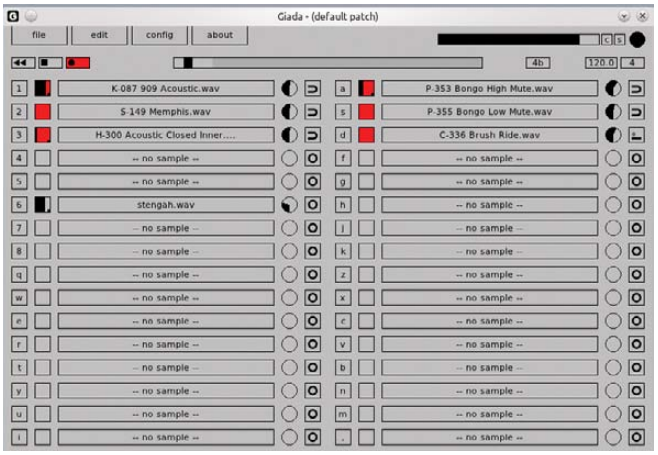
www.monocasual.com/giada

Live DJs chasing a simplistic but attractive application are going to love Giada. At the same time, Giada also covers something I've been wanting for a year now, which is a way to trigger individual samples from a computer keyboard simply in real time. According to the documentation:

> Giada is a free, minimal, hard-core audio tool for DJs and live performers. Load up to 32 samples, choose to play them in single mode (drum machine) or loop mode (sequencer), and start the show with your computer keyboard as a controller. Giada aims to be a compact and portable virtual device for Linux and Windows for production use and live sets.



The Giada hard-core looping program for real-time DJ performances with a GUI that's sleekly minimal.



Giada in its full-flight recording mode lets you layer a live performance piece by piece.

Installation Using Giada is pretty easy, but its ease of use comes at a price. It's a freeware binary. This is the first freeware program I've covered, but don't worry, I won't make a habit of it.

The project's Web site consists of a (well-designed) single page, with download links for both the Windows and Linux versions of Giada.

As far as library requirements go, the documentation states:

> Giada is based upon RtAudio and FLTK (GUI). They are both statically linked, but the former needs libjack.so to provide JACK's features. In a modern Linux-based OS, you should be able to run Giada without any further installation or hack.

Regarding binaries, the manual also noted: "This software is compiled for x86 processors; we still don't know what happens if you run it under a 64-bit OS/environment; try it and tell us your experience." Yes, I'm on 64-bit Linux, and it runs just fine.

Once you have the dependencies out of the way, download the latest tarball and extract it. Personally, I found I could just open the new folder, click on the binary, and it worked. For those wanting more control, open a terminal in the new folder, and enter the command:

```
$ ./giada_lin
```

Usage Once you're inside, using Giada is actually pretty easy. Although I was rather confused at first glance, a quick bit of "RTFM" shows that its methodology is very basic, but you need to understand a few things from the outset.

First, this isn't for programming songs over some sort of grid, such as the way that programs like Fruity Loops operate. Giada is for playing *live*. All of your actions take place in real time, as you perform what is essentially a live DJ set (even if it's only in your bedroom to an audience of one). So trust me, you'll want to practice before using it in public.

First, this isn't for programming songs over some sort of grid, such as the way that programs like Fruity Loops operate. Giada is for playing *live*.

Second, Giada is designed to be run by your keyboard, and by that I mean the thing on which you type, and not something that resembles a piano.

Before I explain the three modes of operation, let's first load some samples so we have something to play with. In terms of format, Giada likes only 44KHz .wav files. A good starting point is Hydrogen's drumkits. If you have Hydrogen installed (and if you don't, at least install the drumkits), look for .wavs under /usr/share/hydrogen/data/drumkits. Kicks, snares and a cymbal or two—hi-hats in particular—are the best starting point. With these, you can lay down a basic beat and then layer other samples over the top to make a song.

To load these samples, click on the long and wide buttons that say, "-- no sample --", and choose your .wav file from the file browser. Now, if you look to the left of each sample, a keyboard character is shown; with this, you switch samples on and off. Try pressing it now, and nothing will play, but fear not. I discuss Giada's three running modes below, and it all will make sense.

Oneshot mode: this is the most basic way of operating Giada. Press a

keyboard button, and that button's sample will play. However, first you must turn on this mode, as well as turn up this sample's volume. Starting with the volume, the empty circle to the immediate right is actually a volume knob. Clicking and dragging inside the circle turns up the volume. However, unless you've pressed the Play button, there still will be no sound; you have to enable Oneshot mode.

The next control to the right, with the small circle inside the square, is the key to operating Giada. Click the button, and you'll have a choice of looping modes, or Oneshot "basic", "press" and "retrig". Choose basic, press the key, and at last, a sound plays!

With this basic mode, you press a key and a sample plays until it's finished—pretty basic. But, you also can interrupt the sample by pressing the key again. With the "press" option, you have to hold down the key to play the sample, and as soon as you release it, the sample stops. The "retrig" option (and this is the functionality I've been chasing) plays a sample upon pressing a key, but pressing again restarts the sample whether or not the sample has finished. You even can keep thrashing away at the key for instant response, which is handy for playing hi-hat notes or ripping up a waveform.

Loop mode: this is the second mode, and perhaps the most conventional. When Giada is actually playing, choosing either "Loop . basic" or "Loop . once" plays a sample on the next bar along. In order to use this, press the Play button near the top-left corner, then press each sample's button to activate/deactivate it on the next bar. The "basic" option simply keeps the sample looping until you turn it off manually; the "once" option plays a sample until it's finished, and then starts it again at the beginning of the next bar.

Recording mode: this is the pièce de résistance. Basic loops can be turned on and off willy-nilly; however, the Oneshot samples start turning this into a real live performance. As the bar moves along each of its counts, every time you play a note, that note is repeated every bar that follows. Using this method, you genuinely can layer an entire song, creating new beats on the fly. Be wary, however. Every note you play is a commitment, and you'll be stuck with that note repeating for the rest of the song. Get it right, and you'll have people dancing. Get it wrong, and you'll look totally lame and ruin the party.

Nevertheless, some tools are available to help out the mere mortals among us. The beat bar will be immediately obvious, because it's the only moving thing on screen. Use this and its (default) four boxes to guide your counting. Over to the right, the box that's marked "off" is for quantizing your music. For the uninitiated, this

aligns your notes to even places on a musical grid, removing the element of human error: "1b" is the most severe, making each note land on a whole count; "8b" is the least severe, allowing you to make much more intricate music.

If you look farther right, you'll see a tempo and beat number, set to a default BPM of 120 and a time signature of 4/4. You can turn these up or down, allowing for strange feels, such as 7/4 @ 72 BPM (less dance-friendly, but much more trippy).

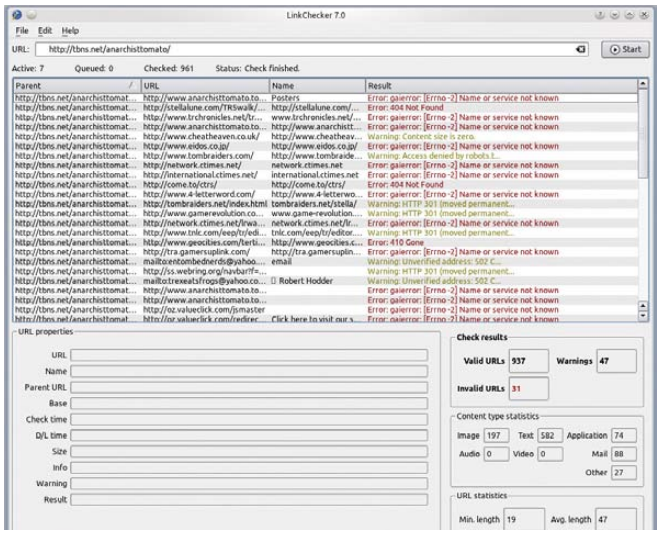
Still, this early software does have its limitations. First, it's freeware. In this day and age? Why? Ech! Second, there weren't any panning controls as far as I could tell. Any stereo imaging you'll have to do beforehand, manually.

Nevertheless, this program is incredibly cool. It allows you to output to JACK, which makes it more powerful, and just *look* at it. It's a techno-minimalist's wet dream! Giada has an amazing economy of space and features in its design that's quite deceptive. I actually thought this was going to be a very short review when I started. Giada is a must-have for any electronic musician.

LinkChecker—Web Site Testing

linkchecker.sourceforge.net

Broken links are a serious pain in the backside for Webmasters, and keeping track of every individual link becomes so laborious, most Webmasters simply give up on the idea. Thankfully, there's a way

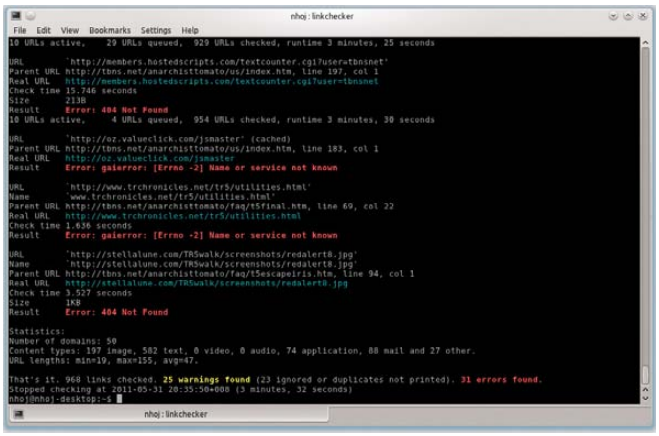


LinkChecker makes Web site maintenance that much easier by scanning your Web site for broken links.

to automate the process with LinkChecker. According to its Freshmeat entry:

> With LinkChecker, you can check HTML documents and Web sites for broken links. It features recursion, robots.txt exclusion protocol support, HTTP proxy support, i18n support, multithreading, regular-expression filtering rules for links and user/password checking for authorized pages. Output can be colored or normal text, HTML, SQL, CSV or a sitemap graph in DOT, GML or XML format. Supported link types are HTTP/1.1 and 1.0, HTTPS, FTP, mailto:, news:, nntp:, Telnet and local files.

Installation The Web site has packages for Windows, OS X and Debian (yes, it actually specifies Debian), and the



The effect of ten years' neglect on a Web site: here I'm running the console version against my old Tomb Raider page.

obligatory source. The Debian packages are available for just about every architecture on the planet, and they worked immediately with my Kubuntu installation. Installing the .deb package is much easier, so you may want to run with that; however, unless I missed something, you get only the command-line version.

For those who don't have a Debian-based system or want to use the GUI version, here's a very compressed version of the instructions.

In terms of dependencies, you'll obviously need gcc, as well as Python >= 2.6, including its -dev package, as well as the Qt development tools, which are named qt4-dev-tools on my system. There also are a bunch of optional library dependencies for extended functionality, such as bash completion, syntax checks and so on; see the manual for more information on these.

Once you have the dependencies out of the way, grab the latest tarball from the Web site, extract it, and open a terminal in the new folder. Enter the following commands:

```
$ make -C doc/html
```

(The above generates the Qt help files.) Then:

```
$ python setup.py sdist --manifest-only
$ python setup.py build
```

If your distro uses sudo:

```
$ sudo python setup.py install
```

If your distro uses root:

```
$ su
# python setup.py install
```

To run the command-line version:

```
$ linkchecker
```

To run the GUI version:

```
$ linkchecker-gui
```

Usage Actually using LinkChecker is a simple affair. If you're running the command-line version, enter:

```
$ linkchecker http://{website url}
```

Of course, LinkChecker also can scan

local files, but unless the page starts with www, remember to put the preceding http:// before on-line pages, or it automatically scans local files instead.

Once inside, enter the URL in the bar at the top and press Enter.

Whether in the GUI or console version, LinkChecker gradually makes its way through all the pages of a given Web site, outputting any broken links or warnings in the process. Depending on the Web page, the output can be pretty verbose, so console users might consider piping the output for larger pages.

Once the scan has finished, a readout is provided with the number of valid and invalid URLs, as well as various statistics having to do with URLs and content.

In the end, LinkChecker is a very simple project that serves its purpose beautifully. Its ease of use and multi-platform nature also make everyday usage much more likely. Any serious Webmaster should check out this project.■

John Knight is a 27-year-old, drumming- and bass-obsessed maniac, studying Psychology at Edith Cowan University in Western Australia. He usually can be found playing a kick-drum far too much.

BREWING SOMETHING FRESH, INNOVATIVE OR MIND-BENDING? Send e-mail to newprojects@linuxjournal.com.

MULTIPLATFORM GNU DEVELOPMENT

Get a guitar synth working with *Rock Band 3*.

NATHANAEL ANDERSON

In my ideal world, mixing games and music would result in music games that use real instruments. Harmonix's *Rock Band* series is the closest mainstream realization of this lofty ideal, except for one major issue. I couldn't plug in my guitar and play pro guitar mode songs out of the box.

Now, before you think this is an impossible task, one important fact should be noted. I play a guitar synthesizer. What this means is my guitar has a hexaphonic pickup that reads and processes every string's signal individually and connects with a 13-pin cable to a guitar processor with midi out. This means my signal is already digital and, therefore, does not require any additional A/D algorithms in my software.

Custom-built guitar synth, built by the author.

The Hardware

I have a few guitars with different 13-pin interfaces that can connect to an Axon AX-100 that has its midi out running to the midi in port on an M-Audio UNO, which gets connected to my Linux box running g2ghpro. The midi out port on the UNO then runs into a PS3 *Rock Band* adapter and into my PS3. A guitar hexaphonic pickup with 13-pin midi out is required to run g2ghpro. The following lists are of tested hardware that will work:

Pickups:

- Roland GK-3a.
- Roland GK-2a.
- Roland GK-3b.
- Graph Tech LB-63 (any Graph Tech piezo bridge will work if using the hexpander module).
- Godin Synth Access guitars.

Guitars with built-in 13-pin capability:

- Godin LGX-SA.
- Godin Freeway-SA.
- Godin LGXT.
- Roland Ready Fender stratocaster.
- Brian Moore i8.13.

Guitar-to-midi converters:

- Axon Ax-100.
- Roland VG-99.
- Roland GR-55.
- Roland GR-20.

Of the listed hardware, I use a Roland VG-99, Axon Ax-100, Godin LGX-SA, Godin LGX with Roland GK-3a and custom-built Ibanez S540 with Graph Tech LB-63 and hexpander.

I've also tested g2ghpro with a Roland GK-3a pickup running into a Roland VG-99. Others have reported using a Roland GK-3a with GR-20 and GK-3b with GR-55 on a bass guitar as well. G2ghpro currently is the only solution for using a real bass guitar in the game, as no official bass guitar controllers have been released at this point.

The Original Controller

Harmonix created two guitar controllers for *Rock Band 3* pro guitar mode. The first controller released was the Mustang, which is a "button" controller, with 102 buttons on the frets and six strings over the body of the guitar. When I wrote the initial version of g2ghpro, the Fender Squier Pro Strat wasn't on the market yet, which is the other pro guitar controller that works with the game. So I had midi dumps only from the Mustang to work with.

Midi

In order to understand how the Mustang worked, I first had to understand what the Mustang dumps meant and relate controller actions to messages sent. This required a refresher on the midi standard. Midi has 16 separate channels, and changing the sending channel is done by adding the value of the channel minus one to the message type value. My first example is a midi "note on" message, which has a decimal value of 144 in the first byte for channel 0. To send the same message on channel 6, add 5 to 144.

Example Data

Action: held fret one of low E and picked low E:

```
TIMESTAMP IN PORT STATUS DATA1 DATA2 CHAN NOTE   EVENT
0023E843    1     --     95    29    00  6 F 2  Note Off
0023E843    1     --     95    29    7D  6 F 2  Note On
0023E847    1     --     F0    Buffer: 8 Bytes System Exclusive
SYSX: F0 08 40 0A 05 06 7D F7
```

The dumps I had found came from a Windows program called Midi Ox, which showed the data in hex. The midi specification shows data in binary, and I was used to seeing this data with aseqdump in decimal. I converted all the examples I had been provided with into decimal so I could understand their behavior. The midi spec states that note on-and-off events contain 3 bytes of data. The first byte is event type plus channel; the second is

From left to right: custom-built guitar synth, custom-built rack for VG-99, Axon AX-100 and Fantom-XR.

note number, and the third is velocity. From experience, I've seen that many devices send a velocity event of 0 instead of an actual note off event, which is what the above shows. So, the result after it has been converted to decimal is (note: velocity zero below is really note off):

Type	Channel	Note	Velocity
(Note On 149)	5	41	00
(Note On 149)	5	41	125

SysEx is short for System Exclusive messages, and they are free-form messages to send data that doesn't fit into the predefined midi message types. Initially, I tried to treat the data from the dumps as a normal midi controller, and I ignored the SysEx data in the

dump, which I later realized is why I didn't have any code that made the game react. All game functions react to SysEx messages, not note events. This is why a guitar synthesizer cannot be plugged in to *Rock Band* and just work. At this point, I requested more dumps, where different frets were pressed down with the same string pressed.

I converted all the dumps I had to decimal and compared SysEx messages to note messages and found a correlation. Here's the resulting structure of the messages (displayed in decimal):

```
Part          1   2   3   4 5 6   7   8
Sample      240 8 64 10 1 1 43 247
```

- Part 1: starting byte of a SysEx message.
- Part 2, 3, 4: identifiers that this is a SysEx message used by the Mustang.
- Part 5: message type (1 = set fret position, 5 = play string).
- Part 6: midi channel (string on the instrument).
- Part 7: midi note number.
- Part 8: end SysEx message.

The Software

To explore the message format, I put together a quick program for sending a combination of note events and SysEx messages to the game. I know that the guitar synthesizer hardware required to use the software I was writing isn't very common, so I want to be proactive in removing any limitations to people using it. I've done midi and C++ programming with ALSA under Linux before, but never midi on Windows or OS X, and I wanted to be able to support all three to make the software more accessible.

From past experience with RtMidi, I knew it was written in portable C++, while supporting Windows, Linux and OS X.

The home page for RtMidi provides detailed, easy-to-read documentation, with examples for many basic midi tasks. Copy-and-paste examples are provided that give a base from which to start working, along with ready-to-compile demos provided in the tests directory in the RtMidi source code.

A good place to start with RtMidi is the bundled code in the tests directory. I started by modifying midiout.cpp and tried sending different SysEx messages based on my data dumps until finally I ended up with the following:

```
std::vector<unsigned char> sysExMessage;
sysExMessage.push_back( 240);
sysExMessage.push_back( 8 );
sysExMessage.push_back( 64 );
sysExMessage.push_back( 10  );
sysExMessage.push_back( 1); // 1 sets fret position,
                            // 5 to play the current string
sysExMessage.push_back( channel + 1 ); // channel
sysExMessage.push_back( note );
```

```
sysExMessage.push_back( 247 );
midiout->sendMessage( &sysExMessage );
```

With the above SysEx message, I was able to toggle fret position and strings played in the game. The actual logic to make this work was less than 100 lines. The full code is available in the Subversion repository for game2midi in g2ghpro.cpp.

RtMidi currently has issues processing active sensing messages that came from my Roland gear, which the author is aware of. A flag is provided to filter out active sensing messages. Setting ignoreTypes to true on your midi input object's third parameter will work around the issue until it is resolved—for example:

```
midiin->ignoreTypes( false, false, true);
```

The main missing feature of RtMidi, as far as the Linux pro audio world is concerned, is no jack-midi support.

The RtMidi documentation listed compiler flags for all three operating systems to link the required libraries, so all that was left for me to do was figure out how to compile under Windows.

Supporting Other Operating Systems

I hadn't touched a Windows development IDE in more than ten years, and I wanted to keep the same code base for all three operating systems. Somewhere during the past few years, I heard mention of MinGW (Minimalist GNU for Windows). As I am familiar developing in a GNU/Linux environment, this sounded like what I needed. To bring your Linux dev environment to Windows, use `mingw-get-inst`, and do a full installation. This will provide you with the MinGW Shell, bundled with many standard GNU tools, including SSH. Next, install TortoiseSVN, which is a Subversion client that integrates with the Windows shell. Checkout and commit actions are accessed by right-clicking on folders in Explorer to keep files in sync. The MinGW shell allows for changing drives' letters like a standard DOS shell with `cd C:`.

The next problem is how to build the code based on operating system. Let's look at two options: Makefiles and autotools. First, let's look at basic Makefile-based builds and compare the differences by platform:

```
# Makefile.linux

all:
        mkdir -p deps
        g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -D__LINUX_ALSASEQ__
        ➥-g -O2 -MT midiio.o -MD -MP -MF deps/RtMidi.Tpo
        ➥-c -o RtMidi.o RtMidi.cpp
        g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -D__LINUX_ALSASEQ__
        ➥-g -O2 -MT -midiio.o -MD -MP -MF deps/midiio.TPO
        ➥-c -o midiio.o midiio.cpp
        g++ -g -O2 -o midiio RtMidi.o midiio.o -lasound

# Makefile.mingw

all:
        mkdir -p deps
```

```
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -D__WINDOWS_MM__
➥-g -O2 -MT RtMidi.o -MD -MP -MF deps/RtMidi.Tpo
➥-c -o RtMidi.o RtMidi.cpp
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -D__WINDOWS_MM__
➥-g -O2 -MT -midiio.o -MD -MP -MF deps/midiio.TPO
➥-c -o midiio.o midiio.cpp
g++ -g -Wl,--enable-auto-import -O2 -o midiio RtMidi.o
➥midiio.o -lwinmm
```

The library I used, rtmidi, requires that the platform be defined, so for Linux, define `-D__LINUX_ALSASEQ__`, and for Windows, define `-D__WINDOWS__MM__`. The last step in each is the linking phase, where you specify system libraries to link to the binary. To enable a clean build under Windows, I had to add `-Wl,--enable-auto-import`, so functions would be auto-imported.

Now, let's look at autotools-based builds. Usually when an autotools-based build is committed to version control, only non-generated files are committed, which includes configure.ac, Makefile.am and src/Makefile.am. The standard practice is to create an autogen.sh script that will call the files to generate configure, Makefile and other required files from *.ac and *.am files:

```
# Autogen.sh
#!/bin/sh
aclocal
autoreconf
automake --add-missing --copy
autoreconf
libtoolize -f --automake
```

configure.ac (Listing 1) contains host-based auto-detection and is where to specify which host-based libraries to check for and link against.

Full code for these examples can be downloaded from the game2midi project's Subversion repository in the basic-midi-io-example folder.

Conclusion

Using GNU libraries and tools can help reduce the time and effort required in supporting multiple platforms. I hope this article encourages you to consider adding multiplatform support to a current or future open-source project.■

Nathanael Anderson has been a UNIX systems administrator for five years. Family, coding and all things guitar keep him active. Feel free to contact him on his blog at wirelessdreamer.com.

Resources

Official Midi Specification: **www.midi.org/techspecs/midimessages.php**

Posts of Mustang Controller Dumps: **www.rockband.com/forums/showthread.php?t=207792&page=1**

Official RtMidi Home Page: **www.music.mcgill.ca/~gary/rtmidi**

MinGW: **www.mingw.org**

Mingw Download: **sourceforge.net/projects/mingw/files/Automated%20MinGW%20Installer/mingw-get-inst/mingw-get-inst-20110316/mingw-get-inst-20110316.exe/download**

TortiseSVN: **tortoisesvn.tigris.org**

game2midi Home Page: **game2midi.sourceforge.net**

Listing 1. configure.ac

```
#configure.ac

dnl - dnl represents a comment in automake config files

dnl here we specify the

AC_INIT(midiio,0.1)

AM_INIT_AUTOMAKE(midiio, 0.1)

AC_PROG_CXX

AC_LANG_C

dnl Checks for programs.

AC_PROG_AWK

AC_PROG_CC


dnl Check for headers

AC_CHECK_HEADERS(unistd.h)


dnl Checks for typedefs, structures, and compiler characteristics.

AC_TYPE_SIZE_T


dnl Detect OS we're building on

dnl this next line is required to be able to read the host value

AC_CANONICAL_HOST

dnl Use the value here to add support for other operating systems

echo "Host Value: '${host}'"


case "${host}" in

        *-mingw32*)

                dnl specify Windows specific compiler flags

                ➥and linker options

                compile_target=win

                CPPFLAGS="$CPPFLAGS -D__WINDOWS_MM__"

                LIBS="$LIBS -lwinmm"

        ;;

        *linux-gnu)

                dnl specify Linux specific compiler flags

                ➥and linker options

                compile_target=linux

                dnl Check for ALSA

                AC_CHECK_LIB(asound, snd_seq_event_output_direct,

                ➥alsalib=yes,alsalib=no)

                AC_CHECK_HEADERS(alsa/asoundlib.h,alsaheader=yes,

                ➥alsaheader=no)


                if test "$alsalib" = "yes"; then

                        if test "$alsaheader" = "yes"; then

                            LIBS="$LIBS -lasound"

                        else

                            AC_MSG_ERROR([** Coulnd't find ALSA

                            ➥header file sys/asoundlib.h **])

                        fi

                else

                        AC_MSG_ERROR([** Couldn't find ALSA library

                        ➥libasound. **])

                fi

                CPPFLAGS="$CPPFLAGS -D__LINUX_ALSASEQ__"

        ;;

esac


AC_HEADER_STDC

AM_CONFIG_HEADER(config.h)

AC_OUTPUT(Makefile src/Makefile)


# Makefile.am


# Here we specify we have files in the source directory to process

AUTOMAKE_OPTIONS = foreign

SUBDIRS = src


# src/Makefile.am

# Here we define there are 2 programs we're compiling

bin_PROGRAMS = midiio midiout

# and here we define what we put together to make the final programs

midiout_SOURCES = midiout.cpp RtMidi.cpp

midiio_SOURCES = midiio.cpp RtMidi.cpp
```

Performance Regression Monitoring with Gauger

Introducing Gauger, a lightweight tool for visualizing performance changes that occur as software evolves.

Regression testing is a well-established technique to detect both the introduction of new bugs and the re-introduction of old bugs. However, most regression tests focus exclusively on correctness while ignoring performance. For applications with performance requirements, developers run benchmarks to profile their code in order to determine and resolve bottlenecks. However, unlike regression tests, benchmarks typically are not executed and re-validated for every revision. As a result, performance regressions sometimes are not detected quickly enough.

Compared to correctness issues, performance regressions can be harder to spot. An individual absolute performance score rarely is meaningful; detecting a performance regression requires relating measurements to previous results on the same platform. Furthermore, small changes in external circumstances (for example, other processes running at the same time) can cause fluctuations in measurements that then should not be flagged as problematic; this makes it difficult to set hard thresholds for performance scores. Also, good measurements often take significantly longer than correctness tests. Performance improvements in one area may cause regressions in others, causing system architects sometimes to consider multiple metrics at the same time. Finally, performance can be

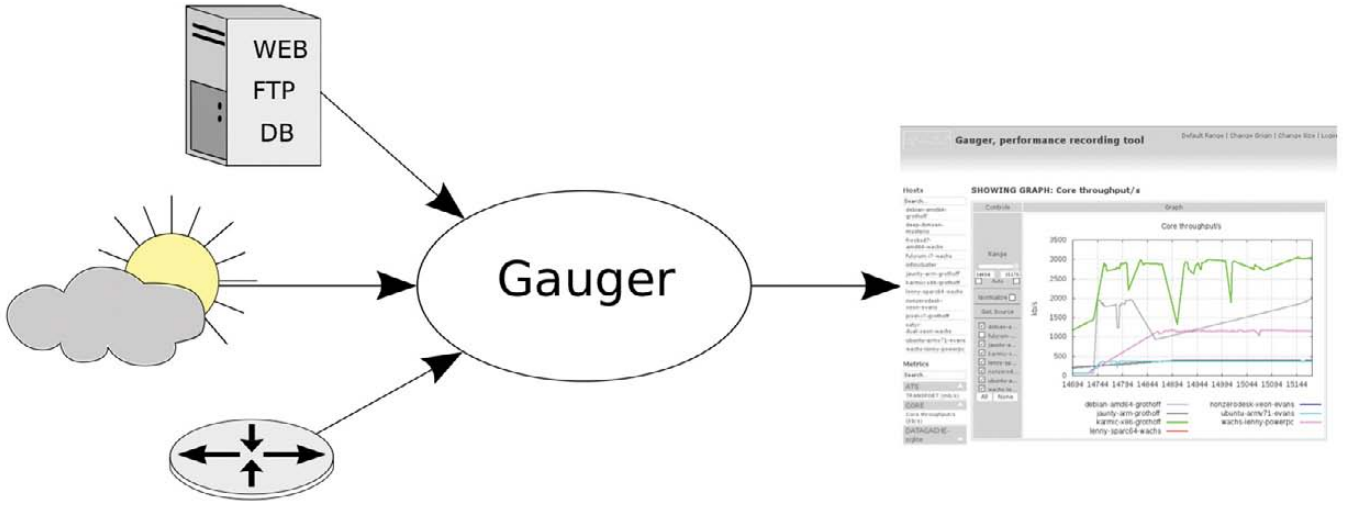


Figure 1. Gauger in action: this screenshot shows performance measurements obtained and visualized by Gauger for the GNUnet Project over the course of a few revisions.

platform-specific. This can make it necessary to perform performance evaluations on a range of systems.

The Gauger package described in this article provides developers with a simple, free software tool to track system performance over time. Gauger is lightweight, language-independent and portable. Gauger collects any number of performance values from multiple hosts and visualizes their development over time (Figure 1). In order to use Gauger, developers need to add the necessary instrumentation to their code to obtain a performance measurement and then submit it to Gauger with the `gauger` function call. The `gauger` function arguments are the description of the value, a category, the value itself and a unit of measurement. Gauger's Web interface then allows visitors to group metrics by category or by execution host and adjust the visualized revision range or the size of the plot. Gauger is ready to be used with many programming languages and revision control systems, and it is easily expandable to accommodate new ones.

Gauger's Web interface can be used to analyze the collected performance data in various ways. It can combine different metrics in a single plot and offers a color-coded guide to help visitors select only unit-wise compatible metrics. Gauger also allows users to normalize the data in order to mask differences in absolute performance between different execution hosts. If multiple measurements were taken for the same revision, Gauger will show the average and standard deviation as long as only a single metric is plotted. For larger projects with many metrics or execution

hosts, Gauger offers a search feature to locate the desired plots. An additional instant search keeps the menus free of irrelevant items.

Finally, should further fine-tuning be needed (for example, for use in presentations), Gauger can be used to retrieve the gnuplot source of any plot. The generated gnuplot source includes the plotted data.

Gauger's Architecture

Gauger uses a traditional client-server architecture, where the clients report performance measurements to a central server. This architecture allows machines behind NAT or with otherwise restricted Internet access to provide performance measurements to Gauger.

All of the performance-monitoring machines to be used with Gauger should install the Gauger client, and the software to be tested should be integrated with the appropriate language bindings. Language bindings are designed to be transparent and (except for a few extra system calls) have no negative effects in case the Gauger client is not installed on the machine. Thus, it is safe to commit the language bindings to a project repository. As long as the (tiny) language bindings are included, integrating Gauger will not disrupt operations on systems where the Gauger client is not installed.

The Gauger server runs the data collection and visualization part. Data is logged through a RESTful API and saved in human-readable plain-text files. The primary job of the server is to provide a dynamic Web interface to visualize and analyze the collected data. All the communication between Gauger clients and the Gauger server is done in standard HTTP(S) requests

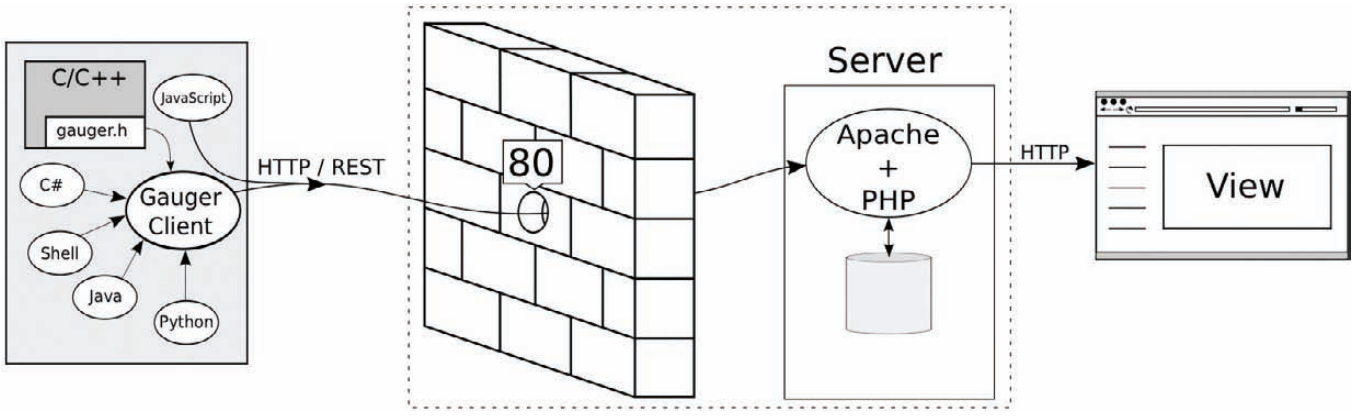


Figure 2. Gauger architecture: the Gauger server is responsible for authentication and receives performance data from the clients. The results are stored in a simple text format in a local directory. PHP is used to generate the Web site.

so that only port 80 (or 443) needs to be open (Figure 2).

Installation

Each Gauger client installation requires a local Python (> or = than 2.6) interpreter. For the Gauger server, a Web server installation with PHP and gnuplot is required.

The provided install.sh script can be used to install the client, install the server code into an appropriate location and generate an updated Apache configuration. The script prompts for key configuration options, such as the installation path and the desired URL at which the Gauger server should run. Installations that do not use Apache currently require manually configuring the Web server.

Configuration

Each part of Gauger uses a simple configuration file. The Gauger client configuration file contains the remote server URL, followed by the user name and password. Here's a sample configuration:

```
https://gnunet.org/gauger/ username password
```

The Gauger server configuration file contains the directory where data and authentication information are stored. Listing 1 shows a sample server configuration. When the auto-add feature is enabled, new hosts can be added by logging in to the Web site using a fresh hostname and password.

Listing 1. A few basic configuration options and a list of clients with their passwords' hashes are needed for the Gauger server configuration file.

```
[config]
data = "/var/lib/gauger"
salt = "makemyhashesunique"
auto_add = 1

[hosts]
debian-amd64-grothoff="da39a..."
```

Integrating Gauger

Adding a single simple call at the places where performance measurements are obtained typically is all that's required to integrate Gauger with existing projects. This call then starts the Gauger client process, which, if installed and configured correctly, submits the performance

Adding a single simple call at the places where performance measurements are obtained typically is all that's required to integrate Gauger with existing projects.

Listing 2. The GAUGER macro makes it easy to integrate Gauger with C code. Note that the code does not need to be linked against any additional libraries (other than libc).

```
#include <gauger.h>
#include <time.h>

int main() {
  time_t start = time (NULL);
  do_test ();
  time_t delay = time (NULL) - start;
  GAUGER ("subsystem", "execution time for f",
          delay, "s");
  return 0;
}
```

measurement to the server. On systems where the Gauger client is not installed, the call fails silently so as not to disrupt normal operations in any way. The syntax of the Gauger client command-line tool is as follows:

```
$ gauger [-i ID] -c CATEGORY -n NAME \
 -u UNIT -d DATA
```

Here, `NAME` is the name of the metric, and `DATA` is any floating-point value that represents the performance measurement. `UNIT` is a string that describes the unit of the value, for example, ms/request. `CATEGORY` is a string used to group multiple performance metrics by subsystem. We recommend using the name of the subsystem or module here, especially for larger projects.

Listing 3. A simple static method call, leading to a single line of Java code, can be used to invoke Gauger from Java.

```
import static org.gnunet.gauger.Gauger.gauger;

class HelloGauger {
     public static void main(String[] args) {
          gauger ("subsystem", "Speed",
                  42 /* value */, "kb/s");
     }
}
```

Revision Numbers

Gauger can autodetect the current revision of the project if the benchmark is executed in a directory that was checked out from a supported Version Control System (VCS). The supported VCSes are Subversion, Git, hg, bazaar, monotone and GNU arch. For distributed VCSes that do not provide an ascending revision numbering system (like Git), Gauger uses the number of commits to the repository. In this case, all execution hosts used for benchmarking should use the same origin to keep the data consistent. If the project uses an unsupported VCS or if the benchmark is executed outside the tree, Gauger needs to know which (revision) number to use for the

Listing 4. Example Code for Invoking Gauger from Python

```
from gauger import GAUGER

GAUGER("CAT", "NAME", 42, "kb/s")
```

x-axis. The `--id ID` option is used to supply the revision number in this case. Some projects may want to use an internal version number or a timestamp instead of a revision number generated by their VCS. The only restriction imposed on the numbers used is that Gauger's regression monitoring assumes a linear progression of development. For projects with multiple branches under active development, different Gauger servers should be set up for each branch.

Listing 5. The browser must be registered with the Gauger Web site before Gauger can be invoked from JavaScript. Once the login cookie is set, the main difference from other languages is that the JavaScript code must supply its revision number explicitly.

```
[...]
<script type="text/javascript"
  src="http://www.example.com/gauger.js">
</script>
[...]
 var rev = $Revision$;
 /* or */
 var rev = <?php echo get_revision() ?>;
 var performance = do_test();
 GAUGER ('CAT', 'NAME',
         performance, 'kb/s', rev)" />
[...]
```

Language Bindings

Gauger ships with bindings for various languages (see, for example, Listings 2, 3, 4 and 5) to make integration easy. The resulting binaries do *not* depend on a Gauger client installation on the target system. The bindings should be integrated with the main project and, as mentioned before, simply do nothing when invoked if the Gauger client is not installed or not configured properly.

The JavaScript binding is unusual. Because JavaScript cannot access the local filesystem to read the configuration file, the login data must be stored in a cookie in advance. The login page on the Gauger Web site, which usually is used to set up new accounts for execution hosts, can be used to set the respective login cookie in the browser. Access to the source code's repository also is not possible from JavaScript, so the revision number must be supplied explicitly to the `GAUGER` call. Typically, the current revision number is obtained on the server side. For example, PHP code can be used to obtain the number from the VCS, or on-commit trigger

Gauger provides developers complete freedom with respect to the names, values and units of the performance metrics to be monitored.

functions provided by the VCS could be used to insert the number into the source code.

Selecting Proper Units

Gauger provides developers complete freedom with respect to the names, values and units of the performance metrics to be monitored. So, how do you choose a good unit? Naturally, part of the unit always is dictated by the kind of performance characteristic you are interested in—for example, execution time (seconds) or space consumption (bytes). However, generally it's a good idea always to relate this basic unit to the amount of work performed as part of the unit given to Gauger.

For example, suppose a benchmark measures the execution time for 100 GET requests. In this case, it is better to choose a name "GET request performance" with unit "requests/second" (and log the execution time divided by 100) instead of the name "Time for 100 GET requests" with unit "seconds". The reason for this is it's quite possible the benchmark will be adjusted in the future—for example, to run 1,000 GET requests. If performance is logged as "requests/second", such a change would then not require any changes to the name of the tracked metric. As a result, the performance regression analysis can continue to track the metric in the same curve.

Additionally, Gauger allows different results to be compared by adding new metrics to existing plots. If the new metric uses the same unit as the old one, they will use the same *y*-axis; otherwise, the new one will be plotted against a second *y*-axis on the right side of the plot. This limits the number of units per plot to two. We recommend using the same units where applicable (for example, no mixing of "seconds" and "milliseconds") to make comparative analysis easier.

Related Projects

Gauger does not include for support for actually automatically running the benchmarks on various systems. However, this is not a drawback, because an excellent tool for that purpose exists in the form of Buildbot. Buildbot typically is used for regression testing and, hence, is by itself not suitable for identifying performance regressions. Nevertheless, Buildbot requires a similar network setup—that is, clients that run the tests connect to a public server. This makes it trivial to

combine a Buildbot setup with Gauger. Buildbot is used to execute regression and performance tests, and Gauger visualizes the development of performance metrics over time.

Another tool for monitoring performance is Munin. Like Gauger, Munin allows users to specify which performance measurements should be created. In contrast to Gauger where execution hosts push data to the server, the Munin server periodically pulls all participating systems for a performance score. As a result, NATed systems are not easily supported. Also, because Munin stores the data indexed by time and not revision number, and given that software performance may differ widely between different platforms, not all systems may have performance scores ready at fixed time intervals. Although Munin is not a good fit for performance regression analysis for developers, it likely is a better fit for system administrators who need to monitor system performance.

Conclusion

Gauger offers a lightweight and language-independent approach for integrating performance regression testing with existing development processes for projects using a wide range of version control systems. With Gauger, performance regressions are detected early, providing users with software that finally is improving consistently in both correctness and performance.■

Bart Polot is working on his PhD as a researcher at the Technische Universität München. His research interests include security, networking, routing and botnets.

Christian Grothoff is an Emmy-Noether research group leader at the Technische Universität München. His research interests include compilers, programming languages, software engineering, networking and security.

Resources

Gauger: **freshmeat.net/projects/gauger**

Buildbot: **buildbot.org**

Munin: **munin-monitoring.org**

man make

A PRIMER ON THE MAKE UTILITY

In the modern world of Integrated Development Environments, we forget what really goes into compiling a large code project. This article should be a refresher on (or teach for the first time) the basics of makefiles, the most underrated part of any code project.

Adrian Hannah

In a compiled language, the makefile is arguably the most important part of any programming project. To compile your project, you first have to compile each source file into an object file, which in turn needs to be linked with system libraries into the final executable file. Each command can have a considerable number of arguments added in. That's a lot of typing and a lot of potential for mistakes. The more source files you have, the more complex the compilation process becomes, unless you use makefiles. Most Linux users have at least a cursory knowledge of make and makefiles (because that's how we build software packages for our systems), but not much more than that. Most developers probably don't have too much in-depth experience with makefiles, because most Integrated Development Environments (IDEs) have the capability of managing makefiles for them. Although this is convenient most of the time, knowing more about how make works and what goes into makefiles can help you troubleshoot compilation errors down the road.

According to make's man page, "The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them." Essentially, make is used to determine efficiently (and without user error) which portions of the source code have been updated since the last compilation and recompile them. It can be used for more than just compiling programs. Because it isn't limited to any particular language, you can use it for anything you can come up with that relates to the modified date of a group of files.

Running make is a straightforward

IMPORTANT:

Command lines must be indented with tab characters; spaces cause funky errors. This has been a design flaw in make for decades. Empty lines must still have a tab character or else make will throw a fit.

process. The more convoluted portion of using make is constructing the makefile. The makefile is a file that consists of a series of rules that define the dependencies of your project. These rules govern the behavior of make during execution.

Listing 1. Example Makefile

```
CC=g++
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp hello.cpp factorial.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=hello

all: $(SOURCES) $(EXECUTABLE)


$(EXECUTABLE): $(OBJECTS)
        $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
        $(CC) $(CFLAGS) $< -o $@
```

Rules and Targets

Each rule in the makefile is an independent series of commands that are executed in order to build a target. Make does not necessarily run each rule in order. Make will run through the rules recursively, building each target in turn, based on modification. Rules are formatted like this:

```
target: dependency list ...

        commands

        ...
```

The target is typically the name of a file, but it can be a phony target (discussed later in this article). The dependency list is a space-separated list of files that designate whether the

THE BASICS:

- Comments start with a pound sign (#).
- Continuation of a line is denoted by a back slash (\).
- Lines containing equal signs (=) are variable definitions.
- Each command line typically is executed in a separate Bourne shell—that is, sh1.
- To execute more than one command line in the same shell, type them on the same line, separated by semicolons. Use a \ to continue the line if necessary.

target needs to be rebuilt. The commands can be any shell command, so long as the target is up to date at the end of them. It is imperative that you indent the commands with a tab character and not spaces. This is a design flaw in make that has yet to be fixed, and it will cause some strange and obscure errors should you use spaces instead of tabs in your makefile.

When make encounters a rule, it first checks the files listed in the dependency list to ensure that they haven't changed. If one of them has, make looks through the makefile for the rule containing that file as the target. This recursion continues until a rule is found where all the dependencies are unchanged or rebuilt (or have no further dependencies), and then make executes the listed commands for that rule before returning to the previous rule, and so on, until the root rule has been satisfied and its commands run.

You may use pattern-matching characters to describe dependencies in the dependency list or in commands, but they may not be used in the target.

Phony Targets

Phony targets (also called dummy or pseudo-targets) are not real files; they simply are aliases within the makefile. As I mentioned before, you can specify targets from the command line, and this is precisely what phony targets are used for. If you're familiar with the process of using make to build applications on your system, you're familiar with `make install` (which installs the application after compiling the source) or `make clean` (which cleans up the temporary files created while compiling the source). These are two examples of phony targets. Obviously, there are no "install" or "clean" files in the project; they're just aliases to a set of commands set aside to complete some task not dependent on the modification time of any particular file in the project. Here is an example of using a "clean" phony target:

```
clean:
        -rm *.o my_bin_file
```

Special Targets

Some special targets are built in to make. These special targets hold special meaning, and they modify the way make behaves during execution:

.PHONY — this target signifies which other targets are phony targets. If a target is listed as a dependency of .PHONY, the check to ensure that the target file was updated is not performed. This is useful if at any time your project actually produces a file named the same as a phony target; this check always will fail when executing your phony target.

.SUFFIXES — the dependency list of this target is a list of the established file suffixes for this project. This is helpful

when you are using suffix rules (discussed later in this article).

.DEFAULT — if you have a bunch of targets that use the same set of commands, you may consider using the .DEFAULT target. It is used to specify the commands to be executed when no rule is found for a target.

.PRECIOUS — all dependencies of the .PRECIOUS target are preserved should make be killed or interrupted.

.INTERMEDIATE — specifies which targets are intermediate, or temporary, files. Upon completion, make will delete all intermediate files before terminating.

.SECONDARY — this target is similar to .INTERMEDIATE, except that these files will not be deleted automatically upon completion. If no dependencies are specified, all files are considered secondary.

.SECONDEXPANSION — after the initial read-in phase, anything listed after this target will be expanded for a second time. So, for example:

```
.SECONDEXPANSION:
ONEVAR = onefile
TWOVAR = twofile
myfile: $(ONEVAR) $$(TWOVAR)
```

will expand to:

```
.SECONDEXPANSION:
ONEVAR = onefile
TWOVAR = twofile
myfile: onefile $(TWOVAR)
```

after the initial read-in phase, but because I specified .SECONDEXPANSION, it will expand everything following a second time:

```
.SECONDEXPANSION:
ONEVAR = onefile
TWOVAR = twofile
myfile: onefile twofile
```

I'm not going to elaborate on this here, because this is a rather complex subject and outside the scope of this article, but you can find all sorts of .SECONDEXPANSION goodness out there on the Internet and in the GNU manual.

.DELETE_ON_ERROR — this target will cause make to delete a target if it has changed and any of the associated commands exit with a nonzero status.

.IGNORE — if an error is encountered while building a target list as a dependency of .IGNORE, it is ignored. If there are no dependencies to .IGNORE, make will ignore errors for all targets.

.LOW_RESOLUTION_TIME — for some reason or another, if you have files that will have a low-resolution timestamp (missing the subsecond portion), this target allows you to designate those files. If a file is listed as a dependency of .LOW_RESOLUTION_TIME, make will compare times only to the nearest second between the target and its dependencies.

.SILENT — this is a legacy target that causes the command's output to

be suppressed. It is suggested that you use Command Echoing (discussed in the Command Special Characters section) or by using the `-s` flag on the command line.

.EXPORT_ALL_VARIABLES — tells make to export all variables to any child processes created.

.NOTPARALLEL — although make can run simultaneous jobs in order to complete a task faster, specifying this target in the makefile will force make to run serially.

.ONESHELL — by default, make will invoke a new shell for each command it runs. This target causes make to use one shell per rule.

.POSIX — with this target, make is forced to conform to POSIX standards while running.

Variables

In other versions of make, variables are called macros, but in the GNU version (which is the version you likely are using), they are referred to as variables, which I personally feel is a more appropriate title. Nomenclature aside, variables are a convenient way to store information that may be used multiple times throughout the makefile. It becomes abundantly clear the first time you write a makefile and then realize that you forgot a command flag for your compiler in all 58 rules you wrote. If I had used variables to designate my compiler flags, I'd have had to change it only once instead of 58 times. Lesson learned. Set these at the beginning of your makefile before any rules. Simply use:

```
VARNAME = information stored in the variable
```

to set the variable, and do use `$(VARNAME)` to invoke it throughout the makefile. Any shell variables that

PREDEFINED VARIABLES

- `$?` — evaluates to the list of components that are younger than the current target. Can be used only in description file command lines.
- `$@` — evaluates to the current target name. Can be used only in description file command lines.
- `$$@` — also evaluates to the current target name. However, it can be used only on dependency lines.
- `$<` — the name of the related file that caused the action (the precursor to the target). This is only for suffix rules.
- `$*` — the shared prefix of the target and dependent—only for suffix rules.

COMMON VARIABLES FOR C++ PROGRAMMING

- `CC` — the name of the compiler.
- `DEBUG` — the debugging flag. This is -g in both g++ and cxx. The purpose of the flag is to include debugging information into the executable, so that you can use utilities like gdb to debug the code.
- `LFLAGS` — the flags used in linking. As it turns out, you don't need any special flags for linking. The option listed is `-Wall`, which tells the compiler to print all warnings. But, that's fine. We can use that.
- `CFLAGS` — the flags used in compiling and creating object files. This includes both `-Wall` and `-c`. The `-c` option is needed to create object files—that is, .o files.

existed prior to calling make will exist within make as variables and, thus, are invoked the same way as variables. You can specify a variable from the command line as well. Simply add it to the end of your make command, and it will be used within the make execution.

If, at some point, you need to alter the data stored in a variable temporarily, there is a very simple way to substitute in this new data without overwriting the variable. It's done using the following format:

```
$(VARNAME:find=replace)
```

where `find` is the substring you are trying to find, and `replace` is the string to replace it with. So, for instance:

```
LETTERS = abcxyz xyzabc xyz
print:
echo $(LETTERS:xyz=def)
```

will produce the output `abcdef xyzabc def`.

Suffix Rules

In certain situations, you will find that the rules for a certain file type are identical except for the filename. For instance, a lot of times in a C project, you will see rules like this:

```
file.o: file.c
        cc -O -Wall file.c
```

because for every .c file, you need to make the intermediate .o file, so that the end binary then can be built. Suffix rules are a way of minimizing the amount of time you spend writing out rules and the number of rules in your makefile. In order to use suffix rules, you need to tell make which file suffixes are considered significant (suffix rules won't work unless the

suffix is defined this way), then write the generic rule for the suffixes. In the case described above, you would do this:

```
.SUFFIXES: .o .c

.c.o:
        cc -O -Wall $<
```

You may note that in the case of suffix rules, the dependency suffix goes before the target suffix, which is a reversal from the normal order in a makefile. You also will see that you use $< in the command, which evaluates to the .c filename associated with the .o file that triggered the rule. There are a couple predefined variables like this that are used exclusively for suffix rules:

- $< — evaluates to the component that is being used to make the target—that is, file.c.
- $* — evaluates to the filename part (without any suffix) of the component that is being used to make the target—that is, file.

Note that the $? variable cannot occur in suffix rules, but the $@ variable still will work.

Command Special Characters

Certain characters can be used in conjunction with commands to alter the behavior of make or the command. If you're familiar with shell scripting, you'll recognize that \ signifies a line continuation. That is to say, using \ means that the command isn't finished and continues on the next line. Nobody likes looking at a messy file, and using this character at the end of a line helps keep your makefile clean and pretty. If a rule has more than one command, use a semicolon to separate commands. You can start a command with a hyphen, and make will ignore any errors that occur from the command. If you want to suppress the output of a command during execution, start the command with an at sign (@).

Using these symbols will allow you to make a more usable and readable makefile.

Directives

Sometimes, you need more control over how the makefile is read and executed. Directives are designed exactly for that purpose.

From defining, overriding or exporting variables to importing other makefiles, these directives are what make a more robust makefile possible. The most useful of the directives are the conditional directives though.

Conditional directives allow you to define multiple versions of a command based on preexisting conditions. For

Conditional directives allow you to define multiple versions of a command based on preexisting conditions.

example, say you have a set of libraries you want included in your binary only if the compiler used is gcc:

```
libs_for_gcc = -lgnu
normal_libs =

foo: $(objects)
ifeq ($(CC),gcc)
        $(CC) -o foo $(objects) $(libs_for_gcc)
else
        $(CC) -o foo $(objects) $(normal_libs)
endif
```

In this example, you use `ifeq` to check if CC equals `gcc` and if it does, use the gcc libraries; otherwise, use the generic libraries.

This is just a small, basic sampling of the things you can do with make and makefiles. There are so many more complex and interesting things you can do, you just have to dig around to find them!■

Adrian Hannah has spent the past 15 years bashing keyboards to make computers do what he tells them. He currently is working as a system administrator for the federal government. He is a jack of all trades and a master of none. He spends all his waking hours on the *Linux Journal* IRC channel, on Twitter (@codemoney2841) and talking to random chat bots on the Internet.

Resources

GNU make comes with most Linux distributions by default, but it can be found on the main GNU FTP server: **ftp.gnu.org/gnu/make** (via HTTP) and **ftp.gnu.org/gnu/make** (via FTP). It also can be found on the GNU mirrors at **www.gnu.org/prep/ftp.html**.

Documentation for make is available on-line at **www.gnu.org/software/make/manual**, as is documentation for most GNU software. You also can find more information about make by running `info make` or `man make`, or by looking at /usr/doc/make/, /usr/local/doc/make/ or similar directories on your system. A brief summary is available by running `make --help`.

Qt4 Designer and Eclipse

You don't have to be a code guru to develop GUIs.

PJ RADCLIFFE

Qt4 Designer is one of the most powerful GUI builders on the market. If you look at the many Qt4 Designer tutorials, they are all very code-heavy, which really puts off new programmers. After some searching, I put together a few simple methods that make GUI development easy, even for novice C++ programmers, and for experts, it will speed up development.

This article shows how to use a GUI builder based on Qt4 Designer and Eclipse. Unlike other tutorials, this one is not code-heavy; instead, it describes several simple methods that will enable you to develop a GUI in a visual manner with a minimum of C++ code. If you are new to C++ and GUI building, this tutorial will make it possible for you to build a GUI. If you are an expert, these methods at least will help speed your development. The development tools must run under the KDE desktop, although the final program can be run on a GNOME desktop as well.

GUI interfaces are a key part of most operating systems, and with Linux, we are spoiled for choice. So many good tools are available; see the Wikipedia IDE comparison in the Resources section of this article for an excellent summary. I teach some 300 first- and second-year engineers about Linux programming including GUI programming. They need to use C++, and the options tend to be narrow. The two IDEs that look most applicable are Code Blocks and the combination of Eclipse CDT and Qt4 Designer. Qt4 Designer clearly is the most powerful GUI builder with the best documentation, and it's free if you are creating GPL or LGPL code.

Installing the Software

The simplest option is to obtain a live DVD with all the necessary tools installed. See **interestingbytes.wordpress.com** for a Mint-based live DVD with these tools plus many other development tools.

To install the software yourself, work through the following steps:

First, ensure you have the gcc compiler toolchain installed. From a command line, type `g++ --version`. If this fails, use your package manager to install gcc, g++ and libstdc.

Next, you need the GUI builder Qt4 Designer and the documentation system Qt4 Assistant. You can find them in your Linux repository and install them with your package manager. In the unlikely event that Qt4 is not available, try the Nokia Qt download site (see Resources).

You also need Eclipse with CDT (C/C++ Development Tools) installed. There are several options here:

- Install Eclipse CDT from your package manager (easiest option).
- If you already have Eclipse installed but not the C/C++ development tools, start Eclipse and select the menu option Help→Install New software. Select your main repository site and filter on "CDT". Select the CDT package and install.
- The latest version of Eclipse at the time of this writing is Helios, which may not be in your distribution's repository. It has a nice all-in-one package of Linux Tools that includes code coverage and profiling. The Eclipse Helios site has a download and installation instructions (see Resources). Note the wiki link from that page, which has some very useful tutorials.

Finally, you need the Qt4 integration plugins for Eclipse. The Nokia Web site has instructions and the download (see Resources).

Creating a GUI Project

Most Qt4 tutorials show how to write code to build a form. The combination of Qt4 Designer and Eclipse means you can do this in a purely visual manner, which is faster and easier. To build your own GUI, work through the following steps:

- Start Eclipse using the Linux Menu option Development→Eclipse. If any projects are open, select the Project Explorer in the left-hand panel, right-click on the project folder, then select Close Project.
- From the Eclipse menu, select File→New→Project.
- Find the Qt option, select the sub-option Qt GUI project, and then click Next.

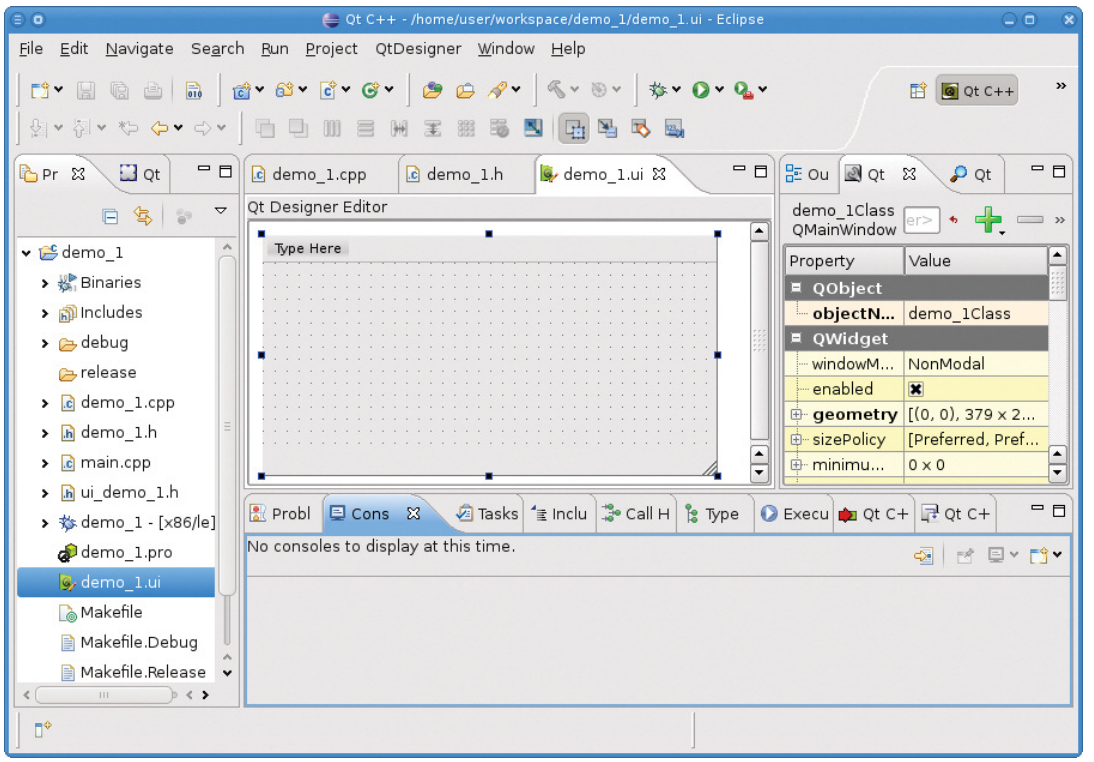


Figure 1. Empty Main Page

- On the next panel, provide a project name (let's use demo_1 for this example), and click Next (not Finish).

- The next panel will set up the class and filenames. You can change the name of the class where you place your code. The bottom list box labeled UI Type is the most important field. It determines the type of GUI you will create. A Widget is a basic panel with no special functions; a Qdialog provides simple interaction with a user, and a QmainWindow is a panel with menus and other features. For this example, select QmainWindow, then click Next.

- This panel allows you to include extra Qt modules—for example, a network interface or SQL interface. For this example, the extra modules aren't needed, so just click Finish.

- You will be asked to accept Qt perspectives; click yes.

You now should have a GUI project ready to add visual components. To see the appropriate files, as shown in Figure 1, expand the project in the Project Explorer panel to the left and double-click on demo_1.ui.

The compile and execution of the GUI you have just created may have a hiccup, especially the first time you run it. You should be able to select from the Eclipse menu File→Save All, Project→Build All, and then Run→Run. If Run→Run is grayed-out, try the following:

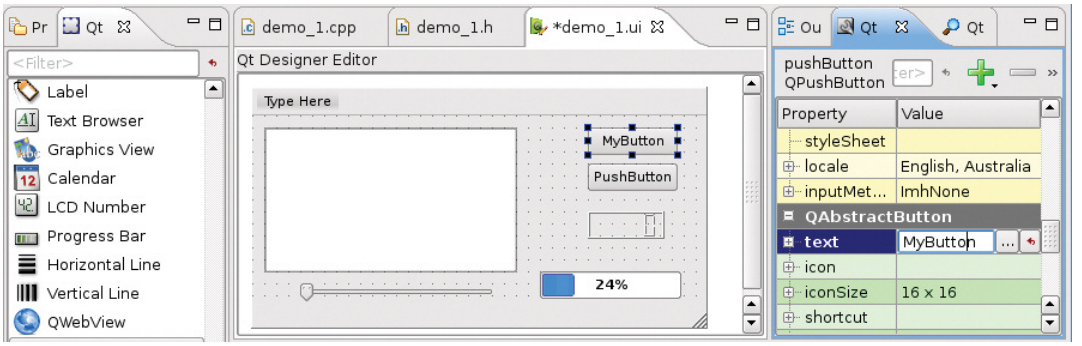


Figure 2. Visual Components

- If demo_1.cpp is not already in the editor window, then in the Project Explorer panel on the left, double-click demo_1.cpp to get the code in the editor window.

- Add a space anywhere, and then from the Eclipse menu, select File→Save All.

- Select Project→Build All, then Run→Run. If you are asked for a debugger configuration, choose gdb/mi.

- Your GUI now should appear. Close it by clicking the close icon on the top right.

Now, it's time to add visual components to your bare form. On the left-hand panel, select the Qt tab to see the visual components. Click and drag across two Push Buttons, an LCD Number, a Text Edit box, a Horizontal Slider and a Progress Bar to get a form similar to Figure 2.

You can see the properties of any visual component by clicking on a component and then examining the property editor in the right-hand panel. The properties are shown as a hierarchy with the parent class first and then the child classes. You may need to scroll down to find the property you want to change. In Figure 2, a button has been selected and the property editor shows its properties. The property called "text" has been changed to MyButton, which is not the name of the button, just the message displayed. It's worth spending some time looking at all the properties you can alter. Try also right-clicking on visual components to see what you can change. For example, right-click on the text edit box and select Change HTML. Type in some text, click OK, and that text is now displayed in the text edit box.

To create a menu for your form, click at the top left where it says "Type Here", and type "File". Your form now has a menu option "File".

Click on this again, and add a sub-option "Save". After finishing this tutorial, it's worth starting a new project and playing with a range of visual components, right-clicking on each and

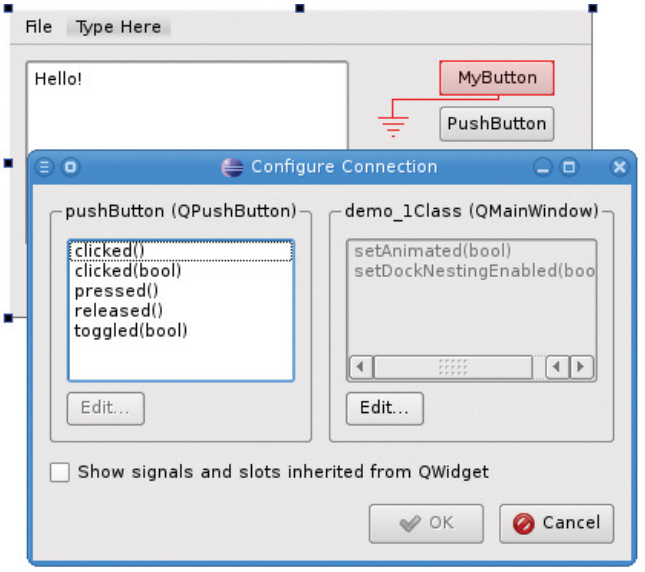


Figure 3. Wired Signal

changing the properties.

Qt Designer has the concept of signals and slots. A class can emit a signal when it has done something—for example, when a button is clicked. A class can accept a signal in a slot—for example, the LCD display has a slot to accept a new value. You can connect visual components directly together using signals. From the Eclipse menu, select Qt Designer→Editor Mode→Signals and Slots. Click on the slider, drag the resulting red line to the progress bar, and release the mouse. In the panel that pops up, select sliderMoved in the left box and setValue in the right box, then click OK. It's important to know this edit mode, because it's a quick way to discover all the signals and slots a visual component can support. To see what signals a button can generate, drag the mouse from the button to the form and

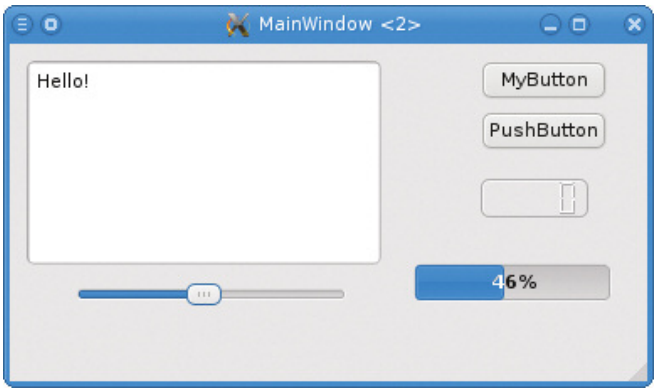


Figure 4. First Run

release the mouse. A window now pops up that names all the signals a push button can emit. Close this window by pressing cancel to delete the link. Now, try dragging from the form to the button to see all the slots the button can support. Try doing the same to the LCD display (see Resources for more details on slots and signals).

To run your GUI, first change the editor mode back to the normal widget view with Qt Designer→Editor Mode→Widgets. Then, compile and run using the Eclipse menu options File→Save All, Project→Build Project and Run→Run. If all goes well, you should get a result like what's shown in Figure 4. Try moving the slider, clicking on the buttons (which do nothing), and then close the window.

I don't have space to cover layout tools and spacers, which ensure that your layout expands and contracts as a user resizes the GUI window. See the large e-book from Blanchette and Summerfield, the short Layout Tutorial

and the Qt Overview listed in the Resources section for more information. Qt Assistant, discussed later in this article, also has an excellent tutorial.

Adding Your Own Code

Other Qt4 tutorials become code-heavy at this point and lose people who are not experienced C++ programmers. Here, I explain how to link your code to the GUI interface in a simple and pain-free manner with minimal C++.

Adding code to a GUI project is not like writing code for a command-line program. Your code must be placed in the class created for you, in the example demo_1.cpp, not a main() routine. You should alter only demo_1.cpp, demo_1.h or demo_1.ui. The other files are generated automatically and will be overwritten on the next build.

Your code can be started only when a GUI event happens and it is linked to your code, or when a timer you have created times out. Furthermore, your code must do its job quickly and return. If your code delays, the entire application freezes and stops responding to user stimuli. These factors require you to design your code differently from how you would design a command-line program. When you write a GUI program, plan for all of your code to respond to GUI events and timers. The GUI package handles the rest for you.

Most GUI systems have a library of useful classes, and it's very much worth learning how to use those classes, as they can save you a lot of time and provide features you could not implement otherwise. The library with Qt Designer is particularly powerful and worth mastering (more on that later).

Now, let's add the links between your own code and the visual components. Click the demo_1.h tab, and add the single #include line at the top of Listing 1. Next, add the declaration for timer, the public slots section and functions that are called when signals are generated. These slots are all of the form `on_objectName_action()`. The available "action" can be discovered from what

Listing 1. Adding Slots to demo_1.h

```
#include <QtGui>
public:
     demo_1(QWidget *parent = 0);
     ~demo_1();

private:
     Ui::demo_1Class ui;

//--- add the following lines.
     QTimer timer ;

public slots:
 void on_pushButton_clicked();
 void on_pushButton_2_clicked();
 void on_actionSave_triggered() ;
 void timer_tick() ;
};
```

Listing 2. Constructor with Timer Start

```
demo_1::demo_1(QWidget *parent)
     : QMainWindow(parent)
{  ui.setupUi(this);
 //--- add the following lines.
   timer = new QTimer(this);
   connect(timer, SIGNAL(timeout()), this, SLOT(timer_tick()));
   timer->start(1000);
}
```

Listing 3. First Push Button Code

```
void demo_1::on_pushButton_clicked()
{ //---  comment out or remove whatever is not needed.
         ui.pushButton->setText("New Name") ;
         ui.pushButton->setFont(QFont("Courier",10,QFont::Bold));
         QPalette palette;
         palette.setColor(ui.pushButton->backgroundRole(), Qt::red);
         palette.setColor(ui.pushButton->foregroundRole(),Qt::blue);
         ui.pushButton->setPalette(palette);
}
```

you did to create Figure 3. The first two slots are called when the buttons are clicked, the next when the form's menu item Save is selected, and the final one when your own timer based on Qtimer ticks. In total, you must add seven lines of code to the existing file.

Now, you can link the timer to the LCD display. In the demo_1.cpp file, modify the constructor to add the three lines of timer setup, as shown in Listing 2. First, the timer is created, and then the timer tick signal is connected to your own routine `timer_tick()`. Finally, the timer is set up to tick at 1,000 milliseconds (one second).

Listing 3 shows the code to react to the first push button. You need to add all the lines, as this member function is new. The first two lines of code change the text in the button and the font type. The next few lines change the color. This is well beyond what a simple application will need to do, but it shows a little more of what is possible.

Note that the properties of any visual component can be changed using the format `ui.ComponentName->member_function()`. As you begin to write more complex applications, you will need to find these member functions. The simplest way is to start a separate application called Qt Assistant, by going to Development→ Qt4 Assistant from the Linux menu. As shown in Figure 5, click the Index tab and enter the name of the visual element. A detailed description of the element will be displayed, including all useful member

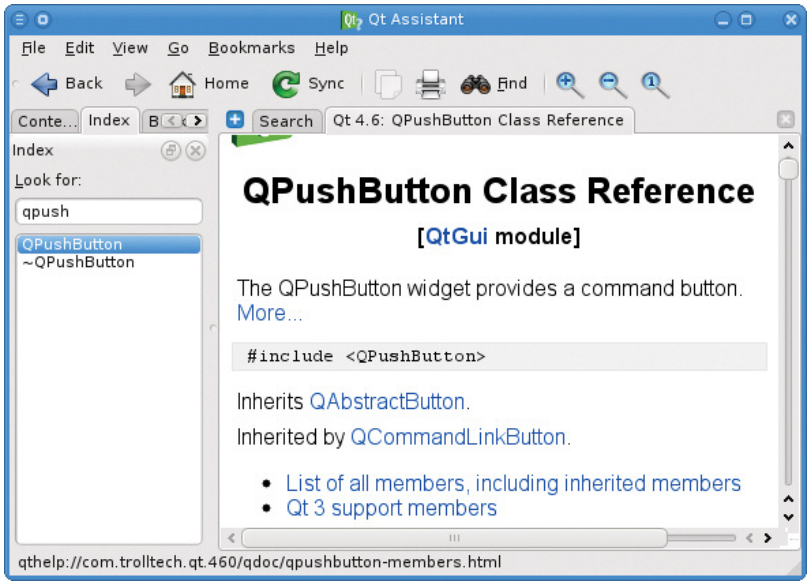


Figure 5. Qt Assistant

functions. If you are serious about mastering Qt Designer, it's worth spending some time working through the Qt Assistant documents. In particular, click the Contents tab, and open the Qt Designer Manual, which includes a wealth of useful tutorials and examples.

The second push button also must have its member function created (Listing 4). Again, the code shown here is more complex than a simple application needs to be, but it shows a very useful feature: calling a command-line program.

Let's work through the code. Note how the text box on-screen is cleared. The `ui.textEdit` gives you access to the visual component, and the member function `clear()` is called. `QbyteArray` is an advanced C++ string type object. The `QProcess` object enables you to run a command-line program. First, the working directory is set, and then the command started. If it returns within 300 milliseconds, the result is written into the textEdit box; otherwise, the command is terminated, and the default error message is printed. Note that the returned text from the command line may have nonprinting characters, such as line feeds. To solve this problem, look up the QbyteArray member function simplified() in Qt Assistant.

To respond to the menu on the

Listing 4. Second Push Button

```
void demo_1::on_pushButton_2_clicked()
{  ui.textEdit->clear() ;
   QByteArray command_line, work_dir("/tmp"),
result ;
   command_line = "ls /home" ;
   result="Nothing happened." ;
   QProcess shell(this) ;
   shell.setWorkingDirectory( work_dir) ;
   shell.start(command_line) ;
   if ( shell.waitForFinished(300)) //ms timeout
      result =  shell.readAllStandardOutput() ;
   ui.textEdit->append(result) ;
}
```

window, add:

```
void demo_1::on_actionSave_triggered()
{ ui.textEdit->clear() ;
  ui.textEdit->append("Menu item Save just triggered.") ;
}
```

To increment the LCD display every second, add:

```
void demo_1::timer_tick()
{ ui.lcdNumber->display( ui.lcdNumber->intValue() + 1) ;
}
```

Now, let's run your program. From the Eclipse menu, go to File→Save All, Project→Build Project and, finally, Run→Run. The LCD timer should be incrementing every second. Try clicking on the buttons and the form menu to make sure they perform as expected.

What Next?

What happens next is up to you. If you have trouble creating the example described here, you can download all of it from **www.pjradcliffe.wordpress.com**, then click on the "Other Useful Resources" page. Place the new directory inside the workspace directory (/home/user/workspace), then start Eclipse.

Browsing through Qt Assistant is a good way to discover the useful member functions of visual components and the other powerful classes, such as QbyteArray. Qt Assistant has many tutorials and examples, and the Web is a great source of information, as Qt has a vibrant user community. It won't take you long to make those visual components perform just as you wish.

Conclusion

After reading this article, you should be able to create a GUI using Eclipse and Qt4 Designer, mostly by visual manipulation of visual components plus a little C++ code. The example given here shows how to call your own code when visual components are activated (as with a button click), how to start your own code with a timer tick, and how to call command-line programs and read back their responses. You can do a lot with just those functions.■

Dr PJ Radcliffe is a senior lecturer at RMIT University in Australia. He once was an ardent Windows programmer, but then he discovered Linux, which he now teaches along with the control of hardware using Linux. If you are interested in these topics, see www.pjradcliffe.wordpress.com.

Resources

Wikipedia Article on IDE Comparison: **en.wikipedia.org/wiki/Comparison_of_integrated_development_environments**.

Qt4 download site from Nokia: **qt.nokia.com/downloads**.

Helios Eclipse Linux tools download and install instructions: **www.eclipse.org/linuxtools**.

Qt4 integration for Eclipse: **qt.nokia.com/developer/eclipse-integration**.

Good designer information (see the "signals and slots" link in particular): **doc.qt.nokia.com/4.6/designer-manual.html**.

A 700-page reference for Qt Designer from Blanchette and Summerfield (definitely worth obtaining although many examples take a code-heavy approach): search for "c-gui-programming-with-qt-4-2ndedition.pdf" to find a convenient upload source.

Layout basics: **thelins.se/learnqt/2009/05/qt-layouts-the-basics**.

Good overview of key Qt4 visual design features: **web.mit.edu/qt-dynamic/www/qt4-designer.html**.


Hacker Halted
USA 2011
Oct 21-27, 2011
Intercontinental Hotel. Miami, Florida
Its more than just a conference.
Its the Convergence of the Best at
a World Class Event
Jeremiah Grossman
Bruce Schneier
Philippe Courtot
George Kurtz
Charlie Miller
www.hackerhalted.com

GNU Awk 4.0: Teaching an Old Bird Some New Tricks

What's new and nifty in gawk 4.0, with a little history and background along the way. ARNOLD ROBBINS

GNU Awk (gawk) is one of those programs that has been available "since forever", which many people never think about. But, it's a standard and important part of just about every GNU/Linux distribution. In fact, it has been available since even before GCC!

During the past year and a half or so, gawk has undergone a quiet revolution, culminating in the release of gawk 4.0. Although not yet released at the time of this writing, work is in progress and moving forward. By the time you read this article, gawk 4.0 will be a fact and not just a promising code base in the Git repository.

A Little History

The awk language was developed by Al Aho, Peter J. Weinberger and Brian Kernighan, then at Bell Labs (hence the name A.W.K.). It first was released in 1978 with V7 UNIX. It offered the pattern-action programming paradigm, powerful regular expression matching, associative arrays, conventional operators and control structures, and a modest array of built-in numeric and string functions. It was only minimally documented. (So minimally, in fact, that I remember being terribly confused after reading the short paper on awk, and deciding to avoid it!) Nonetheless, the UNIX world accepted it and used it; true UNIX wizards were comfortable writing even large scripts in it.

Circa 1985, the authors started beefing up the language, adding user-defined functions, C-compatible operator precedence, more built-in functions, dynamic regular expressions and a few other minor features. More important though, they then proceeded to write a book about the new version

of awk (*The AWK Programming Language*), which was published in late 1987. This version became available to the world with the UNIX System V Release 3.2.

I bought the book, figuring that now was my chance to learn awk. It was (and remains) a great book. Having an interest in programming languages and an interest in contributing to the world at large, I decided to see whether the GNU project had a version of awk. Indeed, it did, but it implemented only old awk (and poorly, at that). Being single at the time, I decided to get involved and see if I could work to make gawk compatible with new awk. (And, thus, the course of history changed, forever.)

As early as 1988, the GNU developers were corresponding with Brian Kernighan and other awk implementers to make sure that the awk semantics were consistent across implementations. System V Release 4, in 1989, brought a few new features for new awk (the `-v` option, the `ENVIRON` array, the `tolower()` and `toupper()` built-in functions) and the first POSIX standard (circa 1992) introduced the CONVFMT variable.

Starting in December 1993, Brian Kernighan was able to release the code to new awk; it continues to be available (see Resources) and sees minor bug fixes from time to time.

GNU Awk

GNU Awk was first written around 1986 by Jay Rubin and Paul Finlason, with some help from Richard Stallman. It barely implemented the original awk language, was buggy and not particularly fast. It worked by building a parse tree representation of the program and then recursively evaluating the parse tree for each input record.

When I got involved in late 1987, David Trueman already had volunteered to upgrade it to new awk, and I joined the effort, contributing code fixes and doing serious work on the documentation. We worked together until around 1994, when I became the sole maintainer.

Along the way, gawk acquired full compliance with new awk, including POSIX, and it improved in code quality, speed and new features. Throughout the course of more than 20 years though, the basic design remained the same: build the parse tree and recursively evaluate it for each input record.

In 2003, out of the blue, a gentleman named John Haque contacted me. He had rewritten the gawk internals to use a byte-code interpreter and provided an awk-level debugger for awk programs. This was a startling innovation. I worked with him to get his version to the point where it was stable and passed the test suite, but I

The most significant new feature is that the gawk internals have been completely redone.

did not integrate his changes, because they were major, and I wanted to understand them better.

Bad move: John disappeared in early 2004, and the code languished, unused. Finally, in fall 2009, I got a volunteer (Stephen Davies) to start bringing the last version of the byte-code gawk that I had into the present. He and I had things working, pretty much, and I even announced a test release to the world.

Again, out of the blue, John resurfaced in early 2010 and joined the effort to make the byte-code gawk viable. This moved things into high gear, and we made a lot of progress. As I write this, the byte-code version has been merged with my "new features" branch of the code. This is the basis for gawk 4.0.

If you don't yet have gawk 4.0, see Resources for information on where to download the source and how to build it; building from source is very easy.

New Stuff in gawk 4.0

With all the background out of the way, let's look at the cool stuff. Due to space considerations, this is just a quick tour; see the documentation (listed in Resources) for details.

New Internals

The most significant new feature is that the gawk internals have been completely redone. The parser now builds a linked list of "instructions". Each instruction contains a code indicating what it is and a few members with needed information, such as the next instruction and which instruction to jump to if a jump is needed. This list then is interpreted for each record by a big switch statement running inside a for loop that traverses the list. Data for operations are pushed and popped off a runtime stack.

This implementation performs no worse than the original recursive evaluator, and in many cases, it performs better. But what's really cool is that John added an awk-level *debugger*!

Since 1978 when awk was first introduced into the world, the only debugging tool was the `print` statement. Now, gawk has a full debugger, with breakpoints, watchpoints, stepping by statement or instruction, the ability to step into and out of functions, and

many other features.

The debugger is a separately compiled program named dgawk. It is a line-oriented debugger modeled after GDB (the GNU Debugger). If you're familiar with GDB, it will be very easy to learn the gawk debugger. In addition, the debugger is fully documented in the gawk.texi file in the gawk distribution.

New Language-Level Features

At the language level, there are several new features.

1. gawk now provides a built-in file inclusion mechanism. Lines that begin with `@include` and have a filename in double quotes cause gawk to include that file, using the same path searching mechanism as the `-f` option. Nested includes are supported, and gawk will not include the same file twice. This effectively obsoletes the `igawk` script that has come with gawk for many years.

2. New patterns named `BEGINFILE` and `ENDFILE` provide "hooks" into gawk's automatic "read a record and process it" loop. The action for `BEGINFILE` is called before the first record is read from each input file. Normally, when a file cannot be opened, gawk exits with a fatal error (such as if you provide a directory on the command line). When a program has a `BEGINFILE` pattern, instead, gawk sets the `ERRNO` variable to a string indicating the problem, so that you can tell if the file is problematic. If it is, use the `nextfile` keyword to just skip it. `ENDFILE` actions let you do easy per-file cleanup actions.

3. You now can call a function indirectly. By setting a variable to the name of the function you wish to call and using special syntax, gawk will "indirect" through the variable and call the desired function:

```
function f1(a, b) { .... }
function f2(c, d) { .... }

{ fun = "f1";    @fun(2, 3)    # calls f1()
  fun = "f2";    @fun(4, 5) }  # calls f2()
```

4. gawk now sports true multidimensional arrays! Regular awk simulates multidimensional arrays (a[x, y]) using string concatenation of the index values. gawk now provides multidimensional arrays (a[x][y]) but does not require that arrays be rectangular (as in C or other compiled languages). Code like this is valid:

```
a[1] = 1
a[2][1] = 21
```

It is up to the programmer to track the type stored at any given index: scalar or array. Subarrays can be passed

to functions, as long as the function knows what to expect.

5. The `switch/case` statement is enabled by default. gawk has had `switch/case` for a long time, but it had to be enabled at build time, and the default was not to do so; now it's enabled automatically.

6. gawk now supports defining fields based on field content, instead of based on the separators between fields. A new variable, `FPAT`, is used. When you assign a string containing a regular expression to `FPAT`, gawk begins splitting fields such that each field is the text that matched `FPAT`. (Normal field splitting is based on the text in between fields matching the regular expression in `FS`.) This is useful for many kinds of data where `FS`-based matching just doesn't work.

The new `patsplit()` built-in function provides access to this functionality for strings besides the input record. It is the analogue of awk's regular `split()` function. Additionally, `patsplit()` lets you capture the text of the separators between fields.

7. Standard awk provides only one-way pipelines, either to or from another process. gawk provides a notation for opening a two-way pipeline to a co-process. gawk uses the same notation with special, internally recognized file-names, to provide TCP/IP communication over sockets. This feature has been available for a long time.

gawk 4.0 enhances the networking by providing explicit filenames to indicate IPv4 or IPv6 connections. Filenames are of the form /inet4/*protocol*/*local-portt*/*remote-host*/*remote-port* or /inet6/*protocol*/*local-port*/*remote-host*/*remote-port*. Plain /inet/*protocol*/*local-port*/*remote-host*/*remote-port* is what gawk supplied up to now and continues to be supported: it now means "use the system default". Most likely, this will continue to be IPv4 for many years.

8. gawk now provides a short (single-letter) option for every long option that it has. This finally makes it possible to use almost every feature from a `!#` script. It does somewhat bloat the manual page. (gawk has too many options, but that's a different problem; nonetheless, I did remove a few redundant long options.)

9. Interval expressions now are available by default. An interval expression is an enhanced regular expression syntax, such as `(foo|bar){2,4}`, which matches anywhere from two to four occurrences of either foo or bar. The part between the curly braces is the interval expression. POSIX added them to awk many years ago for compatibility with `egrep`'s regular expressions. But most awks didn't implement them. For historical compatibility, gawk's default was to

disable them, unless running in POSIX mode. Today, compatibility with POSIX has gained enough importance for enough users that interval expressions now are available by default.

10. Finally, for this release, the code has been reviewed and cleaned up. gawk now requires a full C 89 environment to compile and run. It will not work with K&R compilers or if __STDC__ is defined but less than 1. The code for many obsolete and unsupported systems has been removed completely. This slightly decreases the size of the distribution, but mainly it reduces useless clutter in the source. The documentation also has been reviewed and cleaned up.

Source Code Management

For many years, I was the only one with access to gawk source while it was being worked on. Circa 2006, I made both the stable and development versions available via CVS from savannah.gnu.org. This was a good move; it gave the user community access to all my bug fixes and to my development code base.

In late 2010, I moved to git. I am

expecting greater productivity from using git and better ease of use for the user community. And, it's nice to be using 21st-century tools.

Future Work

Some further interesting development remains to be done.

1. The XMLGawk Project (see Resources) is a fork of gawk based on 3.1.6 that provides better facilities for loading dynamic extensions and several very interesting extensions to go with those features. These should be merged into the main gawk code base and distribution, respectively.

2. Although gawk has had the ability to load extensions dynamically for many years, the API has not been stable or easy to use. I have designed an API for C functions that can be called from an awk program that is considerably better, but I have not implemented it yet. This should be done.

3. Currently, the gawk distribution builds three separate executables: regular gawk, pgawk (for profiling awk programs) and dgawk for debugging them. The new internals enable the possibility of making just one executable that could perform all three functions (based on command-line options). This should simplify the build process and definitely will reduce the total installation "footprint".

4. The documentation could use further cleanup. Some of the examples cause the documentation to show its age. (Who uses dial-up BBS systems anymore?)

Acknowledgements

Thanks to Brian Kernighan, Stephen Davies and John Haque for reviewing this article.■

Arnold Robbins is a programmer, technical author, husband and father. A native of Atlanta, Georgia, he and his family have been living in Israel since 1997, where he now works writing software for a very large semiconductor manufacturing company. He has been involved with GNU Awk since 1987(!). In his non-copious spare time, he maintains gawk and its documentation, among other activities. Arnold is also the author or co-author of a number of UNIX- and Linux-related books from O'Reilly and Prentice Hall, which he hopes that all readers of this article will now run out and buy. For more information, see www.skeeve.com.

Resources

Gawk Home Page at the FSF: **www.gnu.org/software/gawk**

Gawk Project Home Page at Savannah, with Links and Instructions for Using Git: **savannah.gnu.org/projects/gawk**

Gawk Download Directory: **ftp.gnu.org/gnu/gawk**

Gawk Documentation: **www.gnu.org/software/gawk/manual**

Installation Instructions: **www.gnu.org/software/gawk/manual/html_node/Installation.html#Installation**

Brian Kernighan's "one true awk": **www.cs.princeton.edu/~bwk/btl.mirror**

The XMLGawk Download Page: **sourceforge.net/projects/xmlgawk**

AUTOMATION BOOTCAMP!

cPanel Conference '11

OCTOBER 10 - 12, 2011

SHERATON HOTEL, AUSTIN TX

Learn strategies and tactics to automate your operations, free up resources, and conquer the opposition!

400	35	30	02	02	03
GUESTS	EXHIBITORS	LEARNING SESSIONS	LIGHTNING TALKS	MIND-BLOWING KEYNOTES	TEXAS-STYLE BASHES

REGISTER NOW BEFORE IT'S SOLD OUT!

WWW.BOOTCAMP.CPANEL.NET

BROUGHT TO YOU BY cPanel

WaveMaker: It's Like...RAD!

Database stuck in Windows? Take RADical action and try WaveMaker Community edition. DONALD EMMACK

Rapid application development (RAD) is a trendy data-processing idea to shorten application software development time. In the truest sense, RAD should include user-friendly development tools and seamless access between end users and actual development software. Small-business users lust for software applications that fit this description. Commercial tools, like Microsoft Access and AlphaSoftware, share great success in the business community.

Both of these applications give small-business users and IT departments a simple way to develop applications for internal use quickly, with little effort. Usually, a small Web-based application can be created in only a day or two. Rapid application development combined with a treasure trove of professionally presented training videos and other learning resources keep these products in the forefront of technology and embedded in the business world.

Thus, a product like AlphaSoftware frequently is the cornerstone tool for small-business users to deploy Web tools for on-the-fly needs. Although some of these homespun software applications are downright ugly (from an aesthetic and design standpoint), nonetheless, they are critical business tools. In fact, their importance is so profound that some small businesses will not even consider a Linux environment because few tools exist that closely match the feature set of AlphaSoftware or similar commercial products.

Certainly, good arguments exist for the robust programming tools Linux offers. Ruby gains much attention as a powerhouse tool, although Ruby is a "foreign language" to most small-business owners. So, traditional open-source tools seem to lack the absolute simplicity necessary for acceptance by many business users. Further, Linux coding tools are largely unfamiliar, which reduces the pool of people who are able to assist businesses develop

new applications. Alternatively, consider how easy it is to find a local IT guy who can create an Access database with full form control in only a few days!

Now, WaveMaker provides a strong Linux contender, properly positioned to battle for the interest of the business user. WaveMaker clearly targets a user audience with quick Web application development with a minimal learning curve. The product is available in both a community edition and an enhanced beta version poised for more commercial situations (**wavemaker.com/downloads**).

Meet WaveMaker

From a marketing perspective, WaveMaker is a robust tool for rapid application development of Web-based applications. You can peruse the WaveMaker site (**www.wavemaker.com**) for complete details of its promotional position. From a small-business viewpoint, WaveMaker meets the needs and requirements for rapid application development, ease of use and the ability to work across multiple platforms—including Linux.

The two editions of WaveMaker appear separated by the need to integrate with mainline applications. WaveMaker's community edition is limited to single-instance use, while the full edition enables unlimited users and interaction with commercial databases like Oracle, IBM DB2 or Microsoft SQL Server. Conversely, the community version supports the use of MySQL, PostgreSQL or HSQLDB. These two alternatives give small business users enough options for unique operating environments.

The Challenge

WaveMaker seems well suited to compete for Linux business-user interest. A tough sell, most business owners need a real-life example before they consider open-source systems. Thus, this article intends to demonstrate how WaveMaker could migrate a legacy database from the proprietary software world into the community edition of WaveMaker. This simple test is only a baseline, but it depicts WaveMaker's potential. In short, I show how WaveMaker can measure up to the tall demands of the business world and also re-affirm Linux as a viable choice for daily operations.

The example test here includes a sample business database originally developed with AlphaSoftware. The parameters of this test limit the environment to a small local Web application with only one table. It also demonstrates data export in a common format (comma-separated or Excel) file and MySQL data import. Plus, MySQL becomes the platform for the new application development.

This process looks for potential issues that may arise when users cull data from a commercial application.

In summary, the test steps are as follows:

1. Prepare the data in a common format.
2. Install WaveMaker community edition.
3. Use WaveMaker to auto-create a Web form for data entry and update.

Server and WaveMaker Preparation

For novice users, WaveMaker's Web site fully describes how to prepare your operating environment to support the tool. The one key decision point is whether to use the default HSQLDB database. As part of installation, WaveMaker includes the components necessary to use HSQLDB. In this test, let's use the popular MySQL as the database engine for the application.

First, let's use a Debian distribution to install MySQL. Let's also use phpMyAdmin to assist with database creation and data manipulation. If you want to follow along, check your distribution's instructions to install MySQL and phpMyAdmin, and make sure they are running before you continue. Next, download WaveMaker community edition and follow the installation instructions here: **dev.wavemaker.com/wiki/bin/wmdoc_6.3/Install**.

Preparation for the Test

After installation, let's plan a simple data export process to move into the new environment. My research discovered that AlphaSoftware includes a nifty export routine to send the data from its own database to other formats. I tested the export process with both text and Microsoft Excel formats successfully. To assist WaveMaker with field generation, I included field names during export. With the sample data in Excel format, use phpMyAdmin through the browser at http://localhost/phpmyadmin (or similar) to manage the MySQL experience.

To minimize errors with data import, a database and table were created *in advance*. In Figure 1, a new database called "linuxjournal" with a table named "sheet" already is in place. Next, let's use the import tool to populate the new "linuxjournal" database. Afterward, examine the database's structure to make sure the primary key value and index are appropriate for the test environment (Figure 2). In this example, I avoided foreign keys to minimize opportunities for errors. WaveMaker includes functions for database creation; however, the interface is not as familiar as phpMyAdmin. Further, using WaveMaker to build your data structure is an advanced topic. Therefore, the simpler approach is to stay with phpMyAdmin.

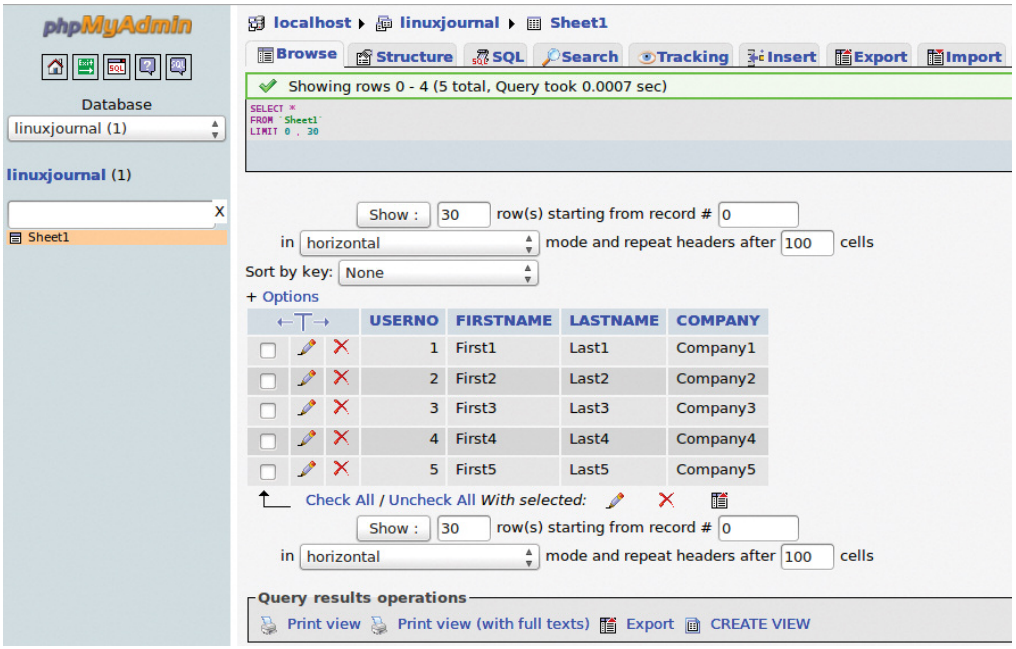


Figure 1. Once logged in to phpMyAdmin, create a test database called "linuxjournal", and import test data with field names.

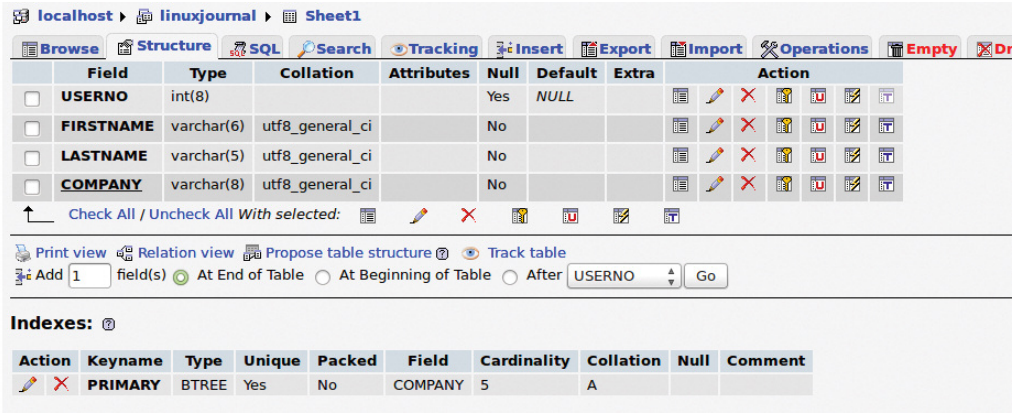


Figure 2. Verify that the indexes and keys are correct for the imported data.

WaveMaker Live

I used Linux Mint (a Debian distribution) for this test. After installing WaveMaker, the icon for startup did not appear on the desktop or in the application menu. To work around this problem, navigate to /opt/{wavemaker sub-directory}/bin/. Then, execute `./wavemaker.sh`, and observe the new WaveMaker application presented on your desktop as shown in Figure 3. From here, press Start to begin the process, and the default browser

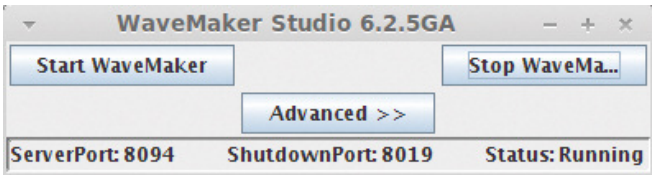


Figure 3. To the point, WaveMaker responds with a minimal window allowing you to start or stop the application.

Figure 4. At first, WaveMaker gives you the opportunity to create projects, open existing ones or just work through a tutorial.

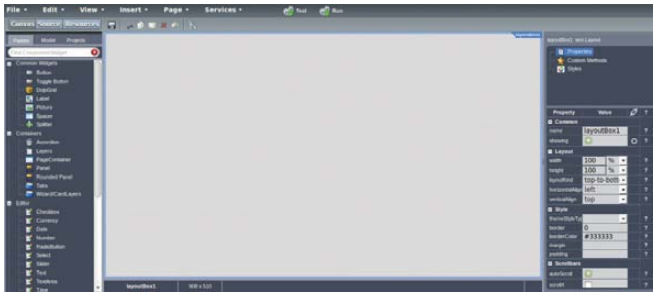


Figure 5. A clean slate or "Palette" is the home screen of WaveMaker.

opens to the application's home screen (Figure 4).

From this point, click Create a New Project, give it a name, and then WaveMaker drafts the workspace for the application within the browser (Figure 5). This is the primary control center for development. While the look and feel may seem unfamiliar, WaveMaker reduces training time through intuitive command labels and software component tree organization.

Establishing Database Connection

Once WaveMaker is up and running, it must establish database connectivity to continue. Next, select Services and then Database Services to define the connection to "linuxjournal". Now, WaveMaker needs the right information to connect to the database (Figure 6). With MySQL, the first step is to change the initial drop-down list from the default HSQLDB to MySQL. Next, input the database userid, password and exact database name in the fields provided. Click Test Connection at the bottom of the window to ensure connection. Once successful, select Import to continue the data import process, and WaveMaker responds with the elements of the MySQL database in a new window (Figure 7). Stop here and evaluate your structure. It's important to

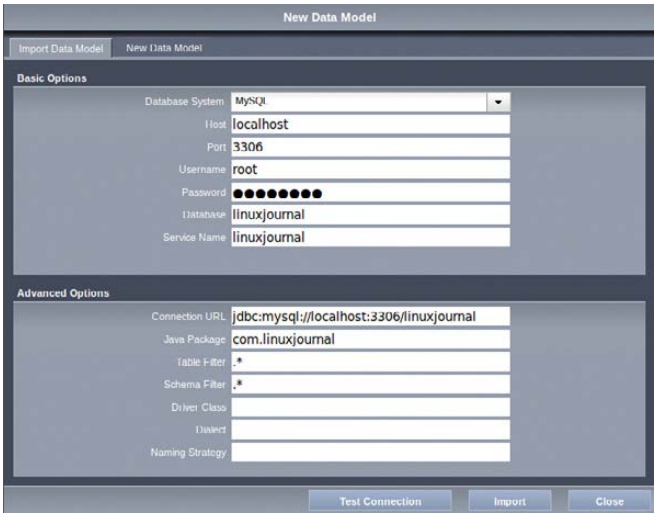


Figure 6. WaveMaker's Template to Establish Database Connectivity

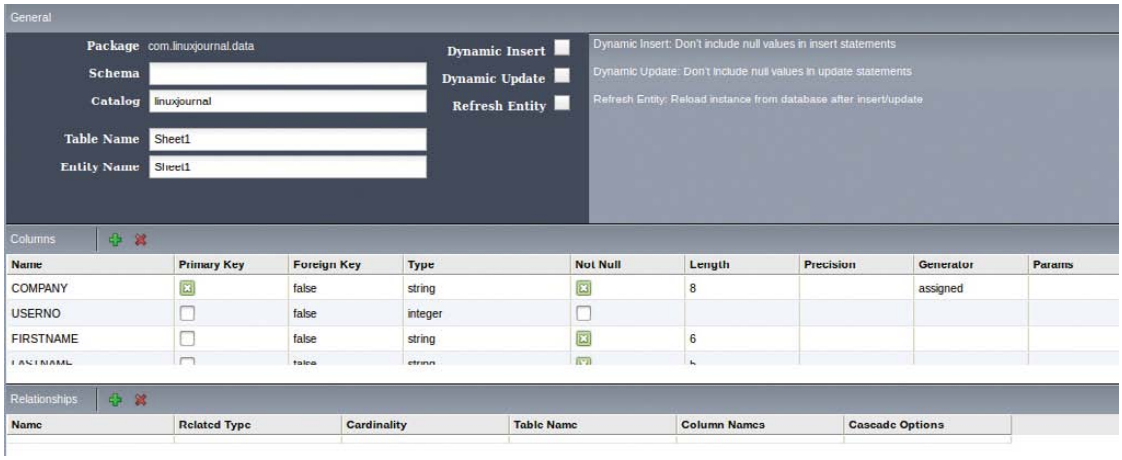


Figure 7. After import, your database structure is displayed.

validate keys and indexes. Also note the field names of your data to ensure they are listed properly.

Application Design

With a predominantly GUI interface, WaveMaker now is ready to create a Web-based application. A pivotal benefit for the small-business user, this interface relies on simple drag and drop for many operations. Automatic processes assist with form creation, data placement and manipulation. Plus, error correction is quite simple and capitalizes on the familiar Edit and Undo routine whenever you change your mind!

First, click on Canvas and Palette as shown in Figure 8. The link to the "linuxjournal" database previously created now is available in WaveMaker's tree view. Now, let's create a simple entry and update application—in one step. With the mouse, drag the database onto the blank area of the working screen. WaveMaker automagically populates the working area of the Canvas with the sample data from "linuxjournal". It's remarkable to watch WaveMaker create editable (bold) fields for the data on the bottom of the Canvas.

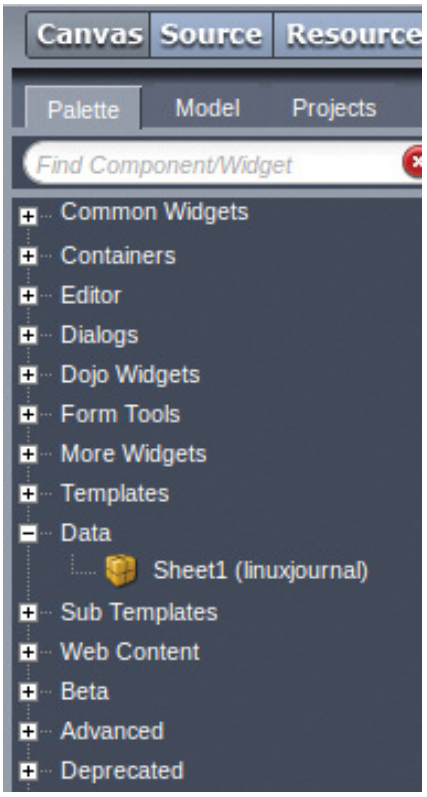


Figure 8. Navigating to the Location of Your Imported Table (Called "Sheet")

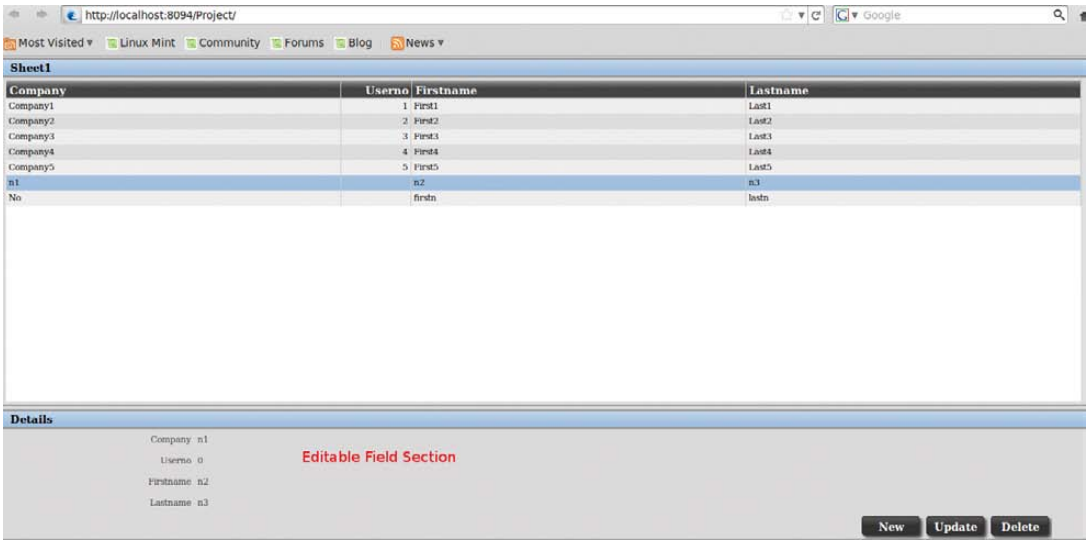


Figure 9. After drag and drop, WaveMaker creates the list of data on the top of the screen, and the editable fields are listed below. Red text was added by the author.

Figure 9 shows the results.

In essence, data from the MySQL table is displayed on top, and a detail section (with editable fields) is created below. At the same instant, WaveMaker creates three buttons for data manipulation: New, Update and Delete (Figure 10). The current view displays the skeleton of the application, complete and test-ready. Select Test from the top window, and a new browser window with the application displays in test mode. Choose Run, and WaveMaker saves and then runs the application in a new browser window, as shown in Figure 11.

Figure 10. In the bottom right, WaveMaker automatically provides New, Update and Delete buttons for the application.

Now, work with this mini-program to verify the success of the transition, exit the application and then re-visit phpMyAdmin to see the elements properly added, changed and deleted.

Pinch Yourself

WaveMaker just completed this test migration in approximately 15–20 minutes! Certainly more complex data migrations will require a concerted effort to ensure proper foreign keys and precise index setup. Yet, the viability of this principle is proven for a small application migration.

In this test, WaveMaker performs quite well. Building the application and running it on the same machine is a bit sluggish, but this will change substantially with respect to hardware specifications and overall configuration.

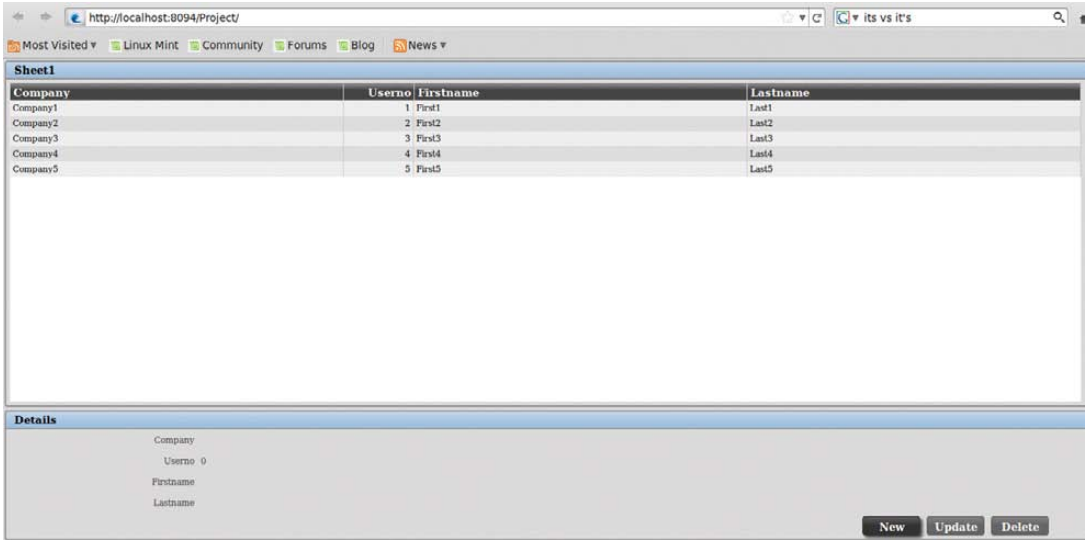


Figure 11. Live Operation of the Sample Data Migration in Less Than 20 Minutes

No Parking!

This test purposely excluded necessary elements to make many applications really suitable for business. Nevertheless, adding buttons, screens and tabs is very straightforward. For example, user login logos and multiple tabbed displays often are staples of even the most rudimentary business applications. WaveMaker includes many nicely packaged tools that enable users to drag and drop additional features immediately into the live Palette.

WaveMaker builds these other essential elements with a host of "widgets" along the application's left panel (Figure 12). Users take advantage of this noncode environment and simply drag and drop new features into the Palette. For example, a handy calendar (Figure 13) or simple page navigation

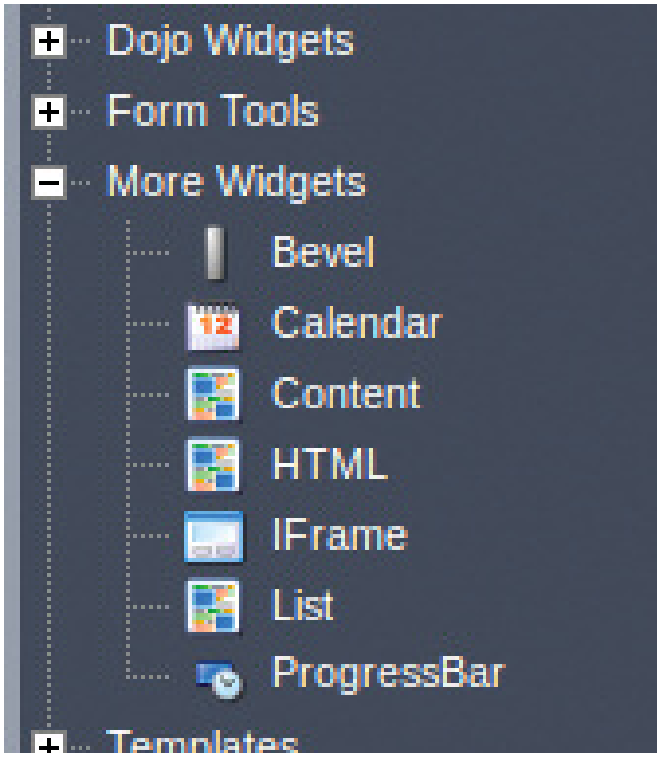


Figure 12. A Sample of Widgets Included with WaveMaker

(Figure 14) require no direct code. Plus, login security and roles are handled

Figure 13. The Included Calendar Widget

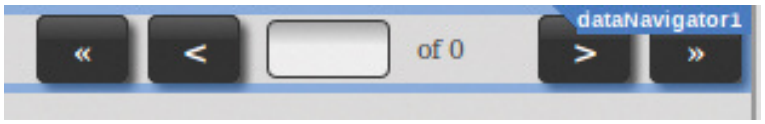


Figure 14. Simple Creation of Data Navigation Buttons

efficiently in the Services and Security tabs of the application—once again, a simple user interface.

The Takeaway

For business users, WaveMaker excites the potential of open-source possibilities. In recent years, the strongholds of proprietary operating systems usually included finance and point of sale. Because few comparable business-class applications exist, the impetus to move into the open-source world is sometimes difficult. It's also clear that mainstream business users may consider Linux when the core applications of their business can be migrated with ease.

Often overlooked, small database development served as a type of commercial software lynchpin. Thus, business operators are often locked in the proprietary realm. As a former AlphaSoftware user, I too see the impediments to change. Plus, applications like AlphaSoftware give developers a standardized tool to develop and deploy executable software in addition to Web applications. This is a market segment not directly marketed by WaveMaker. As with most software, familiarity of interface and technical support are all weighty factors in the decision.

WaveMaker seems like a bold step and immediately gathers the attention of business users. It is a true competitor to the commercial Web application development world and deserves attention by the business community. VMware's recently announced acquisition of WaveMaker (**dev.wavemaker.com/blog/2011/03/08/wavemaker-springs-to-vmware**) gives this application a boost in recognition and in business users' confidence in its future stability. Go give WaveMaker a try!■

Don Emmack, a Change Management consultant, assists government and business users with infrastructure and operational issues. A former Sr. Vice President for an international consulting firm, his work includes domestic and international clientele. Don is an early adopter of new technology, yet remains hopelessly addicted to his 1980s-era fax machine!

DebConf11

http://debconf11.debconf.org

BANJA LUKA

REPUBLIKA SRPSKA
BOSNIA & HERZEGOVINA

DebConf, Debian's annual developer conference, is an event filled with talks, coding parties, and discussions – all highly technical in nature. This year's DebConf took place in Banja Luka, Republika Srpska, Bosnia and Herzegovina, from 24th to 30th July 2011. Immediately before DebConf we held DebCamp, a week of intensive workshops where teams responsible for specific areas in Debian, who usually communicate online, could meet face-to-face and work together outside the scheduled events of the main conference.

Video presentations from Banja Luka are now available from the DebConf website at http://debconf11.debconf.org/

Next year's conference will be held in Managua, Nicaragua. Whether or not you have attended previous DebConfs, if you're interested in Debian development, we look forward to seeing you there in 2012!

Thanks to our sponsors for making DebConf11 possible!

jEdit: a Text Editor and More

Getting started with this extensible GUI text editor.

ADRIAN KLAVER

jEdit is a cross-platform text editor written in Java. The current stable version at time of this writing is 4.3.2, and it's available at **jedit.org**. Besides the cross-platform capabilities, jEdit offers other features, such as a sophisticated plugin system, syntax highlighting for 130 languages, a built-in macro language and extensive encoding support. I wrote this article using jEdit, and I demonstrate some of its features here, especially some of the plugins I have found useful.

Before I start, jEdit is a GUI text editor of some heft. It is not a replacement for using vi on the command line to edit a configuration file on a remote server. It does serve well in handling many files simultaneously with visual feedback and with the benefits of a GUI interface. To put it another way, I use vi or jEdit depending on the need.

Installation is fairly easy; just go to the download page and grab the installer jar. Be sure to check out the compatibility link if you have a non-Sun (Oracle) or Apple version of Java. From personal experience, I have not had success running jEdit on gcj. Assuming you have a compatible version of Java, use the following to install:

```
java -jar jedit4.3.2install.jar
```

This launches an installer program that guides you through the process. For the sake of reference, jEdit keeps its configuration files, on Linux anyway, in ~/.jedit/. I mention this because I keep that directory synced between my laptop and my desktop machines. As a result, I have a consistent working environment between the two.

At its heart, jEdit is a just a text editor, although it's a text editor with a lot of options. You can make these options global or apply them on a per-buffer basis. You can reach the options via the Utilities menu item. The global options stick between editing sessions, but the

buffer options do not, unless you use the buffer-local method. This consists of embedding colon-separated hints to jEdit in the file. jEdit checks the first or last ten lines for these hints. As an example, to specify an indentation of 2, use spaces for tabs and "hard" wrap, the embedded hints would be:

```
:identSize=2:noTabs=True:wrap=hard:
```

jEdit checks anywhere in those lines, so you can place the hints behind comment symbols.

Also note that jEdit supports mode-specific settings, where a mode is a file type, such as Python (*.py), C (*.c), HTML (*.html) and so on. The various modes come with default settings, but they can be overridden. One of the key benefits is that the mode system pulls in file-type-specific syntax highlighting. Other options are available for the editor's layout. As you can see in the screenshots for this article, I tend to run jEdit with two buffers open, split vertically and with line-numbering enabled. The ability to look at the beginning and end of file at the same time, especially source code, is invaluable.

You can use jEdit in a great number of ways. Watching me enter text, although it has its moments, is not terribly inspiring. So to keep things interesting, here I demonstrate some of the plugins I have found useful. Plugins are code that scratches an itch. The base jEdit program does a lot, but it does not cover the universe that is text editing, or other chores for that matter.

jEdit has a macro system (not covered in this article), so you can whip up your own solutions to problems or scope out the plugins available and not re-invent the wheel. So, before getting into the plugins themselves, here's an overlook at the plugin system itself. They can be found at **plugins.jedit.org** or via the Plugins item on the menu bar. Click on the Plugin Manager item and then the Install tab for a list of available plugins. Clicking on an item shows a description at the bottom of the page. Check the box of any plug-in(s) you want to install, and then click Install. If the plugin has dependencies, they also will be installed.

So, where to start with the plugins? Let's go from less-involved to more-involved, beginning with one suggested to me by a member of the Bellingham Linux User Group: WhiteSpace. It does what it says—tracks whitespace. I have it set up to show trailing whitespace and, additionally, to eliminate any such whitespace when I save. I also have it show tabs and modify them according to my jEdit soft tab setting. This setting, when enabled, converts

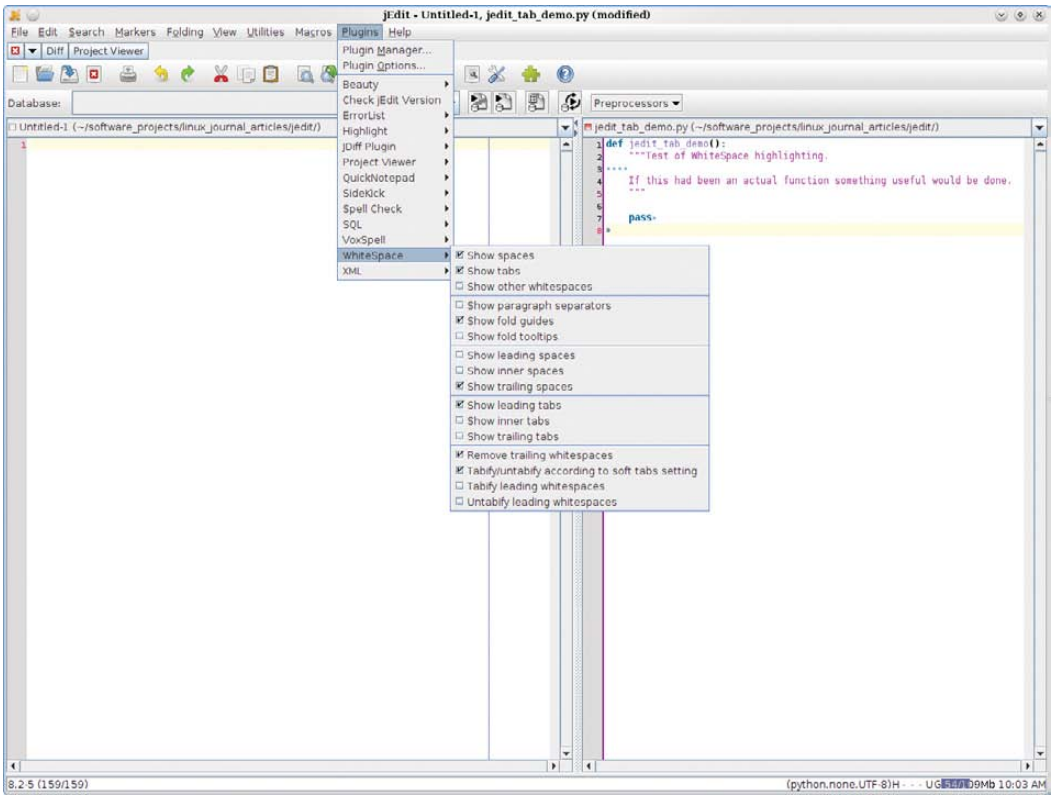


Figure 1. WhiteSpace Plugin Settings and at Work

tabs into a defined number of spaces. WhiteSpace uses the setting to convert preexisting tabs into spaces or vice versa. All of the above helps when I work in Python code, keeping that pesky whitespace in order. This also is valuable when writing for *Linux Journal*, which requires that writers use spaces not tabs. See Figure 1 for WhiteSpace in action on a Python file. From the screenshot, you can see one way to set it up. The other way is to go to Plugins→Plugin Options→WhiteSpace. This is how most of the plugins work, although you will find there often are differences in options available between the two locations.

Another plugin I use quite often is JDiff. As the name implies, it shows the diff between files. Of course, you could use the command line to do the same thing. The benefit of the plugin is the graphical presentation it provides. Figure 2 shows the dual-diff mode using this article as the files. From here, you can create a diff output. You also can walk through the diffs and apply them from one side to another.

The JDiff plugin has a dockable component that allows you to drill down into the lines of the files for differences (Figure 3). In the dual-diff screenshot (Figure 2), you can see

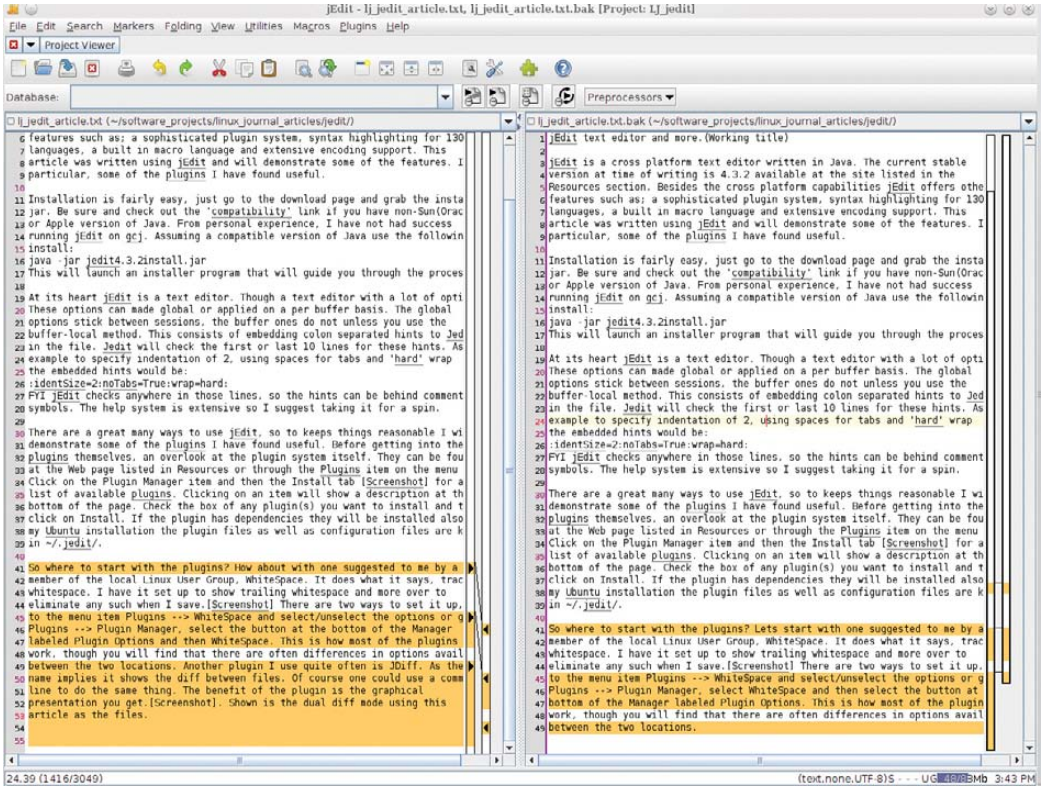


Figure 2. JDiff Plugin in Dual-Diff Mode

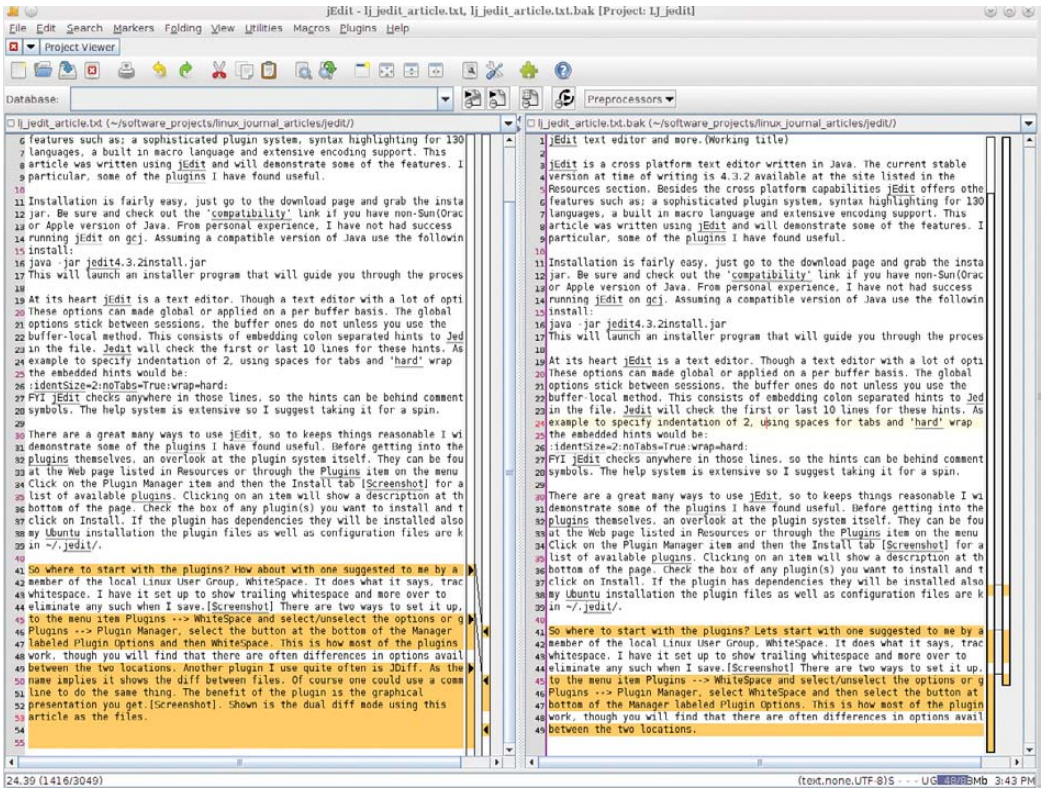


Figure 3. JDiff Dockable Showing Line Differences

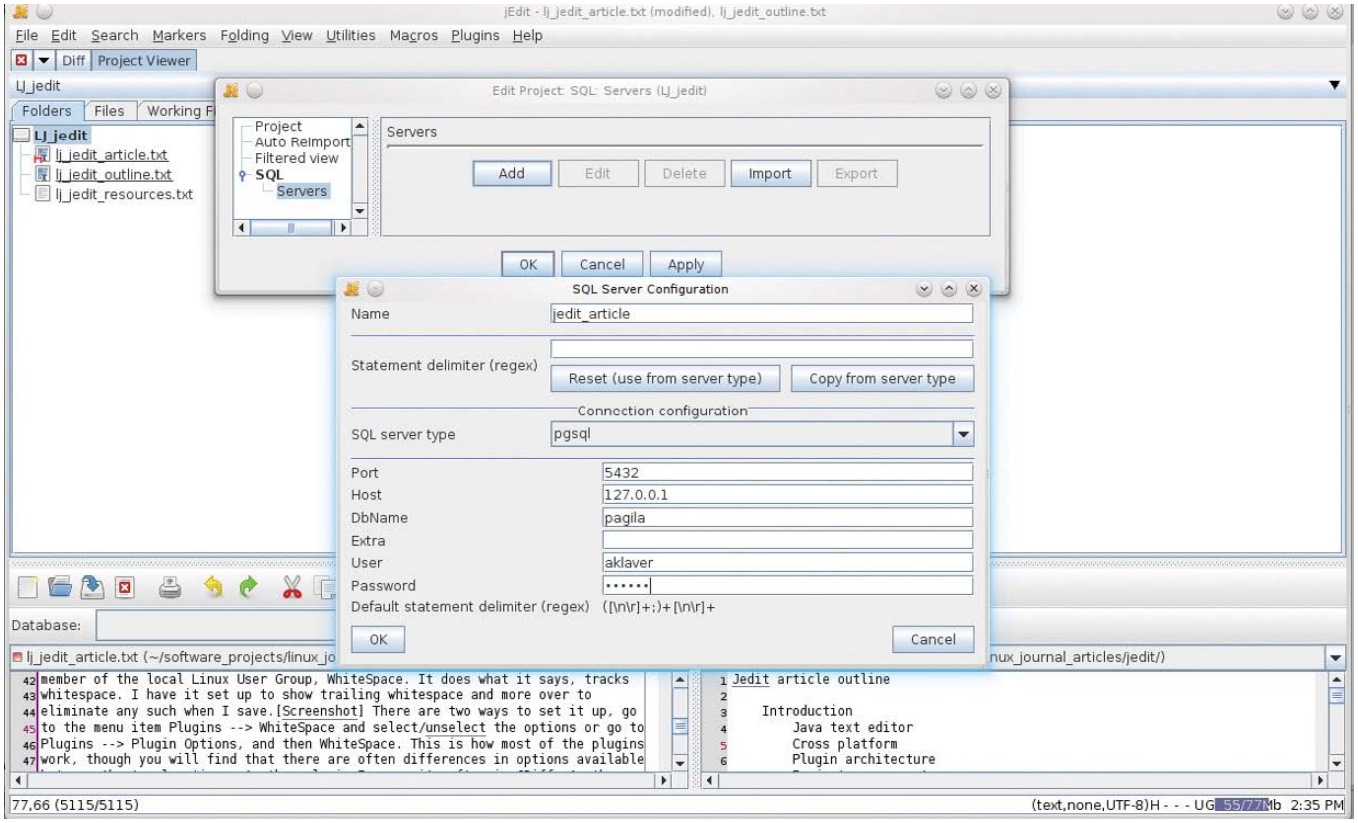


Figure 4. Adding SQL Server to a Project

another plugin at work, VoxSpell. The underlining is the spell-checker at work. If you look at the file, you can see that plugin/plugins go from being underlined to not underlined. I right-clicked on the words and added them to the dictionary as acceptable, at least for the purposes of this article. Note that VoxSpell has a dependency on the Spell Check plugin. It also uses quite a bit of memory, so that may be an issue.

Next is a chicken-and-egg problem. I ran across a reference to the SQL plug-in for jEdit. In the course of installing it, I found it had a dependency on the Project Viewer plugin, which meant I had to learn how to use Project Viewer in order to use the SQL plugin. It turns out that was a good thing. In fact, this article was written using Project Viewer. First, I will cover the SQL plugin and later expand on the Project Viewer plugin.

SQL allows you to work with SQL databases from within the editor. Setting things up to use the plugin is a two-step process. First, you need to do the general setup in the SQL options dialog. Go to the menu, then

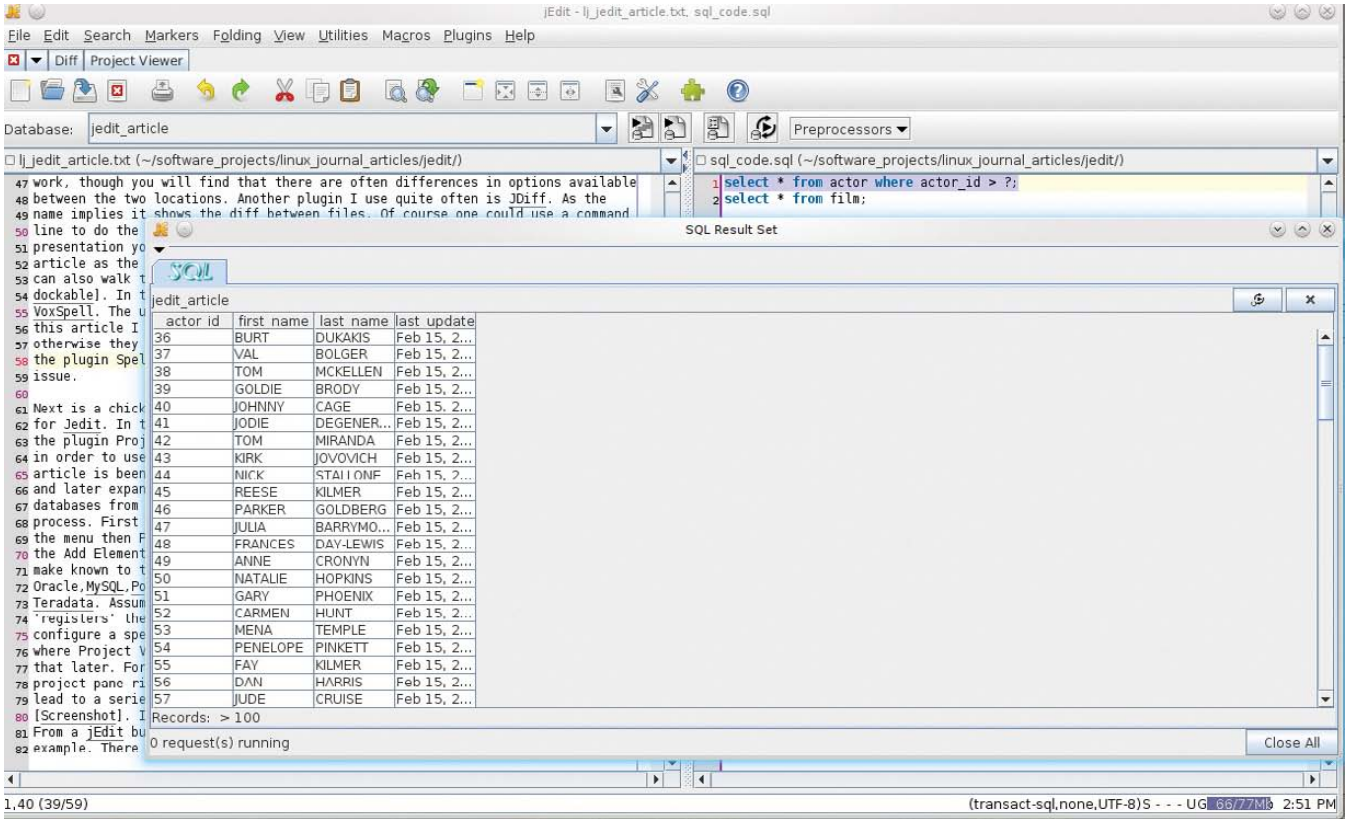


Figure 5. SQL Plugin Returning a Result Set

Plugins→Plugins Options→SQL. Go to the JDBC page and use the Add Element button to indicate the path(s) to the JDBC drivers you want to make known to the program. At the time of this writing, SQL can work with Oracle, MySQL, PostgreSQL, Firebird, DB2, Progress, MS SQL Server 2000, Sybase and Teradata, assuming you have the requisite JDBC drivers. This previous step "registers" the database so it can be used in the next step.

The next step is to configure a specific database (or maybe more than one) with a project. This is where Project Viewer comes in. You use it to create the project (more detail on that later). For now, I will use the project that is this article. From the project pane, right-click on the project name and select properties. This will lead to a series of dialogs that allow you to fill in the needed information (Figure 4). In this case, I am using the Pagila demo database for Postgres. From a jEdit buffer, you now have access to the database (Figure 5). There is quite a bit going on there, so let's take it a step at a time.

Just above the buffers is the SQL

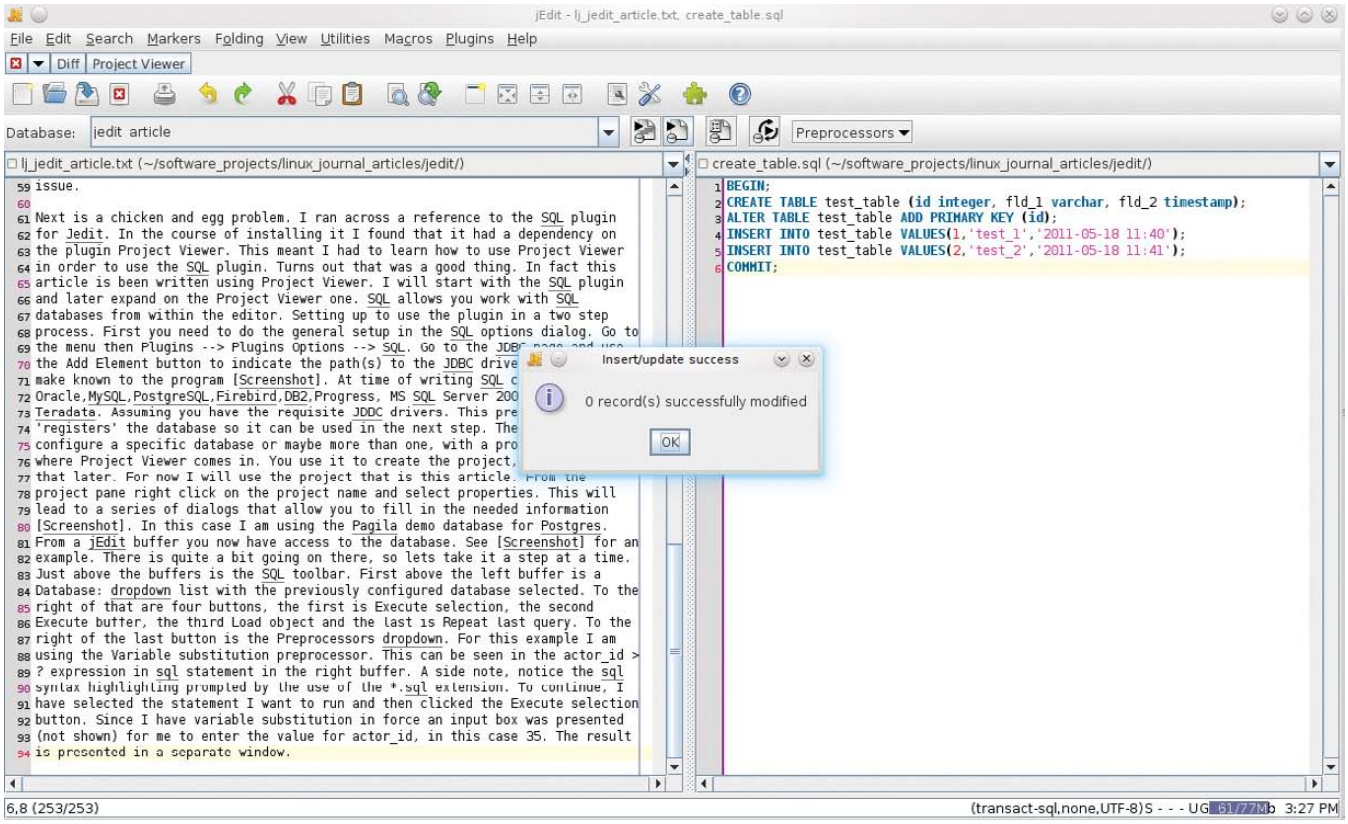


Figure 6. SQL Plugin Running Multiple Statements

toolbar. First, above the left buffer is a Database: drop-down list with the previously configured database selected. To the right of that are four buttons: the first is Execute selection, the second is Execute buffer, the third is Load object, and the last is Repeat last query. To the right of the last button is the Preprocessors drop-down. For this example, I am using the Variable substitution preprocessor. This can be seen in the "actor_id > ?" expression in the SQL statement in the right buffer. (As a side note, notice the SQL syntax highlighting prompted by the use of the *.sql extension.) To continue, I have selected the statement I want to run and then clicked the Execute selection button. Because I have variable substitution in force, an input box was presented (not shown) for me to enter the value for actor_id, in this case 35. The result is presented in a separate window. From the result set, it is possible to save the data as CSV or tab format or as INSERT statements. You also can show/hide columns. Additionally, it's possible to run multiple statements at once (Figure 6). This is a somewhat contrived example, but

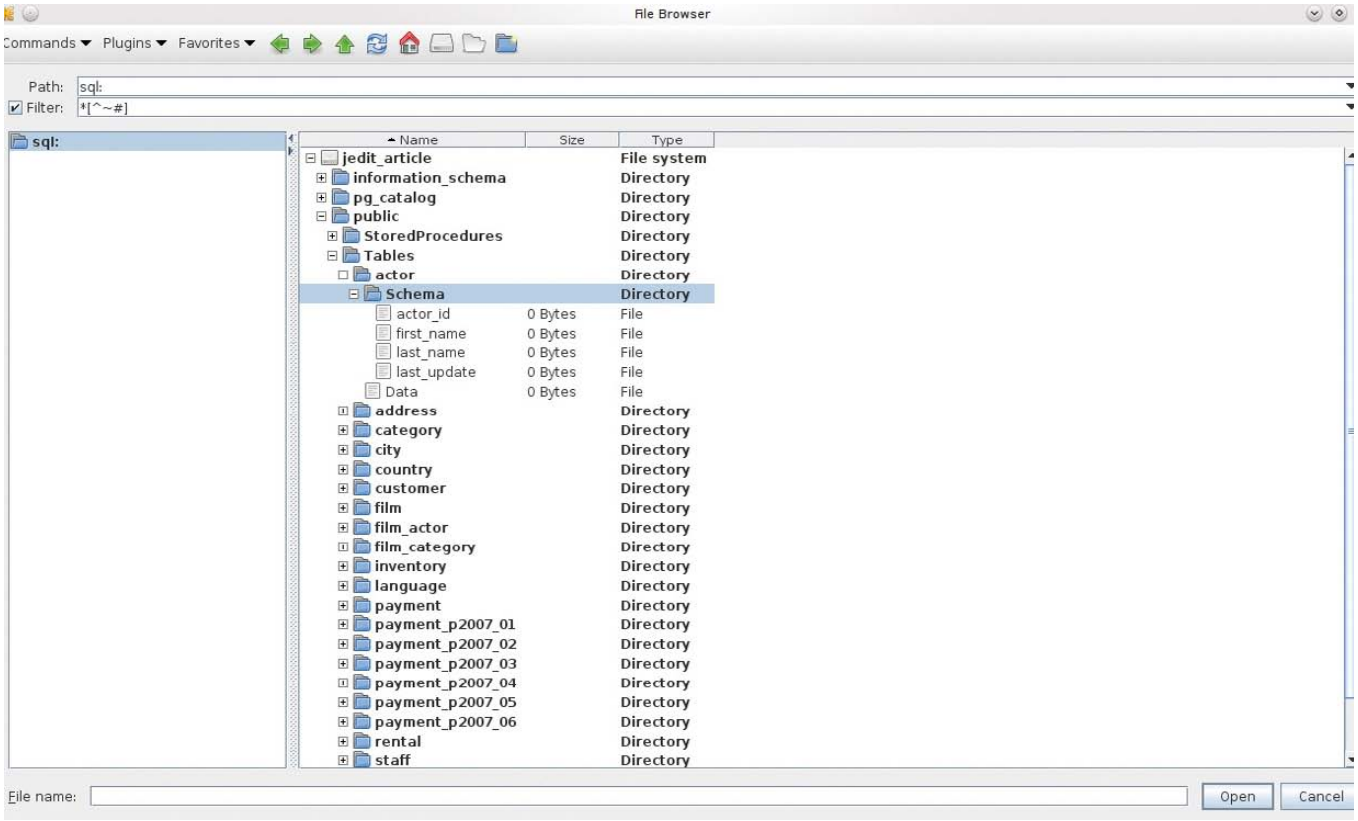


Figure 7. SqlVFS, Database as a Filesystem

it does show what's possible. The previous feature allows me to create database DDL files and run them from within the editor.

The last feature is SqlVFS (Sql Virtual File System). This allows you to browse the selected database as a filesystem. To get there, go to File→Open→Commands→Plugins→Show databases. Figure 7 shows what you get. Note that although Data says 0 bytes, double-clicking on it gives a result set from the table.

Project Viewer is a plugin to make handling a group of related files (a project) easier. For demonstration purposes, I'm using the files that make up this article. Project Viewer creates a docked button below the menu bar. Click it, and a drop-down appears with All Projects listed. Click this, and a window opens. Right-click on All Projects, and select Add project, and you get another window (Figure 8) to enter the required information. Click OK to create the project. Project Viewer then takes you to that project and puts up a prompt about importing files into the project. By default, it imports everything below the root directory.

Figure 8. Setting Up a Project Using Project Viewer

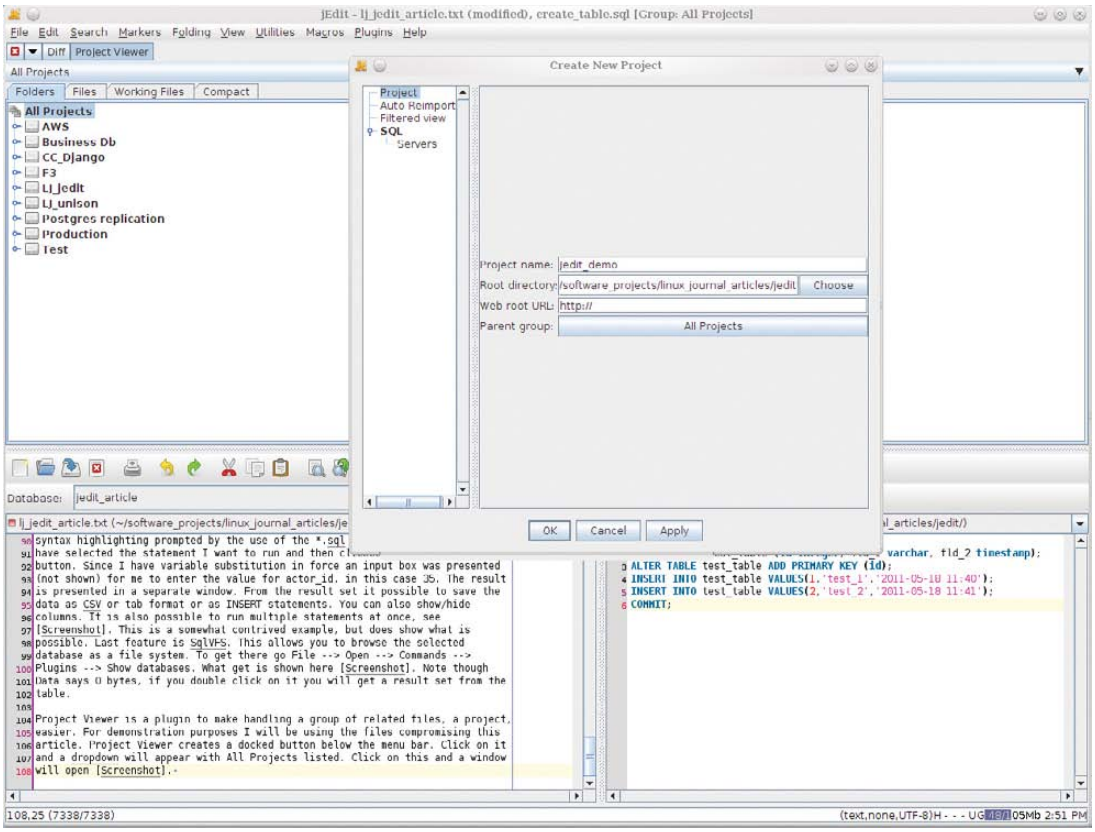
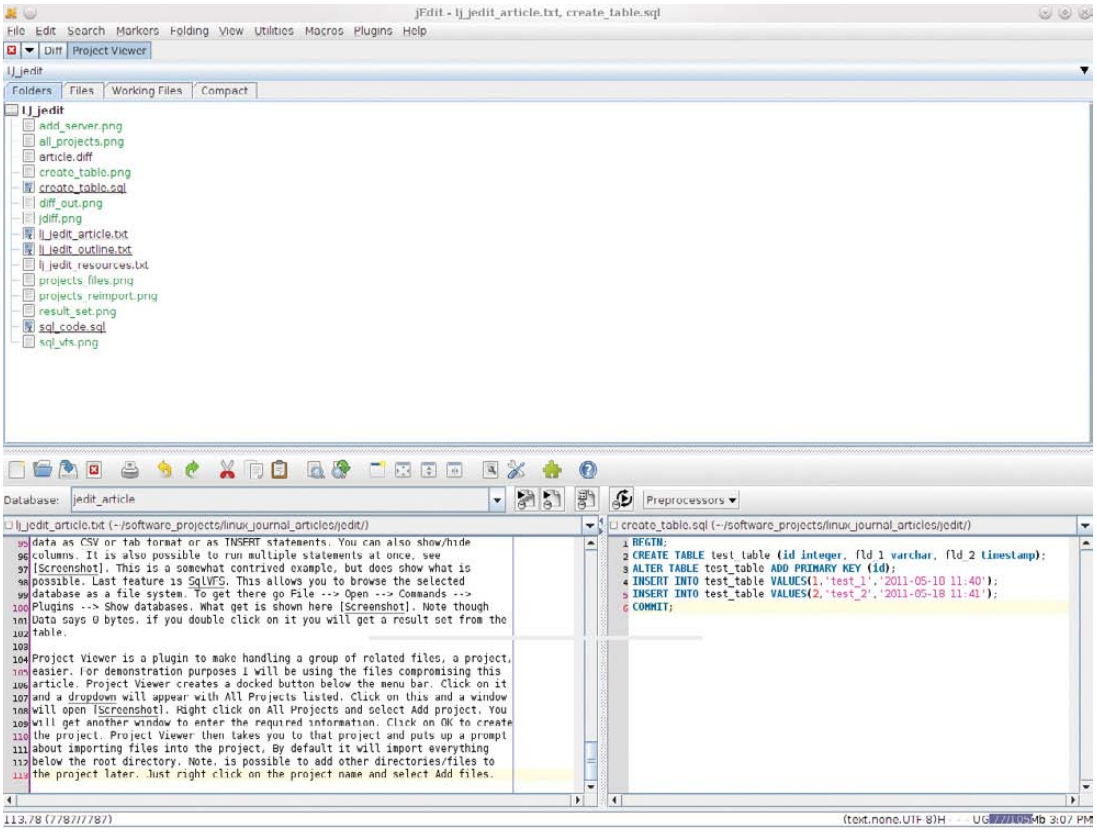


Figure 9. Files in Project Viewer



The benefit is that Project Viewer keeps track of their state and allows you to return to that state at a later time.

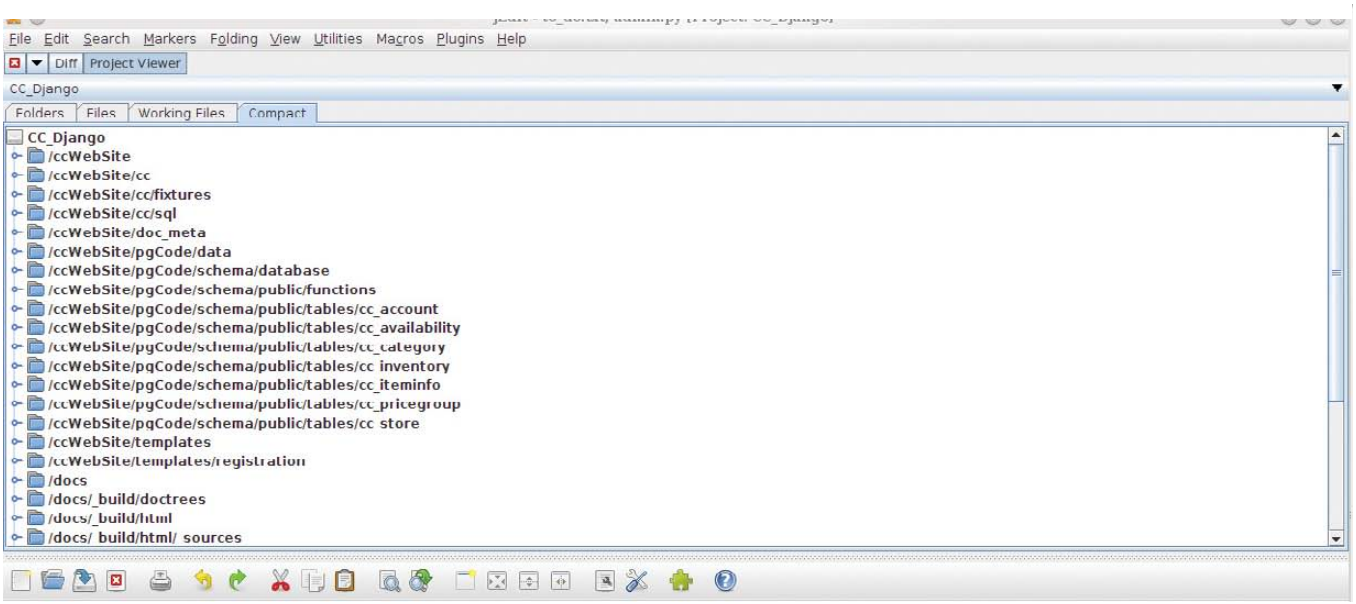


Figure 10. Project Viewer Compact View

At this point, the group of files is bound together as a project. The benefit is that Project Viewer keeps track of their state and allows you to return to that state at a later time. Note that it's possible to add other directories/files to the project later. Simply right-click on the project name, select Add files and navigate to the desired location(s). You can add new files from the existing directory in two ways: one is manual and the other automatic. The manual option is to right-click the project name and select Re-import files. The automatic option is to open the project name context menu, select Properties→Auto Reimport and enable it with a time parameter. Figure 9 shows the visual indicators as to the status of files, where underlining represents open files and color indicates type of files.

One really handy feature is the Compact View of a project. This is enabled in the General Options of the Project Viewer plugin options. It presents a flattened view of a directory structure. This article does not really have the directory depth to illustrate the benefit, so take a look at a screenshot from another project (Figure 10). Each line takes you directly to a directory.

Also of note is the Working Files tab in the project window. This groups all your current open files together, which is handy in a large project.

Some other features include archiving the project files in a JAR file and searching in the project or project subdirectory files for a string. The ability to consolidate all of the files related to a project in a single interface is something I've come to appreciate even more as time passes. Walking through a Project Viewer directory tree renaming/moving/deleting files while looking at the actual files is priceless.

What I have presented above barely scratches the surface of what is possible with jEdit. It has a macro facility that I have not even started to explore. The most important part of jEdit, to me, is that it lets me get work done without getting in my way. Furthermore, it makes that work easier, and I hope you find it useful also.■

Adrian Klaver works with computers, and when that proves frustrating, he pushes wheelbarrows of heavy stuff around to remind himself that maybe computers are not so bad after all.

Save the Date!

LISA'11

25TH LARGE INSTALLATION SYSTEM ADMINISTRATION CONFERENCE

December 4–9, 2011, Boston, MA

SPONSORED BY usenix IN COOPERATION WITH LOPSA and SNIA

Come to LISA '11 for training and face time with experts in the sysadmin community.

The theme for LISA '11 is "DevOps: New Challenges, Proven Values."

LISA '11 will feature:

6 days of training on topics including:

- Virtualization
- Security
- Configuration Management
- And more!

Plus a 3-day Technical Program:

- Invited Talks
- Paper Presentations
- Guru Is In Sessions
- Practice and Experience Reports
- Vendor Exhibition
- Workshops
- Posters and WiPs

Find out more at www.usenix.org/lisa11/lj

Moose

Write powerful object-oriented code in a modern and consistent style—in Perl.

HENRY VAN STYN

Perl has been around for more than 20 years. During that time, it has received its share of both praise and criticism, and lots of misconceptions surround it. Much of this stems from long-outdated notions of what Perl used to be, but have nothing to do with what Perl actually is today.

Perl hasn't been standing still. It's been growing continuously and evolving, and that growth has accelerated dramatically in the past few years. Moose is one of the technologies at the heart of this "Perl Renaissance", which also includes other exciting projects that have emerged, such as Catalyst and DBIx::Class.

Moose is essentially a language extension for Perl 5 that provides a modern, elegant, fully featured object system. I say "language extension", but Moose is written in pure Perl, and as you'll see, its syntax is still normal Perl. You don't need to patch Perl itself to use Moose; under the hood, it's just Perl 5.

Because Moose is still just Perl 5, it's fully compatible with all of those wonderful modules on CPAN, regardless of whether they are written in Moose (and most aren't, as CPAN has been around for so long, and Moose is relatively new).

For me, this is still the single biggest reason to choose Perl. Whatever you are trying to accomplish, chances are, there already is a refined module for it on CPAN. This usually means dramatic cuts in total development time, because someone else already has written a lot of your program for you.

And now, with all the modern object-oriented features Moose brings to Perl, you get to have your cake and eat it too.

In this article, I provide an introduction to object-oriented programming in Moose and cover some of Moose's core features with useful examples. To get the most out of this article, you already should be familiar with object-oriented programming concepts, such as classes, objects, methods, attributes, construction and inheritance.

You also need to know Perl—at least the fundamentals. If you don't know Perl, learning it is not very hard to do.

What about Perl 6?

A lot of the features in Moose were inspired by Perl 6. Perl 6 still is being developed actively, and I believe that when it's finally released for production use, it won't disappoint. The fact is Perl 5 is solid, proven and fast, so there is no reason to rush Perl 6. It is better that the developers take the time to do it really right, which is exactly what they're doing.

At the end of the day, it's just syntax. The good news is you don't need to master Perl by any stretch to start using Moose.

Perl does have its quirks, and Moose doesn't make them all totally go away (and you wouldn't want them all to go away, because a lot of them are really useful). The most important concepts to understand are how Perl references work (the "perlreftut" tutorial is a great place to start—see Resources), and also the basics of working with Scalars, Arrays and Hashes. Also, learn what the fat comma is (=>) if you aren't already familiar with it. Moose makes heavy use of it as an idiom. It's actually not that scary; it's interchangeable with the normal comma (,).

Most of the rest of it you can learn as you go. Normal language stuff like loops, conditionals and operators aren't all that different in Perl than any other language. So give it a shot. I think you'll find it's well worth the investment.

Getting Moose

Chances are you already have a distribution of Perl installed on your system. You at least should have Perl 5.8, but preferably 5.10 or 5.12. Installing Moose from CPAN is an easy task; simply run the following command:

```
cpan Moose
```

This should download and install Moose for you, as well as all of Moose's dependencies.

Object-Oriented Perl (the Old Way)

Even though Perl has had object-oriented features for a long time, it was not originally designed—syntactically—as an object-oriented language. This is more about the API provided to the programmer than it is about the underlying technical design of Perl itself.

Perl 5 provides a lean environment with the fundamental features and hooks needed for object-oriented programming, but then leaves most of the details (such

as setting up object constructors, implementing attributes and handling validation) to you. As a result, the "right way" to go about implementing these concepts is open to interpretation.

The fundamental feature utilized by Perl to support objects is the "blessed" reference. This is like the flux capacitor of objects in Perl. Blessing simply associates a normal reference (usually a Hash reference) with a class. The blessed reference then becomes the "object instance", and its referent is used as the container to store the object's data.

The class name is the same thing as the package name, which is nothing more than the namespace in which subroutines and variables are defined. The subroutines defined in the given package namespace become the methods of the class and can be called on the object reference.

All object-oriented languages have to do something along these lines to implement objects under the hood. Other languages just don't impose so many of the low-level details on the programmer as in pure Perl.

Here is an example of a simple class in old-school Perl 5 OO:

```
package MyApp::Rifle;
use strict;

sub new {
        my ($class, %opts) = @_;
        $opts{rounds} = 0 unless ($opts{rounds});
        my $self = bless( {}, $class );
        $self->rounds($opts{rounds});
        return $self;
}

sub rounds {
        my ($self, $rounds) = @_;
        $self->{_rounds} = $rounds if (defined $rounds);
        return $self->{_rounds};
}

sub fire {
        my $self = shift;
        die "out of ammo!" unless ($self->rounds > 0);
        print "bang!\n";
        $self->rounds( $self->rounds - 1 );
}

1;
```

Save the above class definition into a file named MyApp/Rifle.pm within one of your Perl's include directories, and then you can use it in a Perl program like this:

```
use MyApp::Rifle;
use strict;

my $rifle = MyApp::Rifle->new( rounds => 5 );
print "There are " . $rifle->rounds . " rounds in the rifle\n";
$rifle->fire;
print "Now there are " . $rifle->rounds . " rounds in the rifle\n";
```

Moose Sugar

Moose is really nothing more than syntactic "sugar" that automatically takes care of the boiler-plate tedium and low-level details of implementing objects automatically. This is possible because of Perl's powerful introspection capabilities—Moose dynamically manipulates the class definition at compile time just as if it had been written that way.

The previous class could be implemented like this with Moose:

```
package MyApp::Rifle;
use Moose;

has 'rounds' => ( is => 'rw', isa => 'Int', default => 0 );

sub fire {
        my $self = shift;
        die "out of ammo!" unless ($self->rounds > 0);
        print "bang!\n";
        $self->rounds( $self->rounds - 1 );
}

1;
```

Not only is this code much shorter, cleaner and easier to read, but it is doing all the things the non-Moose class was doing and more. First, Moose is automatically creating the "new" constructor method behind the scenes. It is also automatically setting up "rounds" as an attribute (aka object variable), which Moose understands as a distinct concept.

Pure Perl has no such understanding; if you want "attributes", you have to implement them yourself by writing accessor methods by hand and deciding how they should work (the non-Moose version above illustrates only one of many possible approaches).

Moose, on the other hand, provides a refined, fully integrated object attribute paradigm. It sets up the accessor methods, handles the data storage and retrieval, and automatically configures the constructor to initialize attributes optionally with supplied parameters—and that is just scratching the surface!

One of the problems with the non-Moose version of our class is that there is no validation for "rounds". For example, nothing stops us from doing this:

```
my $rifle = MyApp::Rifle->new( rounds => 'foo' );
```

This is one of the areas where Moose really shines; it provides a complete Type system that is very straightforward to use. In the Moose version, the option `isa => 'Int'` sets up the rounds attribute with a type constraint that automatically will throw an exception (with a meaningful message) if you try to set the value to anything that is not a valid integer. This would stop you from setting rounds to 'foo' because it's

not an integer, it's a string.

This illustrates an important point about Moose's design and approach. Its syntax is *declarative* rather than *imperative*. This means you just need to specify what you want it to do instead of how it needs to do it. This is in sharp contrast to the traditional Perl 5 OO style, where that is exactly what you would have to do—add additional lines of code in the accessor method to test the value for validity and handle the result.

The syntax `isa => 'Int'` doesn't provide any insight on *how* Moose will go about checking and enforcing the type constraint. And that's the whole point—you don't care. But, you can rest assured it will do it in a far more thorough and robust manner than anything you'd want to waste time on doing yourself.

More about Attributes

You declare attributes in Moose with the "has" function. This consists of a unique attribute name, followed by a list of named options (key/values). Although it looks and behaves like a built-in language keyword, it is really just a function call. Its documented syntax is just idiomatic for the purpose of code readability (it's a fancy way to pass an argument list).

Moose provides all sorts of options that define the behavior of a given attribute, including setup of accessors, data types, initialization and event hooks. The simplest attribute is just an object variable that is set up as either read-write (rw) or read-only (ro) with the "is" option:

```
has 'first_name' => ( is => 'rw' );
```

The is option tells Moose to set up the accessor method, which is what you use to get and set the attribute's value. You set the value of an attribute by passing a single argument to the accessor (such as `$obj->first_name('Fred')`), and you get the current value by calling the accessor with no arguments (`$obj->first_name`). Setting the value is only allowed if the attribute "is" => "rw". If it's "ro", and you try to set its value through the accessor an exception will be thrown.

This is the core of the attribute paradigm, but lots of other options provide useful features. Here are a few of the noteworthy ones:

- is: ro or rw, creates either a read-only or read-write accessor method.
- isa: string representing an optional type constraint.
- default: the default value of the attribute.

- builder: alternative to default—name of a method that will generate the default.
- lazy: if true, the attribute is not initialized until it's used.
- required: if true, attribute value must be provided to constructor or have default/builder.

The builder option lets you specify a method to use to populate the attribute's default value. A builder is a normal method defined within the class, and its return value is used to set the attribute's initial value. If the builder needs to access other attributes within the object, the attribute must be lazy (to prevent it from potentially being populated before the other attributes it depends on).

Because this is such a common scenario, for convenience, Moose provides the "lazy_build" attribute option that automatically sets the lazy option and sets the builder to `_build_name` (such as `_build_first_name` for an attribute named `first_name`). For example:

```
has 'first_name' => ( is => 'ro', lazy_build => 1 );
sub _build_first_name {
        my $self = shift;
        return $self->some_lookup('some data');
}
```

Attributes Containing Objects

So far, I've talked only about attributes containing simple scalars. Attributes can contain other types of values as well, including references and other objects. For example, you could add a DateTime attribute to your MyApp::Rifle class to keep track of the last time "fire" was called:

```
package MyApp::Rifle;
use Moose;
use DateTime;

has 'rounds' => ( is => 'rw', isa => 'Int', default => 0 );
has 'fired_dt' => ( is => 'rw', isa => 'DateTime' );

sub fire {
        my $self = shift;
        die "out of ammo!" unless ($self->rounds > 0);

        my $dt = DateTime->now( time_zone => 'local' );
        $self->fired_dt($dt);

        print "bang!\n";
        print "fired at " . $self->fired_dt->datetime . "\n";

        $self->rounds( $self->rounds - 1 );
}

1;
```

This is fairly straightforward. I'm creating a new DateTime object and storing it in my fired_dt attribute. Then, I can

call methods on this object, such as the datetime method, which returns a friendly string representing the date and time.

Delegations

Alternatively, you could utilize Moose's delegation feature when you set up the fired_dt attribute, like this:

```
has 'fired_dt' => (
        is => 'rw',
        isa => 'DateTime',
        handles => {
                last_fired => 'datetime'
        }
);
```

This will set up another accessor method named last_fired and map it to the datetime method of the attribute. This makes the invocations of `$self->last_fired` and `$self->fired_dt->datetime` equivalent. This is worthwhile because it allows you to keep your API simpler.

Types

Moose provides its own type system for enforcing constraints on the value to which an attribute can be set. As I mentioned earlier, type constraints are set with the isa attribute option.

Moose provides a built-in hierarchy of named types for general-purpose use. For example, Int is a subtype of Num, and Num is a subtype of Str. The value `'foo'` would pass Str but not Num or Int; 3.4 would pass Str and Num but not Int, and 7 would pass all of Str, Num and Int.

There also are certain built-in types that can be "parameterized", such as ArrayRef (a reference to an array). This lets you not only require an attribute to contain an ArrayRef, but also set type constraints on the values that ArrayRef can contain. For example, setting `isa => 'ArrayRef[Int]'` requires an ArrayRef of Ints. These can be nested multiple levels deep, such as `'ArrayRef[HashRef[Str]]'` and so on.

Another special parameterized type is Maybe, which allows a value to be undef. For example, `'Maybe[Num]'` means the value is either undef or a Num.

You also can use type "unions". For example, `'Bool | Ref'` means either Bool or Ref.

If the built-in types aren't sufficient for your needs, you can define your own subtypes to do any kind of custom validation you want. The Moose::Util::TypeConstraints documentation provides details on creating subtypes, as well as a complete listing of the built-in types that are available (see Resources).

Finally, instead of specifying the name

of a defined type, you can specify a class name, which will require an object of that class type (such as in our DateTime attribute example). All of these concepts can be intermixed for maximum flexibility. So, for example, if you set `isa => 'ArrayRef[MyApp::Rifle]'`, it would require an ArrayRef of MyApp::Rifle objects.

Inheritance

Subclassing is relatively painless in Moose. Use the extends function to make a class a subclass of another. The subclass inherits all the parent's methods and attributes, and then you can define new ones or override existing ones simply by defining them again.

Moose also provides helpful attribute inheritance sugar that allows you to inherit an attribute from the parent, but override specific options in the subclass. To tell Moose to do this, prepend the attribute name with a plus sign (+) in a "has" declaration in the subclass. (Note: attribute options related to accessor method names cannot be changed using this technique.)

For example, you could create a new class named MyApp::AutomaticRifle that inherits from the MyApp::Rifle class from the previous example:

```
package MyApp::AutomaticRifle;
use Moose;
extends 'MyApp::Rifle';

has '+rounds' => ( default => 50 );
has 'last_burst_num' => ( is => 'rw', isa => 'Int' );

sub burst_fire {
        my ($self, $num) = @_;

        $self->last_burst_num($num);

        for (my $i=0; $i<$num; $i++) {
                $self->fire;
        }
}

1;
```

Here, MyApp::AutomaticRifle can do everything MyApp::Rifle can do, but it also can "burst_fire". Also, the default of the rounds attribute has been changed to 50 in AutomaticRifle, but the rest of the options for the rounds attribute still are inherited from the parent Rifle class.

You might use MyApp::AutomaticRifle like this:

```
use strict;
use MyApp::AutomaticRifle;

my $rifle = MyApp::AutomaticRifle->new;
print "There are " . $rifle->rounds . " rounds in the rifle\n";
$rifle->burst_fire(35);
print "Now there are " . $rifle->rounds . " rounds in the rifle\n";
```

The BUILD Method

Although Moose automatically sets up the "new" constructor for you, there still are times when you need to execute custom code at construction. If you need to do that, define a method named BUILD, and it will be called immediately after the object has been constructed. Don't create a "new" method; that will interfere with Moose's operation.

BUILD is also special as it relates to inheritance. Unlike normal methods that override the parents' methods when redefined in subclasses, BUILD can be defined in every class in the inheritance tree and every one will be called, in order from parent to child.

Roles

Roles define some set of behaviors (attributes and methods) without being full-blown classes themselves (capable of instantiation as objects directly). Instead, Roles are "composed" into other classes, applying the defined behaviors to those classes. Roles are conceptually similar to "mixins" in Ruby.

Roles also can require that consuming classes have certain methods by calling the "requires" sugar function in the Role definition (or throw an exception).

You call the "with" sugar function to consume a Role by name, just like you call "extends" to inherit from a regular class.

Here is an example of a simple Role that could be composed into either MyApp::Rifle or MyApp::AutomaticRifle:

```
package MyApp::FireAll;
use strict;
use Moose::Role;

requires 'fire', 'rounds';

sub fire_all {
        my $self = shift;
        $self->fire while($self->rounds > 0);
}

1;
```

You would then add this single line to MyApp::Rifle or MyApp::AutomaticRifle to give either class the fire_all method:

```
with 'MyApp::FireAll';
```

In the case of MyApp::AutomaticRifle, the with statement must be called after the extends statement, because the "fire" and "rounds" methods don't exist within MyApp::AutomaticRifle before that, and the Role's requires statements would fail.

If you add the Role to MyApp::Rifle, it will be inherited by MyApp::AutomaticRifle automatically, so there would be no need to add it there also (although it

won't break anything if you do).

Method Modifiers

Method modifiers are one of the more powerful and flexible features of Moose. The most common types of modifiers are before, after and around. Before and after are really just "hooks" to execute some code whenever a given method is called, either before or after, as the names imply. For example, this would print a string every time `fire_all` is called:

```
before 'fire_all' => sub {
        my $self = shift;
        print "Say hello to my little friend!\n";
};
```

The "around" modifier is quite a bit more powerful than before and after because it actually can change the arguments passed to, and the data returned from, the original method. It also can programmatically decide whether even to call the original method at all.

Around modifiers actually replace the original method, but get passed the original method and arguments to be able to call it within the new modifier function, but unlike before and after, this has to be done manually in around. The basic blueprint of this is below, which is an example of an around modifier that exactly reproduces the original method (having no observable effect):

```
around 'fire_all' => sub {
        my ($orig, $self, @args) = @_;
        return $self->$orig(@args);
};
```

In an around modifier, the first argument is the method (`$orig`) instead of the object reference (`$self`) like in normal methods. Then, it's up to you to call the original method (`$self->$orig`) and capture its return value (or not) and then return.

NOTE: The semicolons at the end of the method modifier definitions in the examples are required. Like all the keywords provided by Moose, the modifier sugar keywords actually are function calls and are *not* subroutine definitions. The modifier definitions are all just function calls with exactly two arguments: a string representing the name of the method to modify and a *code reference* to the actual modifier. CodeRefs are just treated syntactically as values like any other. It's not important to understand this fully to use method modifiers, but it is important to remember to use the semicolons.

Method modifiers make a great fit with Roles to define behaviors at a

fine-grained level. Let's take a look at another example of a Role for our MyApp::Rifle class that makes use of method modifiers:

```
package MyApp::MightJam;
use Moose::Role;
use Moose::Util::TypeConstraints;

requires 'fire';

subtype 'Probability' => (
        as 'Num',
        where { $_ >= 0 && $_ <= 1 },
        message { "$_ is not a number between 0 and 1" }
);

has 'jam_probability' => (
        is => 'ro',
        isa => 'Probability',
        default => .01
);

sub roll_dice {
        my $self = shift;
        return 1 if ( rand(1) < $self->jam_probability );
        return 0;
}

before 'fire' => sub {
        my $self = shift;
        die "Jammed!!!\n" if ($self->roll_dice);
};

1;
```

This Role adds the random chance of "Jamming" on any given call to "fire" depending on the probability specified in the jam_probability attribute (with the default probability set to 1%). I also illustrate here how to create a custom subtype, by defining a new type "Probability", which must be a number between 0 and 1.

You then could compose simple subclasses like the following:

```
package MyApp::CrappyRifle;
use strict;
use Moose;
extends 'MyApp::AutomaticRifle';
with 'MyApp::MightJam';

has '+jam_probability' => ( default => .5 );

1;
```

And:

```
package MyApp::NiceRifle;
use strict;
use Moose;
extends 'MyApp::AutomaticRifle';
with 'MyApp::MightJam';

has '+jam_probability' => ( default => .001 );

1;
```

The difference between these two is

If you get stuck on something and can't find the answer, try the #moose IRC channel on irc.perl.org

that CrappyRifle will jam on average 5 out 10 times, and NiceRifle will only jam 1 per 1,000 times.

Learning More

This article is just meant as an introduction to Moose, and because of space constraints, I have been able to cover only a few of its core features.

One of the other great things about Moose, and Perl in general, is the community and availability of documentation and resources. The Moose Manual, available on CPAN (see Resources), is well-written and comprehensive. There are also plenty of other docs and information available, and the number of them is growing every day as Moose continues to gain popularity.

If you get stuck on something and can't find the answer, try the #moose IRC channel on irc.perl.org. Many of the top experts are in this channel and are more than willing to help and answer questions. Although they will expect you to RTFM and have done your homework first, they will get you unstuck and pointed in the right direction.

If nothing else, I hope that this article has at least piqued your interest in modern development with Perl and Moose, and that you can see that Perl code can, in fact, be clean, easy to read and modern, while still being "Perlish" and powerful.

As you learn Moose, and modern Perl in general, be sure to check out some of the other projects and modules that are available, including Catalyst, Template::Toolkit, DBIx::Class, Try::Tiny, Test::More and Devel::NYTProf, just to name a few. You might be surprised what's out there, and what is really possible with Perl today.■

Henry Van Styn is the founder of IntelliTree Solutions, an IT consulting and software development firm located in Cincinnati, Ohio. Henry has been developing software and solutions for more than ten years, ranging from sophisticated Web applications to low-level network and system utilities. He is the author of Strong Branch Linux, an in-house server distribution based on Gentoo. Henry can be contacted at www.intellitree.com.

Resources

Moose CPAN Page: **search.cpan.org/perldoc?Moose**

Moose Manual: **search.cpan.org/perldoc?Moose::Manual**

Moose::Util::TypeConstraints Documentation: **search.cpan.org/perldoc?Moose::Util::TypeConstraints**

Moose IRC Channel: #moose on irc.perl.org

perlreftut—Perl Reference Tutorial: **perldoc.perl.org/perlreftut.html**

KYLE RANKIN

BILL CHILDERS

Unboxing Day

It's Christmas time, and Kyle is about to open his new present: a large shipment of blade servers. Find out about the extra present his vendor left him.

As much as I love working with Linux and configuring software, one major part of being a sysadmin that always has appealed to me is working with actual hardware. There's something about working with tangible, physical servers that gives my job an extra dimension and grounds it from what might otherwise be a completely abstract job even further disconnected from reality. On top of all that, when you get a large shipment of servers, and you view the servers at your company as *your servers*, there is a similar anticipation and excitement when you open a server box as when you open Christmas presents at home.

This story so happens to start during the Christmas season. We had just received our first shipment of a completely new blade infrastructure that we were really excited to try out. As the resident server monkey and general minion working under Bill's iron fist, I was to meet up with an engineer from our vendor at the data center and assist with the installation in any way I could. It was a big job—two completely populated blade chassis comprising 32 blade servers, integrated SAN switches and all the assorted power supplies and network pass-throughs that went along with it. We budgeted a full day of the engineer's time to rack the new chassis, slot the blades and make sure all hardware was functional and up to date.

[Bill: Iron fist? I like the sound of that. Reminds me of a mid-1970s Marvel Superhero...but I digress. I remember this occasion. We'd just finished piloting a VMware/Blade infrastructure at our corporate office, and we were about to roll it out to our production data center, on next-generation hardware. It was an exciting time!]

I arrived at the data center a few hours before the engineer so I could get all the boxes from shipping and receiving and move them into our cage. If you ever have ordered a blade chassis,

Instead of containing a dozen large boxes and mostly empty space, this box looked like a failed game of *Tetris*.

you know that everything arrives in these gigantic cardboard boxes that incidentally were about the size of our shared cubicle space back at the office. These boxes open up to smaller boxes for the blade servers, chassis, power supplies and the rest. At first things moved smoothly. I broke down the first set of boxes, and after a number of trips, the empty blade chassis and the blades themselves were stacked neatly near our cage.

The Jack in the Box

It wasn't until I opened the last box that I realized I was in trouble. Instead of containing a dozen large boxes and mostly empty space, this box looked like a failed game of *Tetris*. It was filled to the brim with hundreds of tiny boxes of all shapes and sizes. The engineer was going to be there soon, so I tried to organize the boxes into different piles and then filled my pushcart with swaying stacks of tiny boxes and made trip after trip to the colocation cage until all of its walls looked like the inside of a brown-brick house.

[Bill: I seem to remember you sent a couple e-mails to me along the lines of "Wow, this new stuff sure has a lot of boxes compared to the old stuff."]

This is probably a good point in the story to tell you that up to this time, we normally had taken advantage of our vendor's integration service. We standardized on servers with a certain amount of RAM, CPU revision, storage and network configuration that deviated from the base model, so our vendor would take the base order model and do the work to add CPUs, RAM and the extra parts we wanted so that when we got a server, we could just rack it and turn it on.

In this case, for some reason, we failed to request this integration service, so not only was I looking at the boxes for blades, chassis and power supplies, I had hard drives, CPUs, RAM, fiber-channel HBAs, extra NICs and even battery-backed write caches all individually wrapped in their own boxes. Instead of unboxing a blade and sliding it into its slot to install it, every single blade would need to be opened, and then each and every component would have to be opened, removed from its static wrap, and installed into the blade one by one.

[Bill: I have to say here that up until this point, the vendor always had

"thrown in" the integration service for us, and it's something we (and by we, I mean me) had taken for granted...until the day my boss called the vendor and deleted this "superfluous service" from the quote, without telling me.]

Drowning in Cardboard

When the engineer arrived, I explained the situation, and we both realized we had a long day ahead of us. At the beginning, we made great progress. He opened up and racked the chassis and power supplies until the point that we were ready to install the first blade server. At that point, we agreed on an assembly-line system where he would open up a blade, and I, like a surgical assistant, would unwrap and hand him each component in a certain order so he could install it. Then, while he finished up the blade, I would fill up the cart with empty wrappers and boxes and roll it to the trash area, so we didn't drown in anti-static wrap and cardboard. After a full-day's work, we were able to integrate 20 out of our 32 blades successfully.

Unfortunately, we had booked the engineer only for one day, but he was able to shuffle appointments around and return on Friday morning of that week to finish up. Halfway through Friday morning, we were able to finish with the blade servers so that they all were racked. We were ready to be done at that point, but we were only halfway there. We still had to install all the hard drives, integrated network pass-throughs, fiber-channel switches and finally, upgrade the firmware.

[Bill: I had forgotten how long that job took. Now that you mention it though, there was an amazing amount of cardboard generated from that. It didn't help that the data center didn't allow cardboard on the data center floor, and you had to shuttle all that stuff back and forth.]

Late for the Party

Once we powered on the blades, it looked like we were close to the finish line. I started packing up all my things so I could head home early and get dressed up for our company's big Christmas party later that evening. Naturally, it was at this point that a few of the blades wouldn't power on. After minimal troubleshooting, we were left with just one misbehaving blade. The engineer started the hardware troubleshooting process as I watched the minutes tick by. I realized I had to somehow power through Bay Area traffic, get home, put on my suit and drive back through the traffic to the party, and rush hour was rapidly approaching. Ultimately, we had to open up the server, remove all of the hardware we had

added only hours before, and insert the hardware one piece at a time until we identified a faulty DIMM slot. Finally, we were done and I was able to get to the party fashionably late.

I think the moral to this story is pretty clear. If we had only gotten all of our servers integrated ahead of time, the entire install would have taken a fraction of the time, and any hardware problems in the system would have been identified before anything was shipped to me. When you have the option, especially when it comes to large orders of servers, get all your components integrated ahead of time.

[Bill: The moral for me as a manager, is always to double-check the quote for services, and make sure that all of those are understood so they don't get labeled as non-essential and cut by people higher up the food chain. I'm usually not a fan of too many vendor services, but getting the entire system integrated by a vendor will accelerate deployment time by at least a couple days.]■

Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

Bill Childers is an IT Manager in Silicon Valley, where he lives with his wife and two children. He enjoys Linux far too much, and he probably should get more sun from time to time. In his spare time, he does work with the Gilroy Garlic Festival, but he does not smell like garlic.

Software Freedom Day

Saturday 17th September 2011

www.softwarefreedomday.org

Let's spread the word of Free / Open Source Software and celebrate SFD together!

What is Software Freedom Day?

Software Freedom Day (SFD) is a worldwide celebration of Free and Open Source Software (FOSS). Our goal in this celebration is to educate the worldwide public about of the benefits of using high quality FOSS in education, in government, at home, and in business -- in short, everywhere! The non-profit organization Software Freedom International coordinates SFD at a global level, providing support, giveaways and a point of collaboration, but volunteer teams around the world organize the local SFD events to impact their own communities.

Sponsors and Partners

CANONICAL ubuntu · Google · linode.com · FREE SOFTWARE FOUNDATION · fsfe · LINUX JOURNAL · ADMIN Network & Security · smart developer · LINUX MAGAZINE · UBUNTU user

Please check softwarefreedomday.org for full sponsor list as it was not available at the time of closing this ad.

ABERDEEN
SERVERS AND STORAGE

ANYONE INTERESTED IN SAVING MONEY?

Looks like these guys are comfortable overpaying for enterprise storage. Are You?

"Hewlett-Packard Co. agreed to buy 3Par Inc. for $2.35 billion" — *Bloomberg.com*

"EMC to Buy Isilon Systems Inc. for $2.25 Billion" — *Wall Street Journal*

"Dell to Buy Compellent for $960 Million" — *CNBC*

So what "benefit" will you see by this spending spree, other than higher costs?

The AberSAN Z-Series scalable unified storage platform, featuring the Intel® Xeon® processor 5600 series, brings the simplicity of network attached storage (NAS) to the SAN environment by utilizing the innovative ZFS file system. The AberSAN Z20 is easily found starting under $20,000.

Who gives you the best bang for the buck?

	3Par InServ F200	Compellent Storage Center Series 30	Isilon NL-Series	Aberdeen AberSAN Z20
Storage Scale-Out	✓	✓	✓	✓
Thin Provisioning	✓	✓	✓	✓
HA Clustering	✓	✓	✓	✓
VMware® Ready Certified	✓	✓	✓	✓
Async / Synchronous Replication	✓	✓	✓	✓
iSCSI / Fibre Channel Target	✓	✓	iSCSI Only	✓
Unlimited Snapshots	X	✓	✓	✓
Native Unified Storage: NFS, CIFS	X	X	✓	✓
Virtualized SAN	X	X	X	✓
Deduplication	X	X	X	✓
Native File System	none	none	OneFS	ZFS 128-bit
RAID Level Support	5 and 6	5 and 6	Up to N+4	5, 6 and Z
Raw Array Capacity (max)	128TB	1280TB	2304TB	Unlimited
Warranty	3 Years	5 Years	3 Years	5 Years
Online Configurator with Pricing	Not Available	Not Available	Not Available	**Available**

Powerful.
Intelligent.

888-297-7409
www.aberdeeninc.com/lj038

Cut Execution Time by >50% with WhisperStation-GPU

Delivered ready to run new GPU-enabled applications:

Design	Simulation		BioTech
3ds Max	ANSYS Mechanical	MATLAB	AMBER
Bunkspeed Shot	Autodesk Moldflow	ACUSIM AcuSolve	GROMACS
Adobe CS5	Mathematica	Tech-X GPULib	NAMD, VMD
			TeraChem

WhisperStation with 4 Tesla Fermi GPUs

Integrating the latest CPUs with NVIDIA Tesla Fermi GPUs, Microway's WhisperStation-GPU delivers 2x-100x the performance of standard workstations. Providing explosive performance, yet quiet, it's custom designed for the power hungry applications you use. Take advantage of existing GPU applications or enable high performance with CUDA C/C++, PGI CUDA FORTRAN, or OpenCL compute kernels.

- Up to Four Tesla Fermi GPUs, each with: 448 cores, 6 GB GDDR5, 1 TFLOP single and 515 GFLOP double precision performance
- Up to 24 cores with the newest Intel and AMD Processors, 128 GB memory, 80 PLUS® certified power supply, and eight hard drives
- Nvidia Quadro for state of the art professional graphics and visualization
- Ultra-quiet fans, strategically placed baffles, and internal sound-proofing
- New: Microway CL-IDE™ for OpenCL programming on CPUs and GPUs

Microway's Latest Servers for Dense Clustering

- 4P, 1U nodes with 48 CPU cores, 512 GB and QDR InfiniBand
- 2P, 1U nodes with 24 CPU cores, 2 Tesla GPUs and QDR InfiniBand
- 2U Twin² with 4 Hot-Swap MBs, each with 2 Processors + 256 GB
- 1U S2050 servers with 4 Tesla Fermi GPUs

Microway Puts YOU on the Cutting Edge

Design your next custom configuration with techs who speak HPC. Rely on our integration expertise for complete and thorough testing of your workstations, turnkey clusters and servers. Whether you need Linux or Windows, CUDA or OpenCL, we've been resolving the complicated issues – so you don't have to – since 1982.

2U Twin² Node with 4 Hot-Swap Motherboards Each with 2 CPUs and 256 GB

Configure your next WhisperStation or Cluster today!

microway.com/quickquote or call 508-746-7341

Sign up for technical newsletters and special GPU promotions at microway.com/newsletter

NVIDIA Tesla · Intel Channel Partner Premier 2010 · AMD Fusion Partner Premier · GSA

GSA Schedule Contract Number: GS-35F-0431N

Microway®

Technology you can count on℠