

LINUX JOURNAL™

Since 1994: The Original Magazine of the Linux Community

PROCESS TONS
OF DATA
WITHOUT
LOSING
YOUR MIND

AUGUST 2015 | ISSUE 256 | www.linuxjournal.com

PROGRAMMING

HASH TABLES

Theory
and
Practice

PICAT

Write Concise,
Declarative
and Efficient
Programs



MAKE YOUR
DATABASE DO
AS MUCH WORK
AS POSSIBLE

DOING
ASTRONOMY
WITH PYTHON

WHAT'S NEW
IN 3D PRINTING
SOFTWARE

LINUX
GAMING
ROUNDUP



WATCH:
ISSUE
OVERVIEW

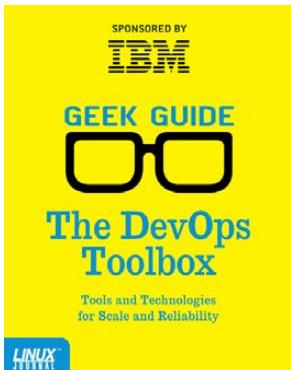


NEW!

Linux Journal eBook Series

GEEK GUIDES

The DevOps Toolbox: Tools and Technologies for Scale and Reliability



By Bill Childers

Introducing *The DevOps Toolbox: Tools and Technologies for Scale and Reliability* by Linux Journal Virtual Editor Bill Childers.

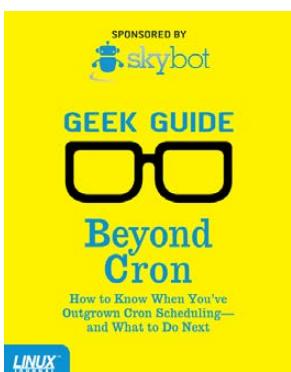
When I was growing up, my father always said, “Work smarter, not harder.” Now that I’m an adult, I’ve found that to be a core concept in my career as a DevOps engineer and manager. In order to work smarter, you’ve got to have good tools and technology in your corner doing a lot of the repetitive work, so you and your team can handle any exceptions that occur. More important, your tools need to have the ability to evolve and grow over time according to the changing needs of your business and organization.

In this eBook, I discuss a few of the most important tools in the DevOps toolbox, the benefits of using them and some examples of each tool. It’s important to not consider this a review of each tool, but rather a guide to foster thinking about what’s appropriate for your own organization’s needs.

Register today to receive your complimentary copy of The DevOps Toolbox:
<http://linuxjournal.com/devops-toolbox-guide>

Beyond Cron

How to Know When You've Outgrown Cron Scheduling—and What to Do Next



By Mike Diehl

If you’ve spent any time around UNIX, you’ve no doubt learned to use and appreciate cron, the ubiquitous job scheduler that comes with almost every version of UNIX that exists. Cron is simple and easy to use, and most important, it just works. It sure beats having to remember to run your backups by hand, for example.

But cron does have its limits. Today’s enterprises are larger, more interdependent, and more interconnected than ever before, and cron just hasn’t kept up. These days, virtual servers can spring into existence on demand. There are accounting jobs that have to run after billing jobs have completed, but before the backups run. And, there are enterprises that connect Web servers, databases, and file servers. These enterprises may be in one server room, or they may span several data centers.

Register today to receive your complimentary copy of Beyond Cron:
<http://linuxjournal.com/beyond-cron-guide>



WINNER



**RULE THE
STACK**
VANCOUVER, B.C.



WHO RULES THE STACK?

SUSE OpenStack Cloud Crowned RULER of the STACK

The "Rule the Stack" competition is a challenge to deploy a fully operational OpenStack cloud as quickly and accurately as possible.

Sponsored by Intel and run at the past three OpenStack Summits, SUSE has won it every time!

At the latest summit in Vancouver SUSE took first and second place, way ahead of the third placed competitor.

SUSE is recognized for delivering trusted, open source solutions that can be deployed quickly and managed easily. Winning this competition perfectly highlights the strengths of the SUSE and OpenStack combination.

CONTENTS

AUGUST 2015
ISSUE 256

PROGRAMMING

FEATURES

62 An Introduction to Tabled Logic Programming with Picat

Picat's high-level features, like tabling and the planner module, allow you to write concise, declarative and efficient programs.

Sergii Dymchenko

74 Hash Tables— Theory and Practice

Let the hash table magic begin!

Mihalis Tsoukalos

ON THE COVER

- Process Tons of Data without Losing Your Mind, p. 88
- Hash Tables: Theory and Practice, p. 74
- Picat: Write Concise, Declarative and Efficient Programs, p. 62
- What's New in 3D Printing Software, p. 40
- Make Your Database Do As Much Work As Possible, p. 34
- Doing Astronomy with Python, p. 26

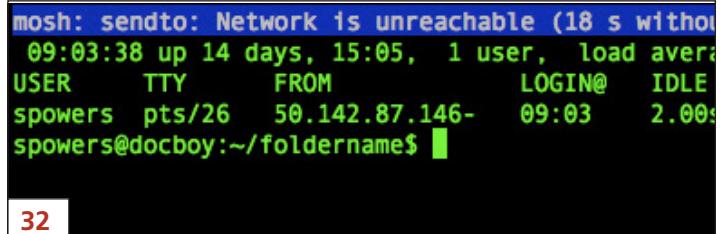


INDEPTH

88 Using MySQL for Load Balancing and Job Control under Slurm

Improving cluster scheduling and load balancing of large dataset processing within Slurm.

Steven Buczkowski



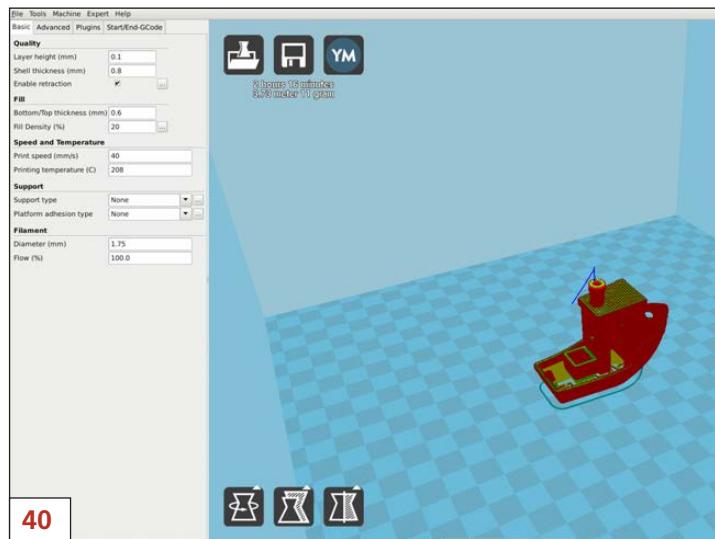
COLUMNS

34 Reuven M. Lerner's At the Forge

Use Your Database!

40 Kyle Rankin's Hack and /

What's New in 3D Printing,
Part III: the Software



46 Shawn Powers' The Open-Source Classroom

Slay Some Orcs!

98 Doc Searls' EOF

Three More Lessons

IN EVERY ISSUE

8 Current_Issue.tar.gz

12 Letters

20 UPFRONT

32 Editors' Choice

58 New Products

105 Advertisers Index



LINUX JOURNAL™

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



ENJOY:

Timely delivery

Off-line reading

Easy navigation

**Phrase search
and highlighting**

**Ability to save, clip
and share articles**

Embedded videos

**Android & iOS apps,
desktop and
e-Reader versions**

SUBSCRIBE TODAY!

LINUX JOURNAL

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

Contributing Editors

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

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

***Linux Journal* is published by, and is a registered trade name of,
Belltown Media, Inc.**

PO Box 980985, Houston, TX 77098 USA

Editorial Advisory Panel

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

Advertising

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

Subscriptions

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

LINUX is a registered trademark of Linus Torvalds.

REGISTER TODAY!

24th USENIX Security Symposium

AUGUST 12-14, 2015 • WASHINGTON, D.C.

The USENIX Security Symposium brings together researchers, practitioners, system administrators, system programmers, and others interested in the latest advances in the security of computer systems and networks. The Symposium will include a 3-day technical program with more than 65 refereed paper presentations, invited talks, posters, a panel discussion on security and privacy research ethics, and Birds-of-a-Feather sessions. Featured speakers/sessions include:

Keynote Address by Richard Danzig, *member of the Defense Policy Board, The President's Intelligence Advisory Board, and the Homeland Security Secretary's Advisory Council*

Invited Talk: "Using Formal Methods to Eliminate Exploitable Bugs" by Katherine Fisher, *Tufts University*

Invited Talk: "Machine vs. Machine: Lessons from the First Year of Cyber Grand Challenge" by Mike Walker, *DARPA*

Invited Talk: "Preventing Security Bugs through Software Design" by Christopher Kern, *Google*

The following co-located events will precede the Symposium on August 10–11, 2015:

3GSE '15: 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education

CSET '15: 8th Workshop on Cyber Security Experimentation and Test

FOCI '15: 5th USENIX Workshop on Free and Open Communications on the Internet

HealthTech '15: 2015 USENIX Summit on Health Information Technologies

Safety, Security, Privacy, and Interoperability of Health Information Technologies

HotSec '15: 2015 USENIX Summit on Hot Topics in Security

JETS '15: 2015 USENIX Journal of Election Technology and Systems
(Formerly EVT/WOTE)

WOOT '15: 9th USENIX Workshop on Offensive Technologies

www.usenix.org/sec15



Stay Connected...



twitter.com/USENIXSecurity



www.usenix.org/facebook



www.usenix.org/youtube



www.usenix.org/linkedin



www.usenix.org/gplus



www.usenix.org/blog



SHAWN POWERS

Welcome to Issue 1000000!

Binary jokes are always fun, and although technically I could say "welcome to issue <Binary_Number>" every month, it's more fun with nice round numbers like 256. Plus, this is our *Programming* issue, and there are 10 types of programmers: those who understand binary, and those who don't.

Reuven M. Lerner starts things off by not only pointing out why we should use databases for data manipulation, but also how and why we should focus on doing so efficiently. Databases, SQL databases specifically, often are a performance bottleneck in applications, but that doesn't mean databases aren't incredibly useful and efficient. If we use them correctly, they will do what they do best: manipulate data for us.

Kyle Rankin guides us into the murky world of 3D printing software.

With the popularity of 3D printing on the rise, so goes the proprietary interests in the technology. Trying to keep the world of 3D printing as open as possible is challenging, but thankfully there's still hope. Kyle covers a few open-source options for printing and design sharing, and sets the stage for his concluding article next month, when he'll provide an Octoprint tutorial.

My column this month is helpful as well, but in a very different way. I was on vacation in the Smoky Mountains when I wrote this article, and rest and relaxation were on my mind. So I decided it was a good time to help everyone relax, nerd style. Whether you're a casual puzzle-gamer or a hardcore Orc-slayer, I tried to cover some of my favorite options for wasting away your vacation time behind a screen. (Note: any wasting of time at work should not be blamed on me, but rather on "burn-in testing" for your workstation hardware!)

Logic programming is vital for



VIDEO:

Shawn Powers runs through the latest issue.

Hash tables are incredibly powerful programming constructs, especially for storing unsorted (or unsortable) data.

many applications from artificial intelligence to creating mazes for video games. Sergii Dymchenko introduces Picat this month, which is a programming tool for testing and programming with logic. If you're familiar with tools like Prolog, Picat will be a pleasant step up, but even if you've never dealt with tabled logic before, Sergii's article will help you (and your programs) think like a Vulcan in no time. It's only logical!

Hash tables are incredibly powerful programming constructs, especially for storing unsorted (or unsortable) data. Ideally, the hash tables will make it easy to retrieve the data in your program and help manage seemingly unmanageable chunks of data as efficiently as possible. Conceptually it can be hard to wrap your brain around, but Mihalis Tsoukalos helps us understand and implement hash tables in our programs. If you're finding yourself needing arrays full of arrays in order to handle your variables, be sure to read his article!

Steven Buczkowski takes us into the world of Slurm. (For you *Futurama* fans, I assure you no

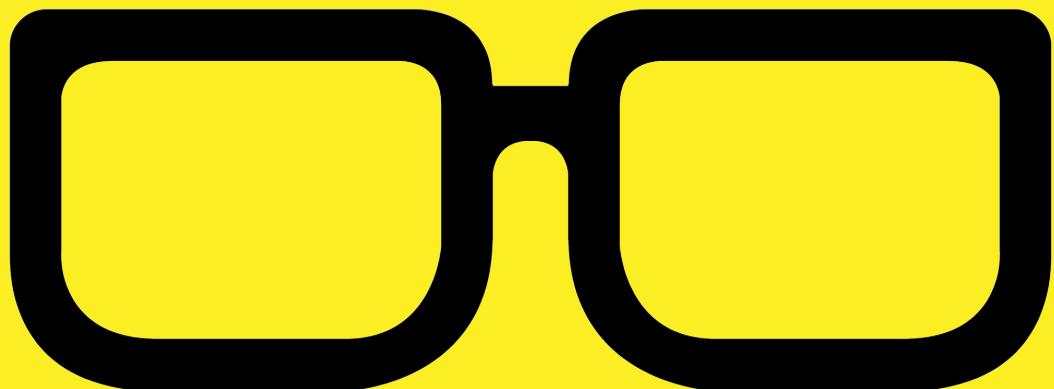
Wormulon queens were abused in the making of this article.) Specifically, Steven explains how jobs are managed in a Slurm cluster and how utilizing a MySQL database for controlling those jobs is an ideal and efficient way to do it. If you've moved from manually load-balancing your jobs to a full-blown cluster environment, Steven's article will help jumpstart your methodology.

If you're not a programmer (or not a *Futurama* fan), this issue still has plenty to keep you entertained and informed. We have new product announcements, tech tips and tons of features for any open-source enthusiast—programmer or not. And if you're not careful, this issue might turn you into a programmer as well! Heck, if you got my binary joke earlier, you're already on your way to the twisted mindset of a programmer! ■

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.

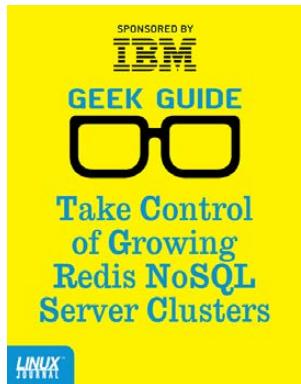
Practical books
for the most technical
people on the planet.

GEEK GUIDES



Download books for free with a
simple one-time registration.

<http://geekguide.linuxjournal.com>



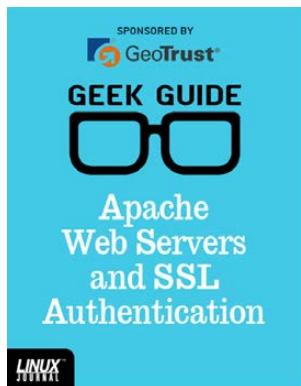
Take Control of Growing Redis NoSQL Server Clusters

Author:
Reuven M. Lerner
Sponsor: IBM



Linux in the Time of Malware

Author:
Federico Kereki
Sponsor:
Bit9 + Carbon Black



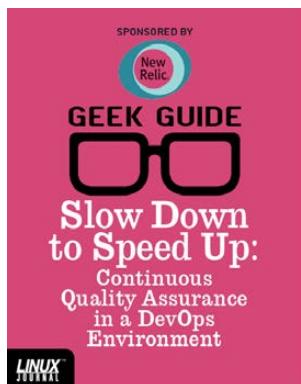
Apache Web Servers and SSL Encryption

Author:
Reuven M. Lerner
Sponsor: GeoTrust



Build a Private Cloud for Less Than \$10,000!

Author:
Mike Diehl
Sponsor:
Servers Direct and Seagate



Slow Down to Speed Up: Continuous Quality Assurance in a DevOps Environment

Author: Bill Childers
Sponsor: New Relic



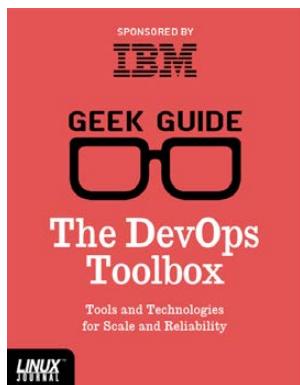
Enterprise Monitoring Success

Author:
Mike Diehl
Sponsor:
InterMapper



Beyond Cron: How to Know When You've Outgrown Cron Scheduling—and What to Do Next

Author: Mike Diehl
Sponsor: Skybot



The DevOps Toolbox: Tools and Technologies for Scale and Reliability

Author:
Bill Childers
Sponsor: IBM

letters



Work the Shell Column, June 2015

In “When Is a Script Not a Script?”, Dave Taylor cautions that a loop like `for f in *` would have problems handling filenames with spaces—for example, “hello world”. However, field separator processing is not performed on the generated strings and `f` would be properly set to “hello world”.

In the two `find` examples, the semicolons that terminate the `-exec` clauses should have been quoted. Most likely a backslash was lost between the author and the printer. In the first example (`-exec chmod`) the “plus” terminator would have been a good alternative that would not require quoting. Because the “+”

terminator aggregates filenames for fewer command executions, it would not be a good alternative for the second example (`-exec echo`).

Dave’s use of the `file` command to classify files with executable permission is laudable. However, Dave’s shell bias (shared by myself) comes through as his code does not account for other scripting languages, such as Awk or Perl.

The `file` command looks for “magic strings” to identify the file type. The strings it knows about are listed in a file appropriately named “magic”. It is located in `/usr/share/misc` on my system and has its own man page, `man -s 5 magic`. My version of `file` can identify about 20 different scripting languages. They all report with the words “script” and “executable”.

Compiled programs are also understood by `file`. Rather than looking for matching C source files, Dave could have checked the file description for the words “executable” and “ELF” (Executable and Linkable Format).

—**Jon LaBatie**

Dave Taylor replies: You raise some good points here. Thanks for writing!

Scanning on Linux Distributions

Regarding Alan Lewis' letter in the June 2015 issue about printing and scanning, I have found that Hewlett-Packard multifunction printers combined with HP's excellent (free-to-a-good-home) HPLIP software fully supports all printer functions over many years and three or four HP printers. Currently, I have an HP OfficeJet Pro 8600: Ethernet interface, print, automatic duplex print, scan, copy and fax (built-in telephone connection). Everything works, works the way I expect, and I have zero problems. I also have an older (about 12 years) HP Business Inkjet 2280tn with two trays and Ethernet that is fully supported (it doesn't do much but print) by HPLIP.

I bought a Canon Pixma iP 6310D a few years back. Zero Linux support from Canon. It's a doorstop, and it's free to a good home if somebody wants to pay the shipping.

HPLIP is generally included with most Linux distributions (mine is Slackware). HP fixes any problems

quickly (and adds new printer support), and installs and integrates with CUPS quickly and easily—just run hp-setup and away you go.

Why go anywhere else, particularly manufacturers that don't provide Linux support?

—Thomas Ronayne

Although I agree HP is usually good about Linux compatibility and Canon is horrible, it's still always a good idea to check before buying.—Shawn Powers

Typo in the June 2015 Issue

The find command at the bottom of page 11 in the Letters section is wrong [see the "Pipes and Xargs" letter]:

```
find / -name "*mp3" -exec rm '{}' \;
```

It should be this:

```
find / -name "*.mp3" -exec rm '{}' \;
```

The quotes must match.

—Alan Olsen

You are, of course, correct. I suspect Richard's right pinky is a little quicker than his left, and Shift wasn't pressed in time! (As far as me not catching the

error, sadly I don't have a crafty excuse for that.) Thanks, Alan.—Shawn Powers

A New Algorithm

The following may be of interest to your readers:

/* AN ALGORITHM TO FACTOR PQ PRIMES

The following is an algorithm to factor pq into the individual primes p and q that were multiplied together to form pq. It is not required to know any primes in the beginning and no divisions are required.

Input the number pq to begin and take the square root of pq.

Ensure the square root is negative by subtracting 1 if required.

Set the variables p and q equal to the negative square root.

Decrement p by 2 and increment q by 2.

Set up a continuous loop.

Set variable a equal to p*q.

Set up for/next loop which will terminate when a becomes greater than pq.

Within the for/next loop:

Add p to a.

Increment q.

When a and pq are exactly equal terminate the for/next loop and goto end of program.

When the for/next loop terminates because a is greater than pq decrement q by 1 and decrement p by 2 then goto the beginning of the continuous loop.

When a and pq are equal and the program terminates

p will be the original p prime used to form pq and q will be the other prime. */

```
#include <stdio.h>
long long i,pq,x,y,sr,a,m,p;
int main(void)
{
    /* pq = 299; */
    /* pq = 989; */
    /* pq = 7957; */
    /* pq = 16972174847; */
    /* pq = 52364476296131; */
    pq = 2549809323176597;
    /* pq = 29837383766136799; */
    /* pq = 303703809540747433; */
    /* pq = 1043931805398125881; */
    /* pq = 2786765675348933501; */
    /* pq = 1844674407370955161711; */

    printf("pq = %llu\n",pq);

    /* Routine to calculate sr, the square root */
    m = sr = p = 1;
    while(m <= pq) { p += 2; m += p; sr++; }
    if((sr % 2) == 0) sr--; /* If the square root is even,
                                * make sure it is odd */

    x = y = sr;

    printf("sr = %llu\n", sr);

    /* Adjust x and y to skip the first loop iteration since
       * a square area cannot equal a rectangular area */
    x -= 2;
    y += 2;
    Loop:
```

```

a = x*y; /* Set a equal to the area to expand
           * toward the area of pq */

/* The for loop expands area a toward area pq by
   * multiple values of x until either they are equal
   * or area a is larger than pq */

for(i = 0; a < pq; i++) {
    a += x;
    y++;
    if(a == pq) goto End;
}

/* If a and pq are not equal, decrement x by two for
   * the next try and remove the excess y count
   * caused by the for loop */

y--;
x -= 2;

goto Loop;

```

End:

```

printf("The rectangle pq has prime sides %llu and %llu.\n",x,y);
return 0;
}

```

—Glen A. Dobbs

Glen, I hope I wasn't just "Rick Rolled", because your math and scripting are beyond my ken. I'm sure others will appreciate the script though. Thank you!—Shawn Powers

A General Issue with GUIs

Some GUIs, before initiating an action, present the user with check boxes for "OK" and "Cancel", in that order. Others in the same situation present the user with boxes for "Cancel" and "OK", in that order. Once in a while, out of habit, I punch the unintended box. This could be a Very Bad Thing if I meant to punch "Cancel" and instead punched "OK". This is one case where we need an autocrat to dictate that only a specific style may be used.

—Jim Haynes

I totally feel your pain, Jim! I have no solution, nor do I have the global influence to fix such things, but there should be an IEEE standard for dialog boxes!—Shawn Powers

Scanning Tips for Linux

I may have part of a solution for Alan McConnell's difficulties with his Canon scanner (see the Letters section in the June 2015 issue). Based on the description of the problem that he sent to Canon, I suspect he has permissions issues (a problem I have become quite familiar with using my HP printer/scanner). Try changing the ownership of the device files located at /dev/bus/usb/xxx/yyy where you replace xxx and yyy by the two numbers shown after

libusb (in Alan's post, that would be 003 and 004). By default, these devices are owned by root; you need to have ownership of the files yourself (unless you don't mind making them world-read and writeable, which is probably a bad idea). Once you have fixed the ownership, running `sane-find-scanner` should return the human-readable version of the vendor and product information. Hope this helps!

—Greg Mahon

Erratum

In the June 2015 issue, in my "Doing Stuff with Docker" article, I wrote this:

```
sudo docker run -d --net="host" \
-v /mnt/docker/plex:/config \
-v /mnt/videos:/data \
-p 32400:32400 \
timhaak/plex
```

But, it should be this:

```
sudo docker run -d --net="host" \
-v /mnt/docker/plex:/config \
-v /mnt/videos:/data \
-p 32400:32400 \
timhaak/plex
```

(There should be slashes after the colons.)

I apologize for any frustration.

—Shawn Powers

Linux Drivers for Canon and Other Unsupported Scanners

VueScan, a non-FOSS, non-free (\$75) scanner program, may restore functionality to Canon and other scanners lacking drivers for Linux (or Windows 7). I've used the program to restore functionality to a Nikon LS-5000 slide scanner at work (unsupported in Windows 7) and an older Epson scanner at home (no Linux driver). The program was easily worth the \$150 for the two licenses. Free trials are available from the publisher: <http://hamrick.com>.

I am pleased that the Linux desktop market is large enough to attract developers. Most of the software I use is free and open source. There is also room on my Linux desktop for non-free, closed software.

Update: I just double-checked and see that the price of the professional license for VueScan has gone from \$75.00 to \$89.00 and includes free upgrades for life. The software supports Macs as well as Linux and Windows.

—Steve Johnson

I can second Steve's support for VueScan. The devices it supports work very, very well with it.—Shawn Powers

Re: June 2015 Letters— Canon MF4770n and SANE

The sane-pixma back end is supposed to support the Canon MF4770n on USB and Ethernet. SANE's Web site says the status is "good", which I assume means that SANE works with the MF4770n, but not all of the device's features are available.

—Neal

January 2015 Issue

I admit that I am way behind on my reading, but I just finished the January 2015 issue, and I am excited to try some things.

I read Jeremiah Bowling's article "How to Perform an Internal Security Review", and I was thinking that it would be a cool thing to try just on my home network.

Then it occurred to me that in the same issue Shawn Powers wrote "Vagrant Simplified", and then the light bulb went on for me. I went looking, and sure enough, there is a Vagrant box for Kali Linux.

I haven't gotten any further than that, but the Vagrant box is downloading now, and soon I expect to be able to follow Jeremiah's steps in my own environment.

Thanks for a great set of articles!
—Werner

Awesome! I love when our various articles complement each other. Have fun, Werner!—Shawn Powers

Erratum, II

First, congratulations for the great work! In Shawn Powers' "Gettin' Sticky with It", in the June 2015 Upfront section, he uses the example `ls -l /bin/ping` to show the SUID set in the permissions string, but in Fedora 22 and CentOS, it isn't there, because it's not defined by a permission, but by a capability. In the `ping`, if you use the `getcap` command, it will return the following capabilities:

`cap_net_admin, cap_net_raw+ep`

For more info about them, see `man capabilities`. Thanks!

—Adail

Gosh darn progress! Back in my day, our binaries were SUID, and we liked it! Seriously though, thanks for the info. I didn't check all major distros, and I probably should have done so.—Shawn Powers

More on "Gettin' Sticky with It"

In response to the question that Shawn Powers had about the sticky bit [see

the June 2015 Upfront section], its original use had no effect on directories (folders). Instead, it was used on executable files (compiled programs, to be specific), so that when the program was run, its text area would “stick” in the swap space. So “t” stands for “text”. This also explains the origin of the term “sticky bit”.

—Mark Iszler

Cool info Mark, thanks!—Shawn Powers

Hacking a Safe with Bash

I know it’s late June and I’m responding to an article from the May 2015 issue—sometimes I have more magazine than month and I get behind—but I wanted to comment on a few things that jumped out at me when reading this fine article [see Adam Kosmin’s “Hacking a Safe with Bash”]. I did learn a thing or two, and the focus of the article was not shell script programming, but ...

I write a lot of shell scripts in my job and off hours, and when I write a production-level script, my goal is portability and safety.

1) Even though the article was focused on using the Bourne-Again shell, there was nothing there that really needed any of that shell’s

advanced features, and I would opt for the Bourne shell (/bin/sh), which every POSIX-compliant system *must* have. For this script, it would just be translating all `$(...)` constructs to back ticks `...` and a few other changes. On many systems, /bin/sh is either a link to /bin/bash, /bin/dash or /bin/ash, which all understand the Bourne shell syntax as a subset.

- 2) The Bourne shell (and Bash) has special `${} constructs for setting a default value if the given environment variable doesn’t exist—for example, SOURCE_DIR=${SOURCE_DIR:-$HOME/safe}.`
- 3) Instead of tar, I would use cpio, which works “natively” with `find` and `xargs`, and is designed to archive a list of files passed in via `stdin`. I would also use the `-0` for each, which is a useful GNU-ism for delimiting the filenames with the NULL character and allows the use of filenames with spaces.
- 4) I notice a number of “rm”s after some action is performed. This is probably the thing that caught my attention the most. If the action failed, the script would be wiping out the safe directory or file contents, losing everything! In all such cases, I would append the `rm` command to the previous action with the `&&` control

At Your Service

operator or onto a test for the existence of a non-zero file-length safe file or directory existence—something that would give that warm fuzzy feeling that the action succeeded.

5) All the commands used, like `rm`, `tar` and `shred`, are not fully qualified, so the first executable found in the PATH list will be used. This can lead to a hacker placing a rogue executable script to intercept, log or screw with your actions. So I make variables for all the commands I use—for example, `RM=/bin/rm`; `TAR=/bin/tar`; and so on. For executables that may not be in the same location of differing systems, you can quickly test the few likely paths and set the variable accordingly.

I found the article useful, and I'll probably hack my own version soon. Thanks for the fine article.

—R.K. Owen

Adam Kosmin replies: All great stuff. Would you be willing to send a PR (<https://github.com/windowsrefund/safe>)?



WRITE LJ A LETTER

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

PHOTO OF THE MONTH

Remember, send your Linux-related photos to ljeditor@linuxjournal.com!

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

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

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

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

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

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

diff -u

WHAT'S NEW IN KERNEL DEVELOPMENT

Linus Torvalds reported on some **GCC** compiler warnings that he felt were unnecessary, and he gave his opinion on how they should work instead. Specifically, GCC 5.1 would issue a warning against using a `switch` statement with a boolean variable, presumably because a boolean would be better handled by a simple `if` statement.

Linus posted the following counter-example:

```
switch (a) {  
    case 1:  
        A;  
        if (b)  
            break;  
        B;  
        /* fallthrough */  
    case 2:  
        C;  
}
```

And he said:

You share that C case for some conditions but not others.

You can do the same using `goto`, of course, but you can *not* do it with pure nested `if ()` statements.

So even with just two cases, `switch ()` is syntactically more powerful than `if ()`, because it allows more structured exits.

Linus said that a more appropriate warning from GCC would be when the data type of the `switch` variable was different from the data type of the `case` variable. In that case, he said, a warning would make perfect sense. But warning against using a boolean variable in a `switch` at all, he felt, was going overboard. He said that:

```
switch (boolean) {  
    case true:
```

made far more sense than following the recommendation in the GCC documentation of casting to an integer:

```
switch ((int)boolean) {  
    switch 1:
```

He said anyone who preferred the latter over the former “clearly has absolutely zero taste and is just objectively wrong”.

Tadeusz Struk has proposed some patches to implement a new **public key encryption API**. The idea is to have software routines in place when no encryption hardware is available, but to offload the work to hardware when possible. In general though, any module could provide its own **RSA** and **DSA** implementation.

It's not entirely a new concept—the kernel already supported public key encryption for verifying digitally signed driver modules. But, Tadeusz's code is intended to replace the old crypto code. Along with his patches implementing the feature, Tadeusz submitted patches migrating older usages to the new service.

There were some technical suggestions and some minor objections, but ultimately, it does seem as though the old crypto code will be replaced by Tadeusz's new stuff. The biggest problem seemed to be how to make it easy for user code to handle drivers that implemented only a partial subset of the RSA and DSA API.

Initially, Tadeusz planned to allow drivers simply to encode a list of the features they provided, but **Herbert Xu** convinced him to require drivers to implement at least a complete minimal subset of features, so users could rely on them without doing lots of tests.

Filesystem capabilities still have not come into their own. Originally intended as a way to loosen targeted security constraints on user processes without going all the way to the extreme of running as root, the poor initial design of capabilities has sometimes led to more security problems than it solved.

One problem has been capability inheritance—the ability of one process to pass along its same set of capabilities to processes it invokes. This is the equivalent of user processes running sub-processes as that user, or root processes running sub-processes as root—something UNIX always has supported. But capabilities haven't implemented that feature properly, and the available workarounds have tended to make it easier for hostile users to gain root privileges on a system.

Recently **Andy Lutomirski**, working off ideas by **Christoph**

Lameter and **Serge Hallyn**, produced some patches that re-envisioned capability inheritance in a way that, without breaking current usage, they claimed would provide a saner and more secure form of inheritance.

Capabilities work by identifying secure abilities that are available to a given process. A process has a certain set of capabilities, which can then be masked off, leaving only the ones that actually will be needed by that process. One of these masks is the “inheritable” mask, called **pl**. The pl mask is supposed to control which capabilities are inheritable by sub-processes. The problem is that it doesn’t do that properly, although changing its behavior could break existing user code.

Andy’s code got around this dilemma by introducing a new **pA** mask, which he said would do what pl should have done originally. The pA mask, Andy said, would introduce new logic and some nuanced behaviors to allow a predictable form of inheritance that would avoid the security exploits that had been plaguing users.

Several folks pointed out, and Andy acknowledged, that this wasn’t a perfect solution and left certain problems unsolved. But the bottom line, he said, was that his code represented a real improvement and better overall direction in a situation where no one else was able to offer any alternatives beyond the status quo.—**ZACK BROWN**

They Said It

If people only knew how hard I work to gain my mastery, it wouldn't seem so wonderful at all.

—**Michelangelo Buonarroti**

Real freedom lies in wildness, not in civilization.

—**Charles Lindbergh**

Things are only impossible until they're not.

—**Jean-Luc Picard, Star Trek the Next Generation**

The thing about chameleoning your way through life is that it gets to where nothing is real.

—**John Green**

The only way most people recognize their limits is by trespassing on them.

—**Tom Morris**

containercon

Sheraton Seattle

August 17-19, 2015

events.linuxfoundation.org/events/containercon

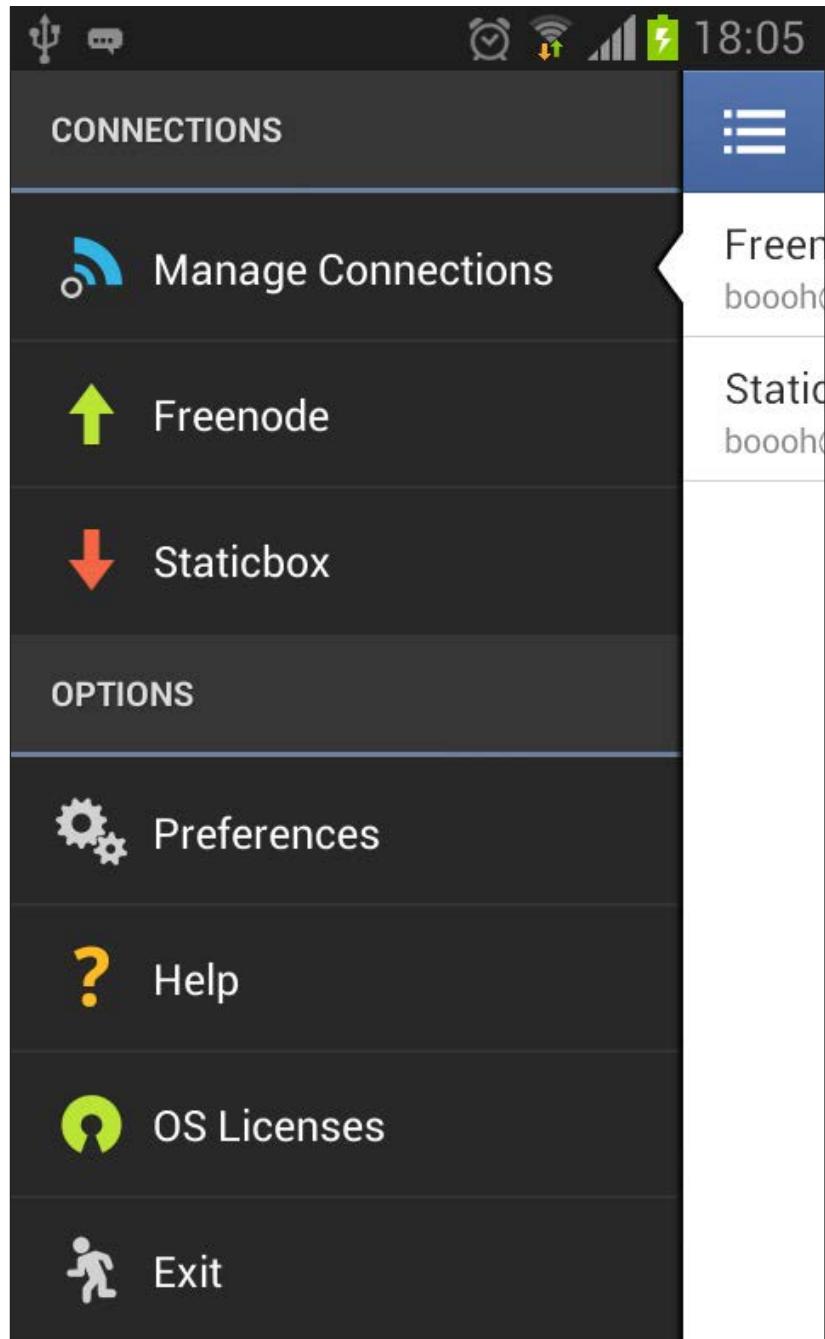
Android Candy: Rice IRC

If you're excited to try the ZNC IRC bouncer I highlighted in another UpFront article this month, you really should test its flexibility using an Android IRC client. I've never actually used IRC on Android, because connecting temporarily isn't really how IRC works best. And, the thought of staying logged in via my phone's data plan sounds unpleasant.

I tried a few IRC clients for Android, and although most are functional and intuitive, Rice IRC stood out above the rest. It's completely free, has no ads, and it has an interface that makes perfect sense. It also has all the features you'd expect in a mobile chat client, including notifications, background tasks and so on.

An IRC client is one of those apps where simplicity is better. IRC is simple, and so should be the interface you use to connect with it. Rice IRC ticks all the boxes. Get your copy today from the Google Play store: <https://play.google.com/store/apps/details?id=it.mneri.android.rice>.

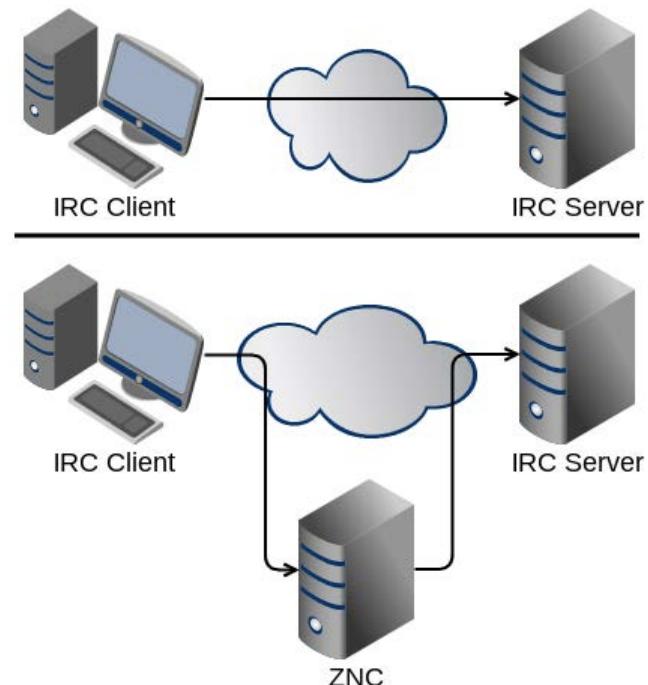
—SHAWN POWERS



Bounce Around IRC with ZNC

In my discussion on IRC with “bkidwell” (see the Non-Linux FOSS article for more on our talk), we were discussing how we connect to IRC. My main method is to SSH in to my co-located Raspberry Pi in Austria and connect to a screen session I have running that is constantly connected to IRC with Irssi. This works really well for me, and I never miss messages when I’m away. The big problem I have is occasionally I’m away from a laptop, and so I can’t efficiently use the terminal to chat. It might be technically possible to IRC chat via an SSH app on my phone, but it would mean super-tiny text and awkward keyboard shortcuts.

That’s where ZNC comes into play. I’ve never used an IRC bouncer before, but the concept is that it stays connected to IRC using your credentials, and it allows you to connect to it with your IRC app and pick up where you left off. Functionally, it’s like a screen session, but instead of connecting via SSH, you connect with an IRC client. Since it keeps you logged in, when you connect, you have access to the chats and personal queries that have occurred while you were disconnected. Plus, it



(Image from <http://znc.in>)

allows you to connect from multiple locations at once and use whatever IRC client you want—there’s no “client” application you have to use!

ZNC is really an awesome way to use IRC. It allows me to keep using Irssi too, because I just point Irssi at the ZNC bouncer, and never miss a beat even when I’m remote on my phone. If you use IRC regularly or are hesitant because of the issues I’ve highlighted here, I urge you to check out ZNC. It’s at znc.in and is available for most platforms.—**SHAWN POWERS**

Doing Astronomy with Python

One of the things that makes Python so powerful is that you can find a module for almost anything. In this article, I cover Astropy, which was originally developed by the Space Telescope Science Institute for doing astronomy calculations like image processing and observatory calculations. Because this is a Python program, you can install it with either pip or easy_install. Your Linux distribution already should have a package included. For example, in Debian-based distributions, you can install it with this:

```
sudo apt-get install python-astropy
```

There is also a separate package, python-astropy-doc, that contains extra documentation for Astropy. Because Astropy is a fairly large system, it is broken into smaller sub-packages. This should be familiar to anyone who has worked with packages like SciPy or NumPy before. So, simply using the following actually isn't very useful:

```
import astropy
```

You more likely will need to import individual sub-packages with commands like this:

```
from astropy.io import fits
```

There are sub-packages to handle file IO, cosmological calculations and coordinate systems, among many other topics. Let me provide a basic introduction to some of the available functionality so you have an idea of what you can do.

First, let's look at how to deal with data files. The common file format used in astrophysics and astronomy is the FITS file format. The PyFITS Python package was written to read and write FITS files. This code is actually the same as the code in the sub-package astropy.io.fits, so you can use it in the same way. You actually can even just drop Astropy in as a plugin with the following:

```
from astropy.io import fits as pyfits
```

This way, you can use existing file management code without having to make any changes.

The first thing you need to do is open your data file with:

```
from astropy.io import fits
hdulist = fits.open("My_File.fit")
```

This returns an object that behaves like a list. Each element of the returned object maps to a Header-Data Unit (HDU) in the file. You can get more information on the file with this command:

```
hdulist.info()
```

Each of the individual elements has a header and data portion. You can access them to see details about the data you are about to process.

Along with all of the library functions, Astropy includes a series of command-line utilities to work with FITS files. You can check the headers of a FITS file with the `fitsheader` utility. You can check your FITS file with `fitscheck`, and you even can find the differences between two files with `fitsdiff`.

A common computational process in astronomy is image processing. The convolution sub-package provides two categories of convolution operations: direct and FFT. You can do one-, two- and three-dimensional convolutions. The visualization sub-package handles

more basic image processing like normalization and stretching. You can combine multiple transformations very easily. The + operator is overloaded to apply transformations that are “added” together in series. So, a command like this:

```
transform = SqrtStretch() + PercentileInterval(90.)
```

gives you a new function, `transform`, that combines the two separate transformations in a single step. This sub-package also includes a script, `fits2bitmap`, that can do conversions between different file formats.

A second common computational task in astronomy is doing statistics based on observations, and Astropy provides a sub-package called `stats`. Although the `scipy.stats` sub-package provides a lot of functionality, some astronomy-focused functions are missing, so the `astropy.stats` sub-package fills in those missing functions.

Once you have your data loaded, you can use the modeling sub-package. You can do 1D and 2D modeling with `astropy.modeling`. This includes curve-fitting functionality, where you can do linear and nonlinear function fitting. There are built-in functions to fit Gaussian curves and polynomials. This fitting is handled

with a least-squares method. With version 1.0, you can build compound models by combining existing models with arithmetic operators.

When you are ready to start doing calculations, you will need to use constants. Instead of remembering them or using them with potential typos, Astropy includes a complete list of all the standard scientific constants that you will need when doing numerical work. You can import the entire list with this:

```
from astropy import constants
```

If you need only a few of the constants, like maybe the speed of light, you can import them individually with this:

```
from astropy.constants import c
```

The really cool thing about these constants is that they are actually “Quantity” objects. This means you can do something like change the units being used with a command like the following:

```
c.to('km/s')
```

Because it is so prevalent, you can use CGS units with `c.cgs`.

There are also two sub-packages

to handle coordinate systems. Astronomical coordinate systems are handled by the coordinates sub-package, and world coordinate systems are handled by the wcs sub-package. In the coordinates sub-package, the core object is the `SkyCoord` object. There are methods of this object to handle conversions between coordinate systems or distances from one point to the origin within a given coordinate system. The wcs sub-package allows for mapping data within a FITS file onto a “real-world” coordinate system in order to analyze them correctly. This includes functionality to deal with complications, like projections onto the sphere of the sky.

You even can do cosmological calculations with Astropy. The cosmology sub-package actually includes functionality to model the evolution of the entire cosmos based on a set of initial conditions that you set. Although you can set your own cosmology, several built-in cosmologies are available. These are based on the WMAP and Planck satellite data.

Most functionality is built off a core FLRW object. This object represents a homogeneous, isotropic cosmology defined by the Friedmann-Lemaître-Robertson-Walker metric

from General Relativity. However, this class can't be used directly. You need to subclass it first. There are several included in the cosmology sub-package, such as the FlatLambdaCDM object that includes dark energy. You can look at various values, like the Hubble constant, at various points during the evolution of the cosmology. You also can include contributions to the energy density from matter, dark energy and even photons and neutrinos.

Now that you've seen a bit of

what you can do with astropy, if astronomical calculations are on your radar, there is much more available. Additionally, there is the concept of affiliated packages. These are packages that basically are built on top of the core functionality provided by Astropy. Although they aren't part of Astropy, they are being built up into a community-driven environment for doing astrophysics. It definitely will be worth your while to take a look at the extended world of available packages.—**JOEY BERNARD**



Powerful: Rhino

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

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



Tablet: Raven

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



Rugged: Tarantula

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

EmperorLinux
...where Linux & laptops converge

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

Model specifications and availability may vary.



My Network Go-Bag

I often get teased for taking so much tech hardware with me on trips—right up until the Wi-Fi at the hotel, conference center or rented house fails. I'm currently on vacation with my family and some of our friends from Florida, and our rental home has a faulty Wi-Fi router. Thankfully, I have a bag *full* of goodies for just this occasion. I don't really have a set "list" of items I carry, but generally I'll have the following:

- Several Ethernet cables, various lengths.
 - A plug-in-the-outlet Wi-Fi extender.
 - A USB-powered Wi-Fi router/bridge/AP/extender.
 - Extension cables.
 - Large external battery with USB charging port.
 - Tablet (to look for Wi-Fi SSIDs, channels and signal levels).
 - RJ-45 crimper and ends (for fixing poorly crimped cables).
- The USB-powered options are really due to a single incident where I had to put a Wi-Fi repeater in the middle of a field in order to reach a remote building. I tied the repeater and battery in a double-wrapped grocery bag and charged the battery every couple days. It felt like the wild wild west of networking.
- As for our current vacation, my router is working well, and the third floor of the house has Internet access thanks to the repeater I installed on the second floor. At least for the duration of this trip, I don't expect to be teased for bringing so many nerdy accoutrements.

—SHAWN POWERS



AUGUST 21 - 22

Featured Speakers

Keynote: Joan Touze - Apache Software Foundation

Ken Starks - ReGLUE

Ted Gould - Ubuntu

Thomas Cameron - RedHat

Major Hayden - Rackspace

texaslinuxfest.org



Non-Linux FOSS: Flaky Connection? Mosh it!

Most of the work I do on computers is done via the command line. When I'm off on vacation somewhere, that means shoddy Wi-Fi and cell-phone tethering. Because cell-phone tethering gets expensive quick (I also have three teenage daughters with which I share a data plan), I try to use free Internet whenever I can. The biggest hassle with that method is dealing with broken SSH sessions. Yes, I can use programs like screen or tmux to make sure I don't lose work, but it can be very frustrating to have an SSH session lock up because the "TotallySafeBro" SSID in my hotel goes down. And, don't get me started on lag.

That's where Mosh comes into

play. I was lamenting in IRC, and user "bkidwell" mentioned Mosh as being a great terminal client for questionable or often-changing connections. It uses a combination of TCP over port 22 and UDP on a higher port to provide a smooth, flexible terminal session regardless of your connection reliability or performance. Rather than waiting for the remote server to echo what you type, Mosh displays your local typing and edits in real time, then "catches up" with the server as quickly as it can. The coolest part, however, is that Mosh will keep your connection alive and running even if you change network addresses! Switching from McDonald's

```
mosh: sendto: Network is unreachable (18 s without contact.) [To quit: Ctrl-^ .]
09:03:38 up 14 days, 15:05, 1 user, load average: 0.01, 0.02, 0.05
USER        TTY        FROM          LOGIN@        IDLE      JCPU      PCPU WHAT
spowers     pts/26    50.142.87.146-  09:03      2.00s  0.06s  0.00s w
spowers@docboy:~/foldername$ █
```

Figure 1. Mosh detects a disconnect and keeps trying to reconnect, even if your IP address changes!

Wi-Fi to your phone's shared data? No need to log out and back in.

Admittedly, Mosh is lacking in some ways compared to SSH. It doesn't do port forwarding, it doesn't keep a scrollback buffer, and its predictive text is sometimes wrong. That last one is most noticeable on a really poor connection, and it isn't really a problem—the screen just occasionally changes a bit when the server/client syncs up. It's still far nicer than typing eight lines of

instructions, only to see a typo when the screen finally updates.

Due to its flexibility with poor connections, cross-platform server/client availability and convenience for mobile admins, Mosh gets this month's Editors' Choice award. Download the application from your Linux repository, or head over to <https://mosh.mit.edu> for download links and instructions for whatever platform you might be on, including Android. Installation is simple, and the benefits are immediate!—**SHAWN POWERS**

LINUX JOURNAL

on your
Android device

Download the app now on
the [Google Play Store](#)

www.linuxjournal.com/android



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



REUVEN M.
LERNER

Use Your Database!

Objects are great, but your database is the best place to do serious manipulations on your data.

I **love** high-level, dynamically typed languages, such as Python, Ruby and JavaScript. They're easy—and even fun—to use. They let me express myself richly, and they lend themselves to code that easily can be reused and maintained. It's no surprise that interest in such languages is on the rise, especially when creating Web applications.

Now, one of the downsides of these languages is that they tend to execute more slowly than static languages, such as Java, C# and Go. But for a very large number of Web applications, this speed difference doesn't matter, or it is justified by the productivity gain enjoyed by the engineers, or it can be (somewhat) handled by throwing hardware at the problem.

However, the fact that dynamic languages run more slowly than their static counterparts doesn't mean you want to ignore the speed issue

completely. Once you know your way around dynamic languages, as well the frameworks built in them, you get a sense of what runs quickly and what doesn't.

In just the past few weeks, however, I've encountered a pattern—or perhaps I should say, an "anti-pattern"—in the code that several of my consulting clients had written. This anti-pattern is well known to experienced developers, but it seems to be less well known than I would have hoped or expected. That anti-pattern, stated simply, is that you should have the database do as much work as possible.

There are several reasons for throwing as much as possible at your database server. First and foremost, your database almost certainly is written in C, so it's likely to execute more quickly than your high-level, dynamic code.

Second, your database has been

Part of the reason, I believe, for the widespread inefficiencies in people's database queries is that they don't see the SQL they're writing, so they don't understand some of the implications of their method calls.

highly optimized through the years, such that retrieving data from it has been tuned to take into account memory, disk and the frequency of the retrievals.

Third, although network bandwidth is cheap nowadays, it's not infinitely fast. This means that although you could, in theory, write a database query in Ruby that returns a large number of rows and then filters through them using `Enumerable#map`, if you have the database do some of this for you, it can reduce the amount of data you're retrieving dramatically and, thus, lead to faster application responses and less network usage.

So in this article, I explore this anti-pattern of doing work in an application that probably should be done in the database. You'll see how you can get the same results, but much faster, by applying this rule. There's obviously no one right way to do things, but having the database do as much work as possible is likely

to make your applications faster and easier to maintain.

Don't Load Everything

High-level languages, and most high-level Web frameworks, don't encourage you to write SQL directly. Rather, you use objects and methods to work with the database; the methods you invoke are translated into SQL by an ORM (object-relational mapper). Part of the reason, I believe, for the widespread inefficiencies in people's database queries is that they don't see the SQL they're writing, so they don't understand some of the implications of their method calls.

For example, say I'm working on a project in Django. If I have a model named `Person`, I can (and should) invoke the "objects" method in order to work with the corresponding table in the database. I then can take the resulting object and apply additional filters, getting (for example) the records corresponding to people who

are system administrators:

```
>>> admins = Person.objects.filter(admin=True).all()
```

Once I've done that, "admins" will contain a set of records, known in the Django world as a "QuerySet". But in actuality, a QuerySet doesn't contain the records themselves. Rather, it serves as a go between to the database. If you iterate over the QuerySet, you'll get each of the records, one by one.

Thus, even if you'll eventually get one million records back from the database, the above code doesn't retrieve them. You can get the records, one by one, by iterating over the result set. For example, the following will display the user names for all of the administrators:

```
>>> for admin in admins:  
    print(admin.username)
```

This is the right way to work with objects in Django. Although it might seem weird not to have the entire result set in memory, the implications are tremendous. You don't need to worry about using up all of the server's memory if the resulting records will be too large.

Working with iterators is easy and straightforward, if you're used to it. If

you're not, it might seem strange not to have the entire result set at once, and to iterate over it. Moreover, all you need is the right combination of a result set and the following code:

```
>>> admins = list(Person.objects.filter(admin=True).all())
```

Notice what I've changed in the above assignment? I'm no longer asking for the QuerySet, over which I can iterate. Rather, I've asked for the QuerySet's data to be used to create a list and then assigned to "admins". If you have one million records in your result set, this is going to consume a fairly large amount of memory.

It's true that this can be necessary, at times, but those times are fairly rare. After all, the odds are pretty good that you're retrieving the records in order to display them to users, something that is easily and efficiently accomplished with iteration.

Filtering

Let's say I'm interested in displaying all of the administrators on my system. Above, I showed that you can do that with:

```
>>> admins = Person.objects.filter(admin=True).all()  
>>> for admin in admins:  
    print(admin.username)
```

However, there's a variation of this that I've often seen people do:

```
>>> people = Person.objects.all()
>>> for person in people:
    if person.admin:
        print(person.username)
```

Notice what I'm doing here: I'm retrieving all of the objects and then iterating over them. Then, I use an "if" statement in Python to determine whether I want to print the user name. If you're used to working with Python objects, this seems like a perfectly natural thing to do.

However, let's consider what's actually happening here. You're retrieving all of the records and using only a small number of them. This means the database is being forced to read through all of its records, bring all of them into memory and send those records to the Python application—even though the odds are that only a small proportion of these records will be printed.

Moreover, while the "if" statement in Python definitely is quite efficient, there is still some overhead to the lookup of the `person.admin` attribute, not to mention the creation of a new "Person" object for each record you got back from the database. In other words, you're

creating a huge number of Person objects just to display some output.

It's far, far more efficient to do your filtering in the database and create Python objects only for the records that you're most likely to want to display. The database, if defined correctly, has indexes that it can use to speed up the query if you tell it to filter records such that it consumes less memory, less CPU and less network bandwidth.

I've seen a variation on this anti-pattern in that people sometimes want to perform transformations on data that they have retrieved from the database. For example, let's assume that I want to apply a sales tax of 10% on all of the prices in a set of records. I certainly could say:

```
>>> products = Product.objects.all()
>>> for product in products:
    print(product.price * 1.10)
```

But it'll be faster and require less Python code, if I simply say:

```
>>> products = Product.objects.raw('select id, price * 1.10 as
>>> price_with_tax from store_product')
>>> for product in products:
    print(product.price_with_tax)
```

Notice how the use of `raw` allows you to go behind the back of Django's ORM, using whatever SQL you want.

So if you're counting, sorting or grouping, you shouldn't need to step down to the SQL level.

Is this the way you always want to do things? Surely not. But in specific cases, or when you want to use a function, it definitely can come in handy. Note that the object you get back from the call to `raw()` is a `RawQuerySet`, which is an iterator just like the regular `QuerySet`. However, it lacks an `all()` method, which is just as well, given that the `RawQuerySet` is already an iterator, giving the appropriate records when requested (and not before).

Note that for commonly used SQL functions, such as `COUNT`, there are built-in Django methods that handle such things. So if you're counting, sorting or grouping, you shouldn't need to step down to the SQL level. And as a general rule, you don't want to do that. However, there are times when it comes in handy—particularly if you're trying to reduce the amount of data you'll have to handle in Python.

Loops and Joins

The final anti-pattern is something I just saw at a client's office several days before writing this. The company has a large number of

products and wants to perform a query for each of the products. So, they did something like this:

```
>>> products = Product.objects.all()
>>> for product in products:
    ProductInfo.objects.filter(product_id=product.id).all()
```

This query took a very long time to run. Why? Because for each of the thousands of products, they were then issuing an additional SQL query. The funny thing was that each individual query executed quickly, so it didn't show up in our PostgreSQL slow-query logging monitor. But the effect of executing such a query was dramatic and ended up taking many minutes.

The solution was to turn our many queries into a single query. In SQL, we would use an inner join. And indeed, when I used an inner join in raw SQL, we found that instead of taking several minutes to execute, it took 1.5 seconds—obviously, a huge time savings.

There are two possible solutions in Django for this problem. The first is to use a raw SQL query, as I showed above. That's not an ideal solution, particularly since the whole idea of an

ORM is to remove the use of SQL and stay within a single language (Python, in this case). But there are times when you cannot avoid it.

However, if you want to be smarter about it, you can use Django's `selected_related` method. This allows you to retrieve not just one model, but a related model—in effect, creating a join in your database and producing one large query instead of many small ones. The effect on the performance of your application may well be dramatic in such a case, as I discovered when working with my client.

Conclusion

Object-relational mappers are wonderful things. However, at the end of the day, sometimes they can fool you into forgetting that there is a cost (in time and space) to bringing your data from

the database into your language. Most modern frameworks try to help by using lazy-loading and iterators, such that you retrieve individual records and not the entire data set. However, it's all too easy to retrieve everything at once, or make your application work too hard, or even to invoke too many queries on your database.■

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



Send comments or feedback via
[**http://www.linuxjournal.com/contact**](http://www.linuxjournal.com/contact)
or to [**ljeditor@linuxjournal.com**](mailto:ljeditor@linuxjournal.com).

Resources

The Django documentation is at <https://docs.djangoproject.com>. Look for the QuerySet documentation to see more about this subject.

If you're using Ruby on Rails, you should look at the documentation for ActiveRecord at <http://rubyonrails.com>. In particular, see the "lazy-loading" features that are now standard in ActiveRecord.

Finally, Pat Shaughnessy wrote a fantastic blog post on this subject, looking at Ruby on Rails and PostgreSQL. It's worth reading even if you don't use these specific technologies to understand the implications of bringing data out of the database. His piece is at <http://patshaughnessy.net/2015/6/18/dont-let-your-data-out-of-the-database>.



KYLE RANKIN

What's New in 3D Printing, Part III: the Software

Kyle looks at new developments in open-source 3D printing software.

This article is the third part of a four-part series that examines some of the changes in 3D printing that have occurred in the past three years since my first articles on the subject. Because this is *Linux Journal*, instead of discussing the entire 3D printing world, I'm focusing on the sections of the topic most relevant to open source and open hardware. In the first article, I gave a general overview on the current state of 3D printing. In the second, I covered what's changed in 3D printing hardware during the past three years, including the shift away from open hardware and which printers still hold onto their open hardware roots. In this article, I discuss the changes in 3D printing software, and then in the

final piece, I'll walk through setting up OctoPrint on a Raspberry Pi to control your printer remotely.

The software side of 3D printing three years ago was a model example of the power of open-source software. Just about any popular hobbyist printer you could choose used open-source software, all the way from the firmware (often Marlin) on the Arduino-based boards, to the software that sent Gcode to the printers (the Printrun suite of tools), to the slicers that took 3D models and converted them into the Gcode the printer understood (Slic3r and Repetier among others), to the software you could use to create the 3D models to begin with (OpenSCAD, FreeCAD and Blender,

among others). All of this software ran on Linux, so you could work with every part of the 3D printing life cycle without proprietary software. As interest in 3D printing grew, the open nature of all the software helped drive a lot of the innovation we see today.

Unfortunately, if not predictably, as 3D printing grew in popularity and new companies entered the market driven more by profit than by the health of the community, we saw the software side of 3D printing close up and become proprietary, just as we saw with the hardware side. In many cases, a company would enter the market with a proprietary 3D printer but still rely on open-source software to drive it to buy time to write up a proprietary alternative. Just like we saw Makerbot start to close up its hardware designs, we saw once open-source software like Repetier (previously under an Apache license) switch to be closed source. A number of companies that produce commercial CAD software, such as AutoCAD, also have entered the consumer market with proprietary host- and cloud-based CAD software, along with software to slice 3D models and control the 3D printers themselves.

All this talk of proprietary software might make the software side of 3D printing seem rather bleak, but among all those clouds is some silver lining.

While Printron and Slic3r still are running strong, Ultimaker, the creator of the open hardware Ultimaker line of 3D printers, has released its own slicing and printer control software called Cura that has become the popular choice among many in the 3D printing community. Cura combines the 3D printer control features of Printron (with a similar control panel, in fact) with a fast slicer and a sophisticated interface that makes it easy to view, rotate and scale 3D models before you print. Ultimaker even provides packages for a number of common Linux distributions so you don't have to bother building the software if you don't want to.

Although Cura was created for the Ultimaker printers, at startup, it launches a wizard that contains calibration settings for a number of popular 3D printers, including the full Printrbot line among others. In fact, Cura is now the recommended slicing and control software for Printrbot printers. Once the startup wizard completes, Cura knows the basic settings about your printer, such as the default size of the hot end, what size filament it takes and the overall size of the print bed. That doesn't mean you may not have to do some tweaking, however. In my case, I needed to modify the Gcode Cura used at the beginning and end of a print based on

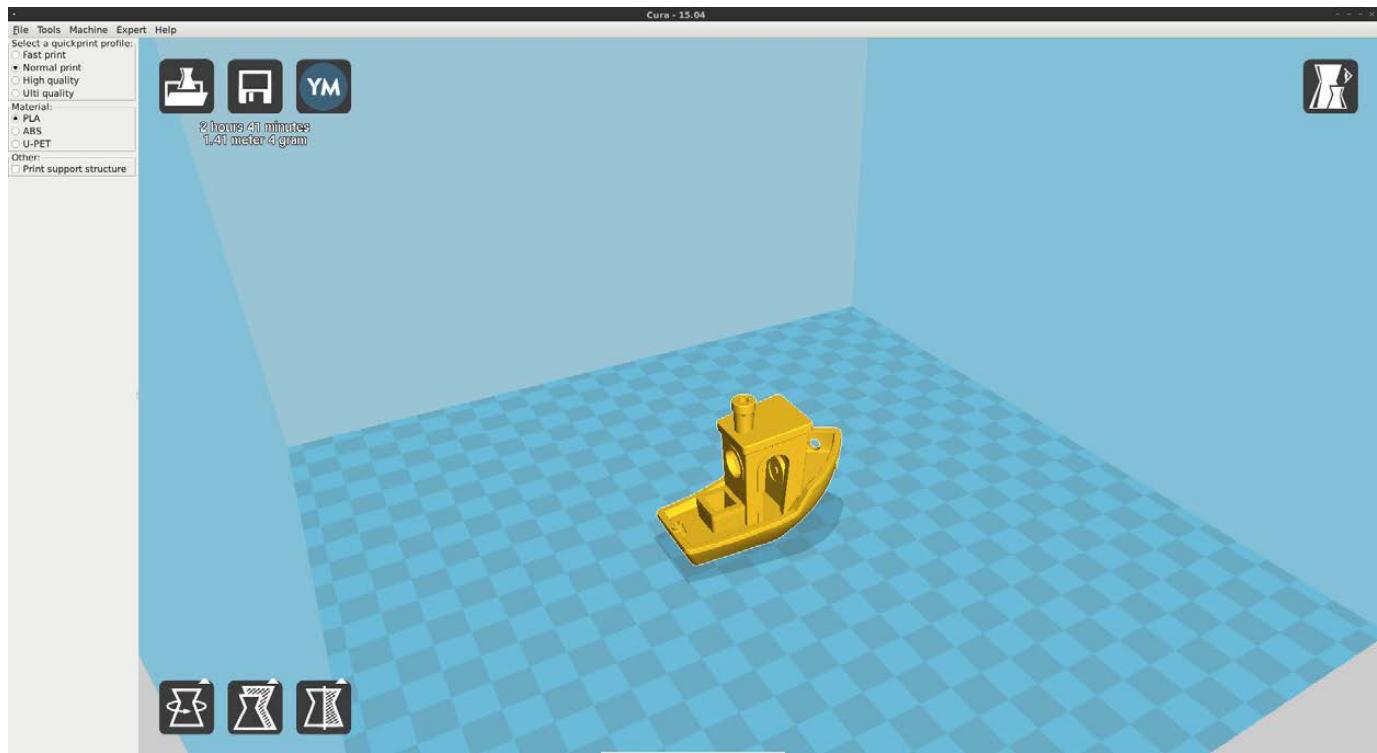


Figure 1. Default Cura Interface

Printrbot's recommendations to make sure that the auto-leveler functioned appropriately and that the extruder counter reset properly.

The default Cura interface is relatively straightforward (Figure 1), and if you click Expert→Switch to quickprint, it provides a simplistic view that hides a lot of the more advanced settings you may want to tweak. In fact, it even hides the more advanced printer control interface in favor of something simpler. When printing an object in this simplistic mode, you just select among a few different quality settings that set reasonable defaults for speed and layer height. The main window gives you a

clear view of any 3D objects you have loaded and allows you to use your mouse to rotate and move the option around as well as scale it up and down, much like with CAD software. One nice feature of Cura is that it not only slices very quickly, it also starts slicing the moment any setting changes instead of waiting for you to tell it to do it, so once you are ready to print the object, Cura likely already has generated the Gcode.

Those basic printing presets are geared toward Ultimaker printers, so if you use a different printer, you probably will want to switch over to the advanced settings view by clicking

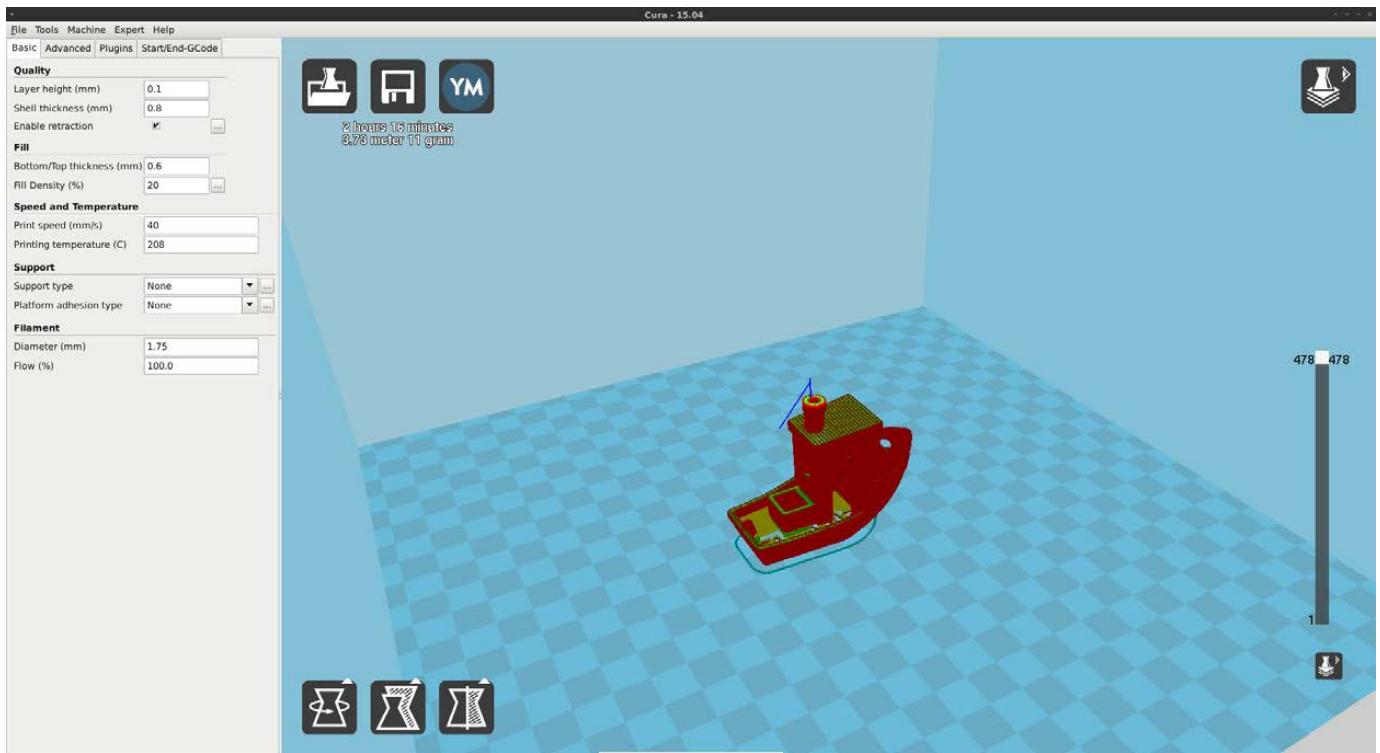


Figure 2. Advanced Cura Settings

Expert→Switch to full settings (Figure 2). In the advanced view, you can tweak just about any slicing setting you need once you settle on a configuration that works for your printer, and if that isn't enough, you can go into the expert settings window and tweak even more. When you get things exactly how you like them, you can save various profiles. This is particularly useful for some of the more exotic materials, such as Ninjaflex. I like to have a basic default profile for the normal PLA I use with average quality prints, and I've created custom profiles for Ninjaflex and for PLA with higher quality settings.

The CAD world from the open-source

perspective is one area that is still relatively similar to what it was three years ago. Tools like OpenSCAD and Blender still dominate, and OpenSCAD has seen variants like OpenJSCad that uses JavaScript as the language to build 3D objects. Most of the growth in CAD software appears to be on the free-in-cost-but-proprietary front from companies like AutoCAD that aim to provide a simple set of CAD features in a desktop or Web tool as an introduction to its more sophisticated suite of CAD software for sale down the line. Apart from the obvious downsides this presents to open-source advocates, it also means that although the .stl files

these tools generate can be opened and printed by everyone, the source project files (which make modifying a model much easier, like a GIMP project file instead of a .png) require that proprietary software to open.

Three years ago, the primary site you would use to share and download 3D printed objects was Thingiverse. Although Thingiverse is still a popular choice, there was some community backlash a few years ago due to some changes to its Terms of Use that gave it more rights over user-submitted prints than some in the community were comfortable with. This prompted a backlash, and protesters uploaded protest statement objects to the site. One result of this controversy was that a number of other sites sprang up to share 3D models. The most notable one for the Open Source community is probably YouMagine, which is run by Ultimaker and is a popular choice by those in the Open Source community for sharing models.

The final interesting advance in open-source 3D printing software is a big enough topic that my final article in this series will be devoted just to it: OctoPrint. With the OctoPrint project, you can turn a Raspberry Pi (or really any Linux computer) into a standalone Web-based controller for your 3D printer. OctoPrint supports Webcams

and the Raspberry Pi camera specifically, so you can watch the progress of your print from another room or another location entirely and take time-lapse videos of your prints. Even though I use Cura as my slicing software, and Cura easily could control my printer as well, I still use OctoPrint to manage my print jobs so I get the benefit of the video and the remote control. It's handy to start a print in my office and then be able to monitor it from the living room. If this sounds interesting to you, be sure to check next month's issue where I will describe how to set up and use OctoPrint in detail.■

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

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

Resources

Cura: <https://ultimaker.com/en/products/software>

OpenSCAD: <http://openscad.org>

OpenJSCad: <http://openjscad.org>

OctoPrint: <http://octoprint.org>

Learn what's new in SharePoint and Office 365!



SPTechCon The SharePoint Technology Conference

August 24-27, 2015
BOSTON

Over 70 classes
taught by expert speakers!

"This was a great conference that addresses all levels, roles and abilities. Great variety of classes, great presenters, and I learned many practical things that I can take back and start implementing next week."

—Kathy Mincey, Collaboration Specialist, FHI 360

SharePoint in the Cloud? On Premises? Or Both?

Come to SPTechCon Boston 2015 and learn about the differences between Office 365, cloud-hosted SharePoint, on-premises SharePoint, and hybrid solutions and build your company's SharePoint Roadmap!

Looking for SharePoint 2013 training?

Check out these targeted classes!

- Custom SharePoint 2013 Workflows that Use the SharePoint 2013 REST API
- SharePoint 2013 Farm Architecture and Visual Studio for Admin
- Creating a Branded Site in SharePoint 2013
- SharePoint's New Swiss Army Knife: The Content Search Web Part

Moving to Office 365?

Here are some targeted classes for YOU!

- Baby-Stepping Into the Cloud with Hybrid Workloads
- Demystifying Office 365 Administration
- Document Management and Records Management for Office 365
- Office 365 Search in the Cloud

MASTER THE PRESENT, PLAN FOR THE FUTURE! **REGISTER NOW!** → www.sptechcon.com



SHAWN POWERS

Slay Some Orcs!

Test your hardware in the best way ever—play some games!

As I write this here in the US, summer is raging. The heat, humidity and relentless sunshine makes it miserable to go outside. So for this article, I figured I'd take an indoor vacation from the weather and play some games. Gone are the days when people would pretend first-person shooters didn't interest them anyway. Although I love *Nethack* as much as the next nerd, sometimes I just want to shoot zombies. Let's look at a few of my current favorite games.

Steam

Before I talk about games, I have to mention Steam (<http://www.steampowered.com>). I haven't been following the development of SteamOS very closely, but the Linux client for Steam is simply amazing. You can filter the available games by Linux compatibility and purchase games like anyone else. Other game-buying options are available,

like Desura (<http://www.desura.com>), for example, but for the largest selection of commercial games, Steam is the platform of choice. Plus, if you have Windows and/or OS X, your games will work on multiple platforms. But enough about the underpinnings—today, I'm talking about games.

Zombies!

There's just something about zombies that I find appealing. Not that I want to hang out at a zombie bar or anything, but when it comes to first-person shooter games, zombies are my favorite shotgun fodder. I suspect it's because I don't really like shooting *people* in FPS games, but zombies are already dead. And creepy. And hungry for brains. Shooting them is a fairly easy call.

Because I'm not alone in my like of zombie mayhem, several such games are available. The ones I normally turn to are in the *Left 4 Dead* series (Figure 1).

Left 4 Dead (Original): This was



Figure 1. The *Left 4 Dead* franchise is full of awesome zombies and has an easy mode for wimps like me.

the first Steam game I played on Linux. I was thrilled (and admittedly surprised) how well it worked. The graphics are smooth, the controls are responsive, and the program is running on a native Linux operating system! *Left 4 Dead* offers individual or multiplayer modes, and either is fun. I'm not a terribly skilled gamer, so the on-line games tend to be more competitive than I'm up for. Thankfully, the single-player games with AI teammates are quite enjoyable—even for those of us who play on easy mode!

Left 4 Dead 2: The truth of the matter is, although I did purchase this game, I don't find it any more or less entertaining than the original. If you're more of a gamer than me, you'll probably appreciate the improvements in gameplay, graphics and so on. The premise, at least for me, is the same. Shoot the zombies. If you have to choose between *L4D* and *L4D2*, I recommend getting the second game, because the improvements are worth it. If you have *L4D* and you're just a casual

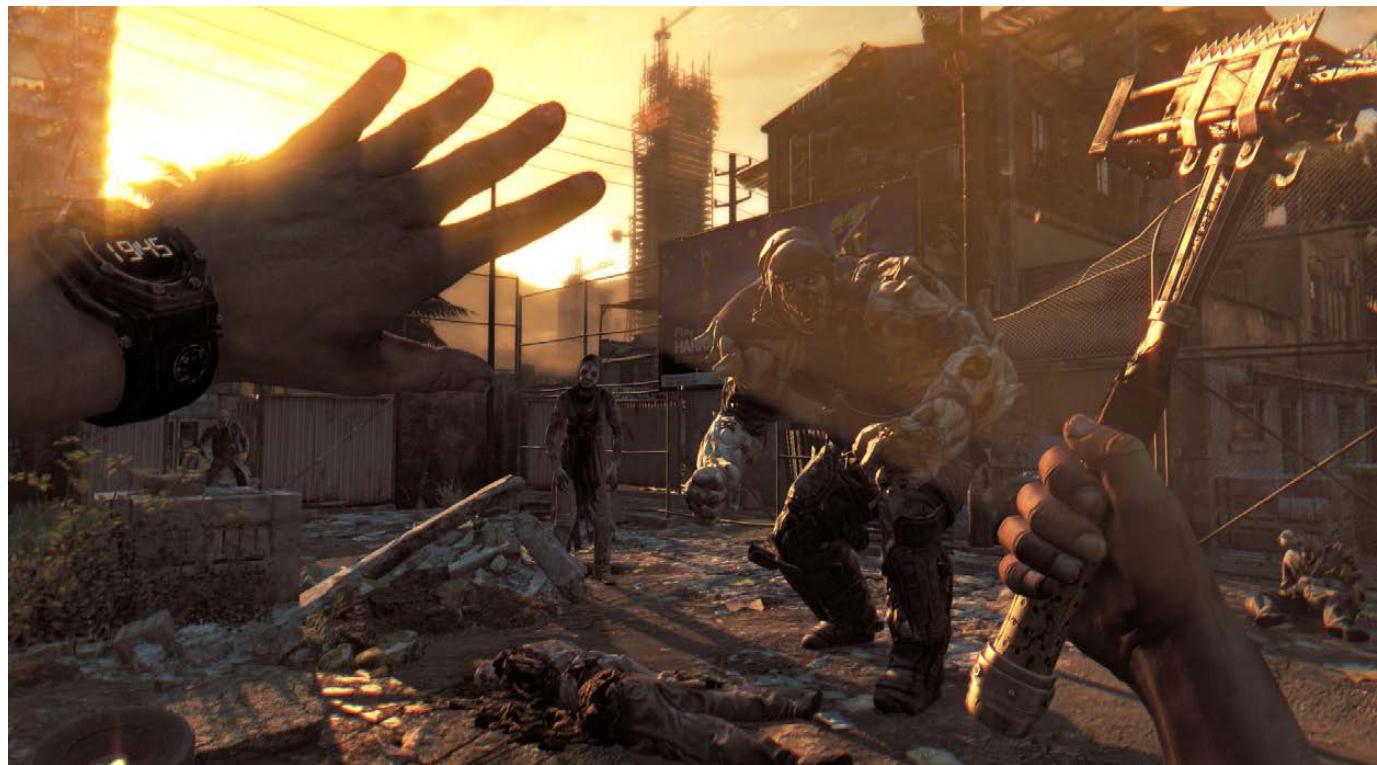


Figure 2. The *Dying Light* zombies are non-traditional, but still creepy enough to tickle my zombie bone.

zombie shooter, *L4D2* might be a disappointment. But still, zombies are awesome.

Dying Light (via Steam):

Dying Light is a zombie-esque game released this year that has a zombie feel, although the zombies are non-traditional—they're more monster-like (Figure 2). There is also an interesting day/night element to the game that actually starts to affect the player psychologically. I find my anxiety level increasing as the sun goes down.

Dying Light is one of the pricier options for the zombie hunter. At

the time of this writing, it weighs in at \$59.99 and is available via Steam. If you've played the *L4D* franchise and want more action along the same vein, *Dying Light* might be right up your alley. I recommend watching the trailers first though, because \$60 is a lot to fork over for a game you might not like.

Dead Island Series (via Steam):

I'm going to lump all the *Dead Island* games together, because conceptually, they're similar (although I admittedly haven't played *Dead Island Riptide*). At first it seems like just another zombie



Figure 3. Zombies in paradise? The tropical environment is a nice change from the standard urban apocalypse.

shooter with a tropical-themed background (Figure 3), however, *Dead Island* actually is a lot more fun than just shooting zombies. The world itself is very interactive (kick a beach ball, use it as a weapon), and with vehicle upgrades, side missions and the potential for awesome modifications, *Dead Island* is a series that puts a sunny disposition on a dire world. The disconnect between sunny beaches and undead zombies is refreshing—at least in a grab-your-shotgun-fun sort of way.

Strategy Games (Turn-Based and Real-Time)

Strategy games have been on Linux for a while, largely because they don't require awesome 3D graphics in order to be incredibly entertaining. Unfortunately, most of my personal favorite RTS (real-time strategy) games haven't been ported to Linux. I'm honestly not sure why, because I suspect like me, many folks would flock to download *Starcraft* if there were a native client. Yes, there are ways to play games like *Starcraft* via

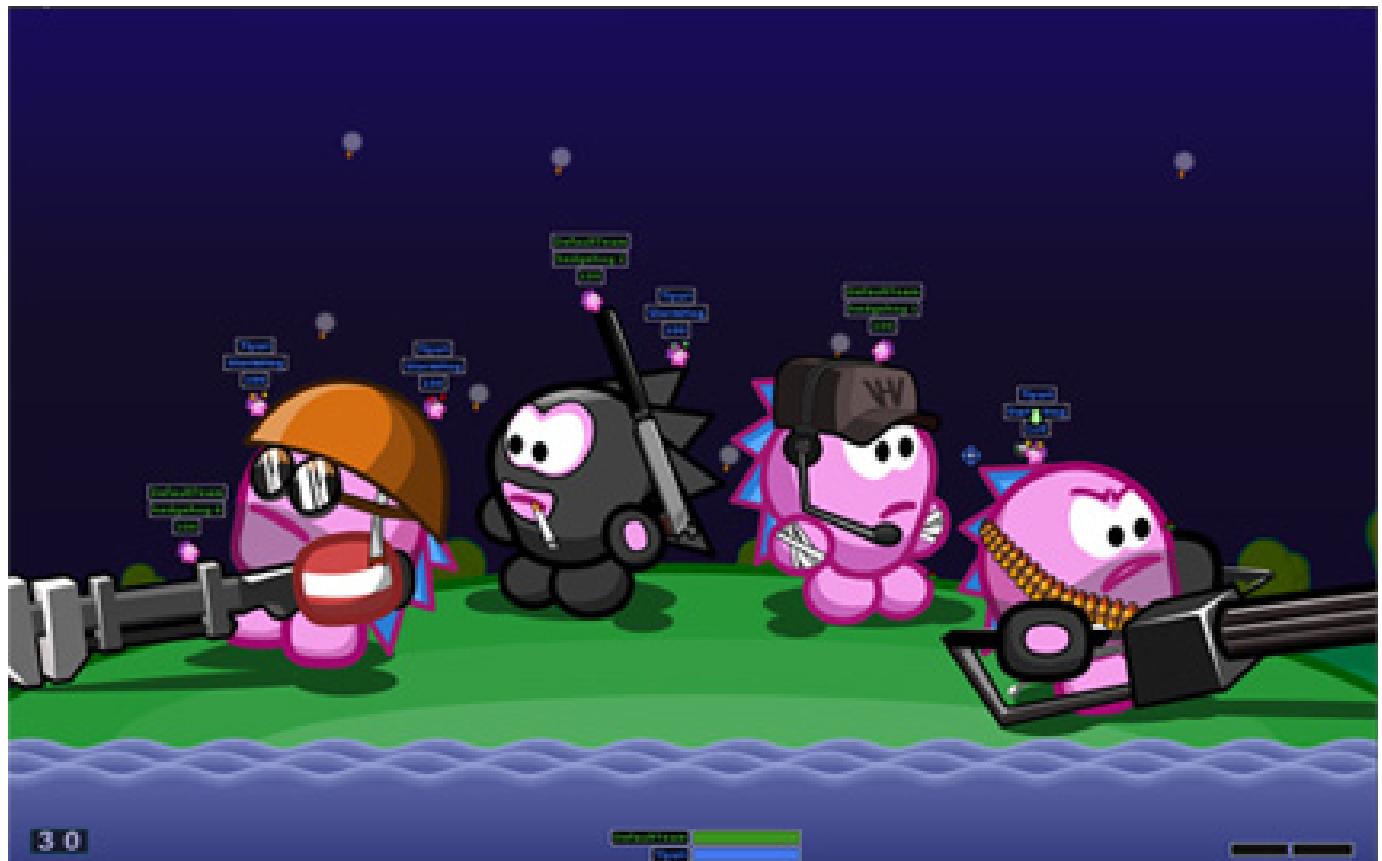


Figure 4. It's hard to hate pink hedgehogs—unless you're an opposing pink hedgehog.

Wine, but that's always a pain, and for me, it only *mostly* works. So for this article, I focus on native Linux games.

Battle for Wesnoth: This has been a staple Linux game for many years. It's a turn-based strategy game and has great gameplay along with cool graphics. If I had to come up with a downside, it would be that *Battle for Wesnoth* is too robust! There are 16 races in the fantasy world, and it supports single-player, multiplayer and on-line modes. The initial learning curve isn't bad, but I had a tough time moving from single-player mode into

multiplayer/on-line mode. I just got my rear end handed to me constantly. Still, the single-player mode is fun, and I really enjoy the game itself.

Hedgewars: I'm not sure if *Hedgewars* falls into the "strategy" realm, but it's turn-based and tons of fun. If you've ever played (or seen) *Worms*, *Hedgewars* will look very familiar. Instead of battling worms, however, the soldiers are pink hedgehogs (Figure 4), because why not? In the same style as *Worms*, *Hedgewars* is full of fun, cartoony graphics and a surprisingly robust



Figure 5. *0 A.D.* feels like it was filmed 2,000 years ago—which is odd, because 3D cameras weren't common back then.

set of weapons, costumes and mods. *Hedgewars* is fun with a capital F, and it makes me smile every time I play it.

0 A.D.: This game actually started its life as a modification for *Age of Empires II*, but it has since become its own game entirely. It's a classic real-time strategy game pitting one group against another, but it has a unique take on world building. Maps are inspired by real geography, and although largely fictional, the game has a very historical feeling to it (Figure 5). *0 A.D.* still is fairly early in

development and likely will get better as time goes on.

Civilization 5: Another in a long line of successful commercial strategy games, *Civ 5* is available via Steam. If you like the *Civilization* franchise, *Civ 5* will appeal to you. I personally never enjoyed playing *Civ*, so the latest iteration doesn't really float my boat, but it felt wrong not to include it in a list of strategy games.

Defense of the Ancients 2: The last game on my strategy list is a frustrating one. *Defense of the*



Figure 6. *Defense of the Ancients 2*—it's so awesome, but it's so stinkin' hard.

Ancients 2 (DotA 2) is an incredibly fun game available exclusively on Steam. The problem is it's really hard! The on-line community playing *DotA* is huge, and n00bs like me really struggle to learn the game well enough to enjoy it. If you're a gamer that is good at picking up skills quickly, *Defense of the Ancients 2* is a game you'll probably love.

First-Person Games

Generally, these are referred to as first-person shooters, but some have more than just shooting involved. Also, all the zombie games I mentioned at previously

are first-person games; I just thought zombies deserved their own category.

Borderlands Series: It's funny that games have become like movies and have sequels, prequels and so on. The *Borderlands* franchise is no different, with multiple games including a prequel. I personally like *Borderlands* because of the cross between RPG and FPS. The graphics also are unique (Figure 7). They're not quite cartoony, but they definitely feel less realistic, which oddly increases their appeal. The *Borderlands* environment makes me smile,



Figure 7. *Borderlands* has awesome graphics—the perfect blend of realistic and cartoony.

which is an interesting trait for a game where monsters try to kill you. Check it out if only to see the graphics. It's a commercial game series, available on Steam.

Metro: Last Light Redux: *Metro: Last Light Redux* improves graphics and gameplay to the fairly unique, not-quite-zombie-apocalypse game. Taking place in Russia, this game does include mutants, but it also contains a far more insidious enemy: other humans. Exploring the darker side of humanity in a post-nuclear world, *Metro: Last*

Light Redux is an interesting game with a unique gameplay. Check out the trailers on the Steam Web site, and see if this game (and series) is your cup of tea.

SuperTuxKart: If you're a *Mario Cart* fan or don't mind a little comic relief in your 3D racing games, *SuperTuxKart* is easy to love. It has lots of great tracks, multiple game modes, awesome characters, responsive controls and even a multiplayer mode. Much like *Mario Cart*, it's fun to play even if you're not a hard-core



Figure 8. Open-source mascots racing in a comical world full of awesome tracks? Yes please!

gamer (Figure 8). You can download this enjoyable open-source game from its SourceForge home page: <http://supertuxkart.sourceforge.net>.

Other Cool Games

Many other games are tons of fun, but don't fit directly into my categories. The easiest way to discover new games is to browse the Steam Web site, check out the

Desura offerings or watch for Humble Bundle game packs that almost always include Linux games. Here are a few I thought were important, but didn't fit into a specific category above.

Voxelands: First, a confession: I don't understand what people like about *Minecraft*. I really don't. I've tried, but I just can't seem to comprehend its draw. Nevertheless, if you like that sort of game, you should

give *Voxelands* a try. It's free and under rapid development with new maps, tools and so on.

Tales of Maj'Eyal: I guess this is technically a turn-based strategy, but to me, *Tales of Maj'Eyal* feels more like *Nethack* on steroids. The map is randomly generated, but the game doesn't suffer from lack of a planned map. In fact, it makes each game unique. Normally, *Nethack* annoys me with its randomness, but *Tales of Maj'Eyal* makes it feel like a plus as

opposed to a negative. It's important to note that *Tales of Maj'Eyal* is free to download, but if you want to support the company and keep your games in one place, you can buy it on the Steam store as well. It's \$7, which seems fair for a game that I enjoy playing. I recommend trying it out with the free download first and then buying it if you like it.

Anodyne: I'm not sure why this game lands on my short list. Don't get me wrong; it's fun and has a dreamy

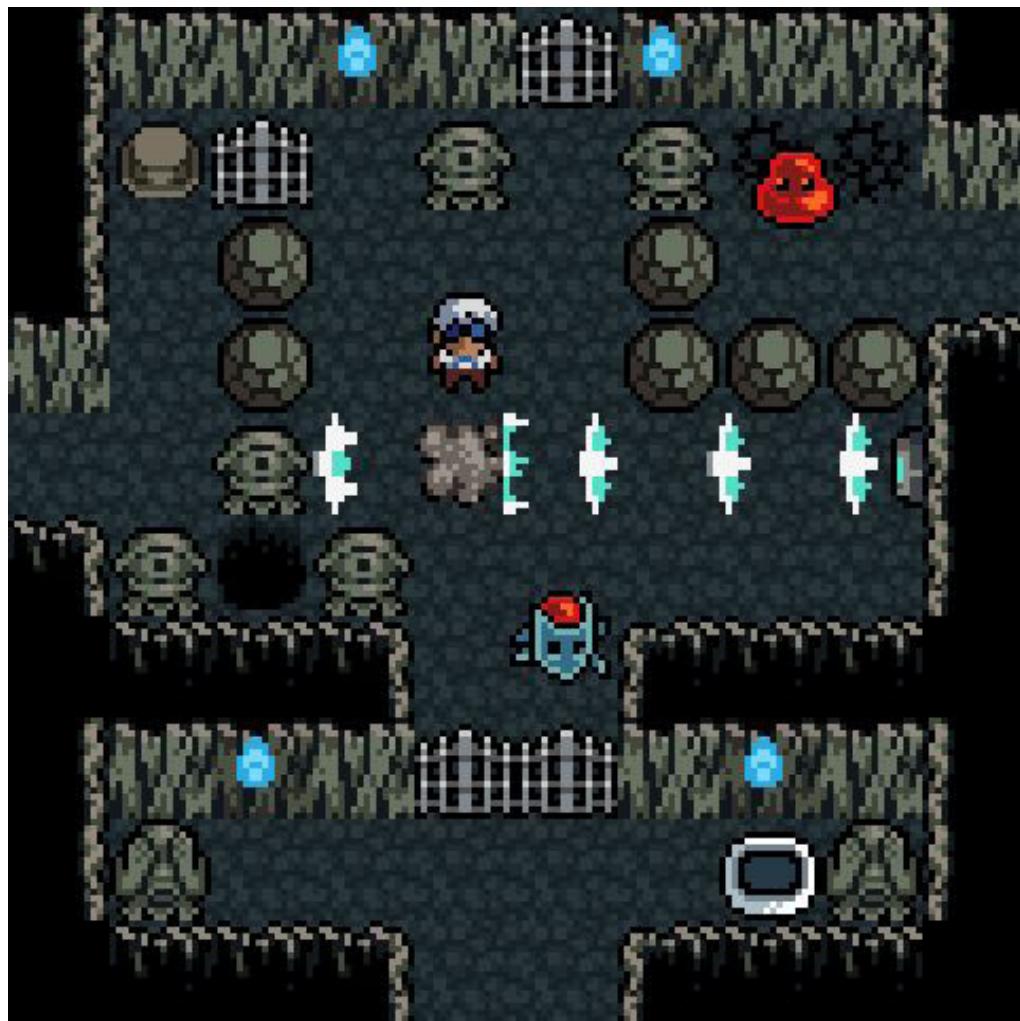


Figure 9.
Why yes, we are
going to party
like it's 1991!

soundtrack that reminds me of the 1990s Nintendo world, but it seems like more modern-themed games would be better. Perhaps it's the nostalgia that draws me in (Figure 9). If you ever played the original NES games like *Dragon Quest* or *Final Fantasy*, *Anodyne* likely will appeal to you. You can get it from <http://www.anodynegame.com> for \$9 and get a Steam key as well.

Game On!

Gaming isn't just for gamers, because if that were the case, I'd never get invited to the party. And although it's taken decades to get here, we're finally to the place where using Linux doesn't mean games are inaccessible or limited to crappy clones. If you like having fun or just need a break from the grind, give some of these games a try. I promise you'll find something worth wasting time! ■

Shawn Powers is the Associate Editor for *Linux Journal*.

He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.



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

Emulation:

Speaking of NES games from my youth, actually playing those games is a viable option as well. Acquiring the original ROM files is still legally and morally awkward, but if you're okay with downloading ROM files you may or may not have the right to own, emulation is a great way to play the actual games from your youth on a modern computer. The emulation programs all support controllers, and inventive gamers even can connect the original controllers to their computers for authentic gameplay. (Stay tuned for a future Open-Source Classroom column where I describe the build of my emulation station.)

Percona Live Europe

Data Performance Conference 2015

September 21-23
Amsterdam

3 days of tutorials & breakout
sessions by MySQL & NoSQL pros

www.percona.com/live



PERCONA
LIVE EUROPE
AMSTERDAM

StackIQ's Stacki



StackIQ, a provider of IT automation solutions for the data center and cloud, recently launched

Stacki, a fast and easy-to-use open-source provisioning tool for Linux servers. Short for “Stack Installer”, Stacki is designed to take any bare physical or virtual machine—with zero prerequisites—and turn it into a working Linux server in an organization’s network. Stacki provides “step zero” automation, allowing organizations to spin up Linux servers without the time, complexity and risk that can impede installations. The solution also addresses a critical “missing link” in today’s toolchain for IT operators—that is, automated RAID configuration and disk partitioning functionality. Stacki is a parallel, package-based installer for Red Hat Enterprise Linux and its derivatives, such as CentOS, Oracle Linux and Scientific Linux. In tandem with this new open-source initiative, StackIQ made available a one-day, on-site Stacki training and an implementation service for users who want to use the tools immediately for production servers.

<http://www.stackiq.com>

VARNISH PLUS⁺

Varnish Software's Varnish Plus

Until this newest release of Varnish Plus, Varnish Software’s complete, enterprise-level HTTP accelerator solution, the company’s customers had to rely on third-party products for SSL/TLS. Henceforth, however, Varnish Plus now offers full, integrated SSL/TLS support on both the HTTP back end and client side, allowing users to improve their Web site security and simplify their Web architectures, ultimately saving time and money. By enabling SSL/TLS in Varnish Plus, customers will enable encrypted and secure communication on both the front and back end and both HTTP server and client. On the client side, the HTTP server intercepts Web requests before they reach a Web server. The SSL/TLS support on this side enables traffic encryption between the client and Varnish. On the back end, the HTTP client fetches content missing in the cache from the Web server, enabling content to be fetched over the encrypted SSL/TLS, which particularly benefits customers who run a fully encrypted data center or have Web servers that reside in a different location from their Varnish Plus servers.

<http://www.varnish-software.com>

```

● ● ●
luke@clusterhq:~$ # Flocker can migrate a container and its associated volume from one physical
node to another and re-route network traffic.
luke@clusterhq:~$ flocker-deploy deployment.yml fig.yml
luke@clusterhq:~$ # The configuration consists of an application configuration which is a fig
file, plus a deployment configuration.
luke@clusterhq:~$ cat fig.yml
mongodb-volume-example:
  image: clusterhq/mongodb
  ports: ["27017:27017"]
  volumes: ["/data/db"]
luke@clusterhq:~$ # The application config references Docker images.
luke@clusterhq:~$ cat deployment.yml
version: 1
nodes:
  "172.16.255.250": []
  "172.16.255.251": ["mongodb-volume-example"]
luke@clusterhq:~$ # The deployment config defines which containers are running where - and lets
you dynamically move them around.
luke@clusterhq:~$ # We also support cross-host Docker links.
luke@clusterhq:~$ # Check out our docs to give it a go!■

```

ClusterHQ's Flocker

For good reason, forward-thinking developers and organizations are latching onto containers, which are emerging as one of IT's most important innovations since virtualization. Riding this wave is the ClusterHQ and its Flocker 1.0 open-source container data management software application.

ClusterHQ explains that Flocker, by enabling

stateful Docker containers to be moved between servers easily, facilitates widespread production deployment of containers for databases, queues and key value stores. As modern applications are being built from both stateless and stateful microservices, Flocker enables and simplifies the process of containerizing entire applications, including their state, to take full advantage of the portability and massive per-server density benefits inherent in containers. This operational freedom allows DevOps organizations to increase the value of their Docker investment and opens the door for containers to be used in a greater variety of mainstream enterprise use cases in production. The Apache-licensed Flocker 1.0 is available for download at ClusterHQ's Web site.

<http://clusterhq.com/flocker>

Red Hat Software Collections

Red Hat says that the more frequent release schedule of its

Red Hat Software Collections gives developers a deeper selection of the latest tools without sacrificing Red Hat's enterprise-grade support for building enterprise applications, including those built for Linux container deployments. The Collections, recently upgraded to v2.0, is a package of essential Web development tools, dynamic languages, open-source databases, C and C++ compilers, the Eclipse IDE, and a variety of development and performance management tools. These updated components can be installed alongside versions included in base Red Hat Enterprise Linux. Version 2.0 includes more than ten new collections, such as Python 3.4 (while supporting Python 2.7 and 3.3), PHP 5.6 and Passenger 4.0, not to mention several other new updates to the already included collections per Red Hat's curation. In addition, Dockerfiles for many of the most popular collections, such as Perl, PHP and Python, also are available to help developers dive head first into the world of Linux containers.

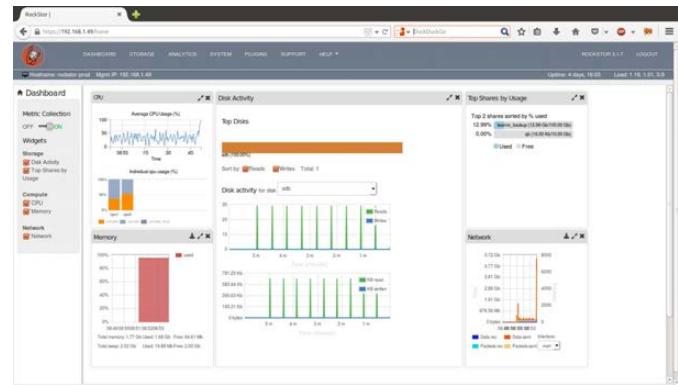
<http://www.redhat.com>



Rockstor, Inc.'s Rockstor NAS

The latest dispatch from Rockstor, Inc., involves a major upgrade to the company's RockStor NAS (Network Attached Storage) solution. The company describes the new 3.8-0 release of the free and open-source Rockstor as an "exciting update with a lot of improvements and new features", the most salient of which is a tightening up the Rock-on app framework. New Rock-ons include Syncthing, Transmission and BTSync. Other improvements include added functional tests to improve coverage by about 10% and the correction of a Samba password reset issue.

<http://www.rockstor.com>



Jonathan Lazar, Daniel Goldstein and Anne Taylor's *Ensuring Digital Accessibility through Process and Policy* (Morgan Kaufmann)

Our Linux Community, which has slayed many a dragon-like challenge, is home to some of the most impressive IT talent on Earth. One frontier to which this talent could be applied wisely is the expansion of digital accessibility to those with disabilities. A guide to navigating this frontier is the new Morgan Kaufmann book *Ensuring Digital Accessibility through Process and Policy* by Jonathan Lazar, Daniel Goldstein and Anne Taylor. As the title implies, this book provides readers with must-know information on digital accessibility from both technical and policy perspectives. Many organizations may not realize that inaccessible digital interfaces and content may indeed present forms of societal discrimination that can be illegal under various laws. In the book, the authors—with legal, technical and research expertise—explore the history of accessible computing, why digital accessibility is socially and legally important, the technical details (interface standards and evaluation methods) and legal details (laws, lawsuits and regulations). Myriad case studies illustrate best practices for guaranteeing access to the world of digital information for all users.

<http://morgankaufmann.com>



Matt Sainsbury's Game Art (No Starch Press)

In the world of video games, a lone independent developer on a tiny budget can create an experience every bit as compelling as a blockbuster built by a team of hundreds. But like all works of art, every game begins with a spark of

inspiration and a passion to create. Those interested in the creative process behind video games will find great delight in art critic Matt Sainsbury's new book *Game Art*. Subtitled *Art from 40 Video Games and Interviews with Their Creators*, *Game Art* is a collection of breathtaking concept art and behind-the-scenes interviews from influential video game studios, both major and independent. Readers can immerse themselves in fantastic artwork and explore the creative thinking behind more than 40 console, mobile and PC games.

<http://www.nostarch.com>

CIARA's ORION HF320D Server

The long and the short of the new ORION HF320D server from CIARA is this: it is reputedly "the first and world's fastest overclocked 2U 2-Node High Frequency Trading (HFT) server". Boasting two independent Intel processors with performance up to 4.5GHz on eight cores in a 2U rackmount chassis, CIARA says that the ORION HF320D server "officially raises the bar on speed and density for high frequency trading". The ORION HF320D is designed to handle today's high frequency trading algorithms and data market analysis, both of which require the fastest speed and lowest latency on the largest number of cores. CIARA's formula lies in the power combination of the latest generation of Intel processors (Haswell E) with the CIARA proprietary half-width motherboard design, which together unlock a vast array of new opportunities for performance and low latency. Elements that round out the ORION HF320D's package include NVMe SSD, LSI 3108 hardware RAID, Processor Cache acceleration and the latest technology in liquid cooling from Asetek.

<http://www.ciaratech.com/orionhf>



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.

An Introduction to Tabled Logic Programming with Picat

Picat is a great starting point for a journey into the realm of logic-based programming languages. It provides many of the goodies Prolog has, and at the same time, it allows you to fall back on convenient imperative concepts like destructive assignments and loops.

SERGII DYMCHENKO

Picat is a new logic-based programming language. In many ways, Picat is similar to Prolog, especially B-Prolog, but it has functions in addition to predicates, pattern-matching instead of unification in predicate heads, list comprehensions and optional destructive assignment. Knowing some Prolog helps when learning Picat but is by no means required.

According to the authors of the language, Picat is an acronym for:

- Pattern-matching.
- Imperative.
- Constraints.
- Actors.
- Tabling.

Picat has a lot of interesting features, such as constraint logic programming support and interfaces to various solvers. In this article, I focus on one aspect of Picat: tabling and a tabling-based planner module.

First, let's install and learn some basics of Picat. Installing Picat is easy; you can download precompiled binaries for 32- and 64-bit Linux systems, as well as binaries for other

platforms. If you want to compile it yourself, C source code is available under the Mozilla Public License. The examples here use Picat version 1.2, but newer or slightly older versions also should work.

After the installation, you can run `picat` from a command line and see Picat's prompt:

```
Picat 1.2, (C) picat-lang.org, 2013-2015.  
Picat>
```

You can run commands (queries) interactively with this interface.

Let's start with the mandatory "Hello, World":

```
Picat> println("Hello, World!").  
Hello, World!  
yes
```

No real surprises here. The `yes` at the end means that Picat successfully executed the query.

For the next example, let's assign 2 to a variable:

```
Picat> X = 2.  
X = 2  
yes
```

Note the uppercase letter for the variable name; all variable names must start with a capital letter or an

underscore (the same as in Prolog).

Next, assign symbols to the X variable (symbols are enclosed in single quotes; for many symbols, quotes are optional, and double-quoted strings, like the “Hello, World!” above, are lists of symbols):

```
Picat> X = a.  
X = a  
yes  
Picat> X = 'a'.  
X = a  
yes
```

For capital-letter symbols, single quotes are mandatory (otherwise it will be treated as a variable name):

```
Picat> X = A.  
A = X  
yes  
Picat> X = 'A'.  
X = 'A'  
yes
```

Note that the variable X in different queries (separated by a full stop) are completely independent different variables.

Lists

Next, let's work with lists:

```
Picat> X = [1, 2, 3, a].
```

```
X = [1, 2, 3, a]  
yes
```

Lists are heterogeneous in Picat, so you can have numbers as the first three list elements and a symbol as the last element.

You can calculate the results of arithmetic expressions like this:

```
Picat> X = 2 + 2.  
X = 4  
yes  
Picat> X = [1, a, 2 + 2].  
X = [1, a, 4]  
yes
```

```
Picat> X = 2, X = X + 1.  
no
```

This probably is pretty surprising for you if your background is in mainstream imperative languages. But from the logic point of view, it makes perfect sense: X can't be equal to X + 1.

Using := instead of = produces a more expected answer:

```
Picat> X = 2, X := X + 1.  
X = 3  
yes
```

The destructive assignment operator := allows you to override Picat's usual

single-assignment “write once” policy for variables. It works in a way you’d expect from an imperative language.

You can use the [|] notation to get a “head” (the first element) and a “tail” (the rest of the elements) of a list:

```
Picat> X = [1, 2, 3], [Tail | Head]
= X.
X = [1,2,3]
Tail = 1
Head = [2,3]
yes
```

You can use the same notation to add an element to the beginning of a list:

```
Picat> X = [1, 2, 3], Y = [0 | X].
X = [1,2,3]
Y = [0,1,2,3]
yes
Picat> X = [1, 2, 3], X := [0 | X].
X = [0,1,2,3]
yes
```

The first example creates a new variable Y, and the second example reuses X with the assignment operator.

TPK Algorithm

Although it’s handy to be able to run small queries interactively to try

different things, for larger programs, you probably will want to write the code to a file and run it as a script.

To learn some of Picat’s syntactical features, let’s create a program (script) for a TPK algorithm. TPK is an algorithm proposed by D. Knuth and L. Pardo to show the basic syntax of a programming language beyond the “Hello, World!” program. The algorithm asks a user to enter 11 real numbers ($a_0 \dots a_{10}$). After that, for $i = 10 \dots 0$ (in that order), the algorithm computes the value of an arithmetic function $y = f(a_i)$, and outputs a pair (i, y) , if $y \leq 400$ or $(i, \text{TOO LARGE})$ otherwise.

Listing 1. TPK

```
f(T) = sqrt(abs(T)) + 5 * T**3.

main =>

    N = 11,
    As = to_array([read_real() : I in 1..N]),
    foreach (I in N..-1..1)
        Y = f(As[I]),
        if Y > 400 then
            printf("%w TOO LARGE\n", I)
        else
            printf("%w %w\n", I, Y)
        end
    end.
```

First, the code defines a function to

calculate the value of `f` (a function in Picat is a special kind of a predicate that always succeeds with a return value). The `main` predicate follows (`main` is a default name for the predicate that will be run during script execution). The code uses list comprehension (similar to what you have in Python, for example) to read the 11 space-separated real numbers into an array `As`. The `foreach` loop iterates over the numbers in the array; `I` goes from 11 to 1 with the step -1 (in Picat, array indices are 1-based). The loop body calculates the value of `y` for every iteration and prints the result using an “if-then-else” construct. `printf` is similar to the corresponding C language function; `%w` can be seen as a “wild card” control sequence to output values of different types.

You can save this program to a file with the `.pi` extension (let’s call it `tpk.pi`), and then run it using the command `picat tpk.pi`. Input 11 space-separated numbers and press Enter.

Tabling

Now that you have some familiarity with the Picat syntax and how to run the scripts, let’s proceed directly to tabling. Tabling is a form of automatic caching or memoization—results of

previous computations can be stored to avoid unnecessary recomputation.

You can see the benefits of tabling by comparing two versions of a program that calculates Fibonacci numbers with and without tabling.

Listing 2 shows a naive recursive Fibonacci implementation in Picat.

Listing 2. Naive Fibonacci

```
fib(0) = 1.  
fib(1) = 1.  
fib(N) = F =>  
    N > 1,  
    F = fib(N - 1) + fib(N - 2).  
  
main =>  
    N = read_int(),  
    println(fib(N)).
```

This naive implementation works, but it has an exponential running time. Computing the 37th Fibonacci number takes more than two seconds on my machine:

```
$ time echo 37 | picat fib_naive.pi  
39088169  
real 0m2.604s
```

Computing the 100th Fibonacci number would take this program forever!

But, you can add just one line (`table`) at the beginning of the program to see a dramatic improvement in running time.

Now you can get not only 37th Fibonacci number instantly, but even the 1,337th (and the answer will not suffer from overflow, because Picat supports arbitrary-length integers).

Effectively, with tabling, you can change the asymptotic running time from exponential to linear.

An even more useful feature is “mode-directed” tabling. Using it you can instruct Picat to store the minimal or the maximal of all possible answers for a non-deterministic goal. This feature is very handy when implementing dynamic programming algorithms. However, that topic is beyond the scope of this article; see Picat’s official documentation to learn more about mode-directed tabling.

The planner Module

Picat also has a tabling-based planner module, which can be used to solve artificial intelligence planning problems. This module provides a higher level of abstraction and declarativity.

To use the module, an application programmer has to specify action

and final predicates.

The final predicate, in its simplest form, has only one parameter—the current state—and succeeds if the state is final.

The action predicate usually has several clauses—one for each possible action or group of related actions. This predicate has four parameters: current state, new state, action name and action cost.

Let’s build a maze-solver using the planner module. The maze-solving program will read a maze map from the standard input and output the best sequence of steps to get to the exit. Here is an example map:

```
5 5
@.#
=.#
.##.
.#X.
.|...
```

The first line contains the dimensions of the maze: the number of rows R and columns C.

Next, R lines describe the rows of the maze. Here is the description of the map symbols:

- @ — initial hero position.
- . — an empty cell.

■ # — a permanent wall.

■ = — a key.

■ | — a closed door.

■ X — the exit.

The hero can move up, down, left and right (no diagonals) to any open cell (a cell without a wall or a closed door). The hero can pick up keys and open doors. Opening a door and moving into a newly open cell is considered one action. To open a door, the hero must have a key.

Because this is a magic maze, the key disappears after it opens a door. All keys are identical, so opening a door basically just decreases the number of keys the hero has by one.

The goal is to reach the exit using the minimum amount of energy. Moving to an open cell costs one energy unit, picking up a key costs one energy unit, and opening a door and moving to the cell previously occupied by that door costs two energy units.

Let's represent a state for this problem as a tuple (R, C, (ExitI, ExitJ), Walls, Doors, Keys, K, (HeroI, HeroJ)):

■ R and C are the number of rows and columns in the maze.

■ (ExitI, ExitJ) are the coordinates of the exit.

■ Walls is a list of the positions of all walls.

■ Doors is a list of the positions of all closed doors.

■ Keys is a list of the positions of not-yet-picked-up keys.

■ K is the number of keys the hero has.

■ (HeroI, HeroJ) are coordinates of the hero's position.

Let's first do some boring work of translating a textual representation of a maze to an initial state in the format defined before.

The main predicate is an imperative procedure in constructing an initial state from a textual representation of a maze: you read the input line by line, symbol by symbol, and then construct the lists of walls, doors and keys, as well as record the coordinates of the hero and the exit.

Let's save this program to maze_read.pi, the maze description

from above to maze.txt, and run the program (the output is split into several lines for clarity):

```
$ picat maze_read.pi < maze.txt
5,5,
(4,3),
[(4,2),(3,3),(3,2),(2,3),(1,3)],
[(5,2)],
[(2,1)],
0,1,1
```

So, you have the dimensions of the maze (5 by 5), the coordinates of the exit (4, 3), the list of the coordinates of all five walls, a one-element list of the closed doors and a one-element list of the keys available for picking up. The hero has 0 keys and starts in cell (1, 1).

Now that you have your state, you can define some predicates to solve the problem. First, the `final` predicate for the planner module:

```
final((_, _, (I, J), _, _, _, _, (I, J))) =>
    true.
```

The state is final when the hero is in the cell with the same coordinates as the exit cell. Variables with name `_` are throw-away, “don’t care” variables that are not required to have any specific value (Picat invents a different name for each `_` behind the scenes, so

Listing 3. Read the Maze Description

```
main =>
    R = read_int(), C = read_int(),
    Walls = [], Doors = [], Keys = [],
    (ExitI, ExitJ) = (_, _),
    (HeroI, HeroJ) = (_, _),
    foreach (I in 1..R)
        Line = read_line(),
        foreach (J in 1..C)
            Char = Line[J],
            if Char == '@' then
                HeroI := I, HeroJ := J
            end,
            if Char == 'X' then
                ExitI := I, ExitJ := J
            end,
            if Char == '#' then
                Walls := [(I, J) | Walls]
            end,
            if Char == '|' then
                Doors := [(I, J) | Doors]
            end,
            if Char == '=' then
                Keys := [(I, J) | Keys]
            end
        end
    end,
    InitState = (R, C, (ExitI, ExitJ),
                 Walls, Doors, Keys,
                 0, (HeroI, HeroJ)),
    println(InitState).
```

they don’t have to be equal either).

Next, describe the action to take a key if the hero is in a cell with one:

```
action(State, NewState, Action, Cost) ?=>
    (R, C, (ExitI, ExitJ), Walls, Doors, Keys,
     K, (HeroI, HeroJ)) = State,
```

```
select((HeroI, HeroJ), Keys, NewKeys),
Action = $take_key(HeroI, HeroJ),
Cost = 1,
NewState = (R, C, (ExitI, ExitJ),
Walls, Doors, NewKeys,
K + 1, (HeroI, HeroJ)).
```

First you decompose the state into components, and then you try to select a key with the current coordinates of the hero from the Keys list. If there is such a key, this will succeed, and the rest of the keys will be assigned to “NewKeys”; otherwise, `select` fails, and Picat will break the execution of this action clause.

The name of the action is `take_key`, with the coordinates of the event in the parentheses (the `$` instructs Picat to treat it literally, like a string, and not to try to execute as a function), and the cost is one energy unit.

The new state is almost the same as the old state, except that the number of keys the hero has is increased by one, and the current key no longer is available to pick up.

Besides picking up keys, there are two more possible actions: moving to an empty cell and moving to a cell with a door after opening it. It’s a good idea to combine both these actions into one clause, because they share a lot of code used to select a new hero position and check whether it’s within the maze boundary:

```
action(State, NewState, Action, Cost) =>
(R, C, (ExitI, ExitJ), Walls, Doors, Keys,
K, (HeroI, HeroJ)) = State,
(
Di = 0, Dj = 1
;
Di = 0, Dj = -1
;
Di = 1, Dj = 0
;
Di = -1, Dj = 0
),
NewHeroI = HeroI + Di,
NewHeroJ = HeroJ + Dj,
NewHeroI >= 1, NewHeroI <= R,
NewHeroJ >= 1, NewHeroJ <= C,
(
% move to open cell
not membchk((NewHeroI, NewHeroJ), Walls),
not membchk((NewHeroI, NewHeroJ), Doors),
Action = $move(NewHeroI, NewHeroJ),
Cost = 1,
NewState = (R, C, (ExitI, ExitJ),
Walls, Doors, Keys,
K, (NewHeroI, NewHeroJ))
;
% open a door and move to that cell
K > 0,
select((NewHeroI, NewHeroJ), Doors, NewDoors),
Action = $open(NewHeroI, NewHeroJ),
Cost = 2,
NewState = (R, C, (ExitI, ExitJ),
Walls, NewDoors, Keys,
K - 1, (NewHeroI, NewHeroJ))
).
```

Again, first you decompose the state into the components. Next, you try all possible new positions for the hero with non-deterministic disjunction: ;.

A position must be within the maze boundaries: I must be from 1 to R, and J must be from 1 to C. After that, there are two possibilities: move to an open cell, or open a door and move to that cell.

Moving to an open cell is possible only if there isn't a wall or a closed door at the desired position. Two not membchk lines verify this condition. If the condition is met, the action name is move, and the cost is one energy unit. The only change in the state is the hero's position.

Opening an door is possible if there is a door at the position and the hero has at least one key. The select line here is similar to the line for the take action, but now you select a door instead of a key. If the conditions are met, the action name is open, and the cost is two energy units. The new state is almost the same as the old state, but the door is removed from the list of doors, the number of keys the hero has is decreased by one, and the hero has moved to a new position.

To use the defined final and action predicates and find the plan, you need

to change `println(InitState)` to `best_plan_unbounded(InitState, Plan)`, `println(Plan)` in the main from the `maze_read.pi` program. (Note: `best_plan_unbounded` is one of the predicates of the planner module for finding best plans. This particular version uses memory to avoid re-exploring states, converting tree search in the space of all possible plans to graph search.)

Listing 4 shows the complete maze program.

After running it for the maze used above, you get an optimal plan (list of actions) to solve the maze (the output is split into several lines for clarity):

```
$ picat maze.pi < maze.txt
[
  move(2,1),
  take_key(2,1),
  move(3,1),
  move(4,1),
  move(5,1),
  open(5,2),
  move(5,3),move(4,3)
]
```

You can try to run this program with inputs of various sizes and with different features. For example, this input requires the hero to take a key to the right, then go left to get more keys,

Listing 4. Full Maze Program

```

import planner.

action(State, NewState, Action, Cost) ?=>
    (R, C, (ExitI, ExitJ), Walls, Doors, Keys,
     K, (HeroI, HeroJ)) = State,
    select((HeroI, HeroJ), Keys, NewKeys),
    Action = $take_key(HeroI, HeroJ),
    Cost = 1,
    NewState = (R, C, (ExitI, ExitJ),
                Walls, Doors, NewKeys,
                K + 1, (HeroI, HeroJ)).

action(State, NewState, Action, Cost) =>
    (R, C, (ExitI, ExitJ), Walls, Doors, Keys,
     K, (HeroI, HeroJ)) = State,
    (
        Di = 0, Dj = 1
    ;
        Di = 0, Dj = -1
    ;
        Di = 1, Dj = 0
    ;
        Di = -1, Dj = 0
    ),
    NewHeroI = HeroI + Di,
    NewHeroJ = HeroJ + Dj,
    NewHeroI >= 1, NewHeroI <= R,
    NewHeroJ >= 1, NewHeroJ <= C,
    (
        % move to open cell
        not membchk((NewHeroI, NewHeroJ), Walls),
        not membchk((NewHeroI, NewHeroJ), Doors),
        Action = $move(NewHeroI, NewHeroJ),
        Cost = 1,
        NewState = (R, C, (ExitI, ExitJ),
                    Walls, Doors, Keys,
                    K, (NewHeroI, NewHeroJ))
    ;
        % open a door and move to that cell
        K > 0,
        select((NewHeroI, NewHeroJ), Doors, NewDoors),
        Action = $open(NewHeroI, NewHeroJ),
        Cost = 2,
        NewState = (R, C, (ExitI, ExitJ),
                    Walls, NewDoors, Keys,
                    K - 1, (NewHeroI, NewHeroJ))
    ).

final((_, _, (I, J), _, _, _, _, (I, J))) =>
    true.

main =>
    R = read_int(), C = read_int(),
    Walls = [], Doors = [], Keys = [],
    (ExitI, ExitJ) = (_, _),
    (HeroI, HeroJ) = (_, _),
    foreach (I in 1..R)
        Line = read_line(),
        foreach (J in 1..C)
            Char = Line[J],
            if Char == '@' then
                HeroI := I, HeroJ := J
            end,
            if Char == 'X' then
                ExitI := I, ExitJ := J
            end,
            if Char == '#' then
                Walls := [(I, J) | Walls]
            end,
            if Char == '|' then
                Doors := [(I, J) | Doors]
            end,
            if Char == '=' then
                Keys := [(I, J) | Keys]
            end
    end,
    InitState = (R, C, (ExitI, ExitJ),
                 Walls, Doors, Keys,
                 0, (HeroI, HeroJ)),
    best_plan_unbounded(InitState, Plan),
    println(Plan).

```

and then go right again to the exit:

```
1 10 ==|=|=||X
```

Of course, you can improve the maze program in many different ways:

- Better user interface: currently, the output is not very easy to read, and the program exits with an error if the maze is not solvable.
- Sets or hash tables instead of lists: looking for a key or wall in a list requires linear time, while with a more appropriate data structure, it will be constant.
- Adding a heuristic: the search could be improved with a heuristic to make it a variant of an IDA* algorithm.
- New maze features: you could implement different kinds of keys, weapons, treasure and monsters.

Overall, Picat looks like a really good starting point for a journey into the realm of logic-based programming languages. It provides many of the goodies Prolog has, such as non-determinism and built-in depths-first search with backtracking, and at the same time, Picat allows you to fall back to convenient imperative concepts like destructive assignments and loops. Higher-level features, like tabling and the planner module, provide ways to write concise, declarative and efficient programs. ■

Sergii Dymchenko is a software developer, a programming languages enthusiast and a great believer in the importance of open-source and free software. You can reach him at <http://sdymchenko.com> and @kit1980 (Twitter).



Send comments or feedback via
[**http://www.linuxjournal.com/contact**](http://www.linuxjournal.com/contact)
or to [**ljeditor@linuxjournal.com**](mailto:ljeditor@linuxjournal.com).

Resources

Picat: <http://picat-lang.org>

“Artificial intelligence planning with Picat” by Sergii Dymchenko:
<http://sdymchenko.com/blog/2015/01/31/ai-planning-picat>

Hakan Kjellerstrand’s Picat Page: <http://www.hakank.org/picat>

“Declaratively solving Google Code Jam problems with Picat” by Sergii Dymchenko and Mariia Mykhailova: <http://arxiv.org/abs/1504.00977>

“Combinatorial Search with Picat” by Neng-Fa Zhou: <http://arxiv.org/abs/1405.2538>

HASH TABLES

THEORY AND PRACTICE

Why are **hash tables** so important?

MIHALIS TSOUKALOS

The first time I heard about hash tables was after taking a compilers course during my BSc. The truth is, I was not able to understand and appreciate their usefulness fully back then. Now that I know more about hash tables, I decided to write about them so others will see their importance as well.

Hash tables can be implemented in any programming language, including Awk. However, the choice of programming language is not the most important thing compared to other critical choices. Hash tables are used in compilers, databases, caching, associative arrays and so on. Hash tables are one of the most important data structures in computer science.

The Problem

The problem that will serve as an example for this article is finding out how many words from one text file appear in another text file. All programs in this article use a text file (*Pride and Prejudice*) for populating the hash table. Another text file (*The Adventures of Tom Sawyer*) will be used for testing the performance of the hash table. You can download both text files from Project Gutenberg.

The following output shows how many words each file contains:

```
$ wc AoFTS.txt  
9206 73845 421884 AoFTS.txt  
$ wc PandP.txt  
13426 124589 717573 PandP.txt
```

As you can see, both text files are relatively large, which is good for benchmarking. Your real-life hash tables may not be as big. In order to remove various control characters, as well as punctuation marks and numbers, both text files were processed further:

```
$ strings PandP.txt > temp.INPUT  
  
$ awk '{for (i = 1; i <= NF; i++) print $i}' temp.INPUT > new.INPUT  
  
$ cat new.INPUT | tr -cd '![a-zA-Z]\n' > INPUT  
  
$ strings AoFTS.txt > temp.CHECK  
  
$ awk '{for (i = 1; i <= NF; i++) print $i}' temp.CHECK > new.CHECK  
  
$ cat new.CHECK | tr -cd '![a-zA-Z]\n' > empty.CHECK  
  
$ sed '/!-/d' empty.CHECK > temp.CHECK  
  
$ sed '/^\s*$/d' temp.CHECK > CHECK
```

The reason for simplifying both files is that some control characters made the C programs crash. As the purpose of this article is to showcase hash tables, I decided to simplify the input instead of spending time trying to figure out the problem and modifying the C code.

After constructing the hash table using the first file (INPUT) as input, the second one (CHECK) will be used for testing the hash table. This will be the actual use of the hash table.

Theory

Let me start with the definition of a hash table. A hash table is a data structure that stores one or more key and value pairs. A hash table can store keys of any type.

A hash table uses a hash function to compute an index into an array of buckets or slots, from which the correct value can be found. Ideally, the hash function will assign each key to a unique bucket. Unfortunately, this rarely happens. In practice, more than one of the keys will hash to the

array. Although the improvement looks small, you should realize that for a hash array with 20 buckets, the search time is now 20 times smaller.

It is important for the hash function to behave consistently and output the same hash value for identical keys. A collision happens when two keys are hashing to the same index—that's not an unusual situation. There are many ways to deal with a collision.

A good solution is to use separate chaining. The hash table is an array of pointers, each one pointing to

A HASH TABLE USES A HASH FUNCTION TO COMPUTE AN INDEX INTO AN ARRAY OF BUCKETS OR SLOTS, FROM WHICH THE CORRECT VALUE CAN BE FOUND.

same bucket. The most important characteristic of a hash table is the number of buckets. The number of buckets is used by the hashing function. The second most important characteristic is the hash function used. The most crucial feature of the hash function is that it should produce a uniform distribution of the hash values.

You can say that the search time is now $O(n/k)$, where n is the number of keys, and k is the size of the hash

the next key with the same hash value. When a collision occurs, the key will be inserted in constant time to the head of a linked list. The problem now is that when you have to search a hash value for a given key, you will have to search the whole linked list for this key. In the worst case, you might need to traverse the entire linked list—that's the main reason the linked list should be moderately small, giving the requirement for a large number

of buckets.

As you can imagine, resolving collisions involves some kind of linear search; therefore, you need a hash function that minimizes collisions as much as possible. Other techniques for resolving collisions include open addressing, Robin Hood hashing and 2-choice hashing.

Hash tables are good at the following:

- In a hash table with the “correct” number of buckets, the average cost for each lookup is independent of the number of elements stored in the table.
- Hash tables are particularly efficient when the maximum number of entries can be predicted in advance so that the bucket array can be allocated once with the optimum size and never resized.
- If the set of key-value pairs is fixed and known ahead of time (so insertions and deletions are not allowed), you can reduce the average lookup cost by a careful choice of the hash function, bucket table size and internal data structures.
- They are not good at keeping sorted data. It is not efficient to use a hash table if you want your data sorted.
- Hash tables are not effective when the number of entries is very small, because despite the fact that operations on a hash table take constant time on average, the cost of a good hash function can be significantly higher than the inner loop of the lookup algorithm for a sequential list or search tree.
- For certain string processing applications, such as spell-checking, hash tables may be less efficient than trees or finite automata.
- Although the average cost per operation is constant and fairly small, the cost of a single operation may be fairly high. In particular, if the hash table uses dynamic resizing, inserting or deleting a key may, once in a while, take time proportional to the number of entries. This can be a serious drawback in applications where you want to get results fast.
- Hash tables become quite inefficient when there are many collisions.

Hash tables also have some disadvantages:

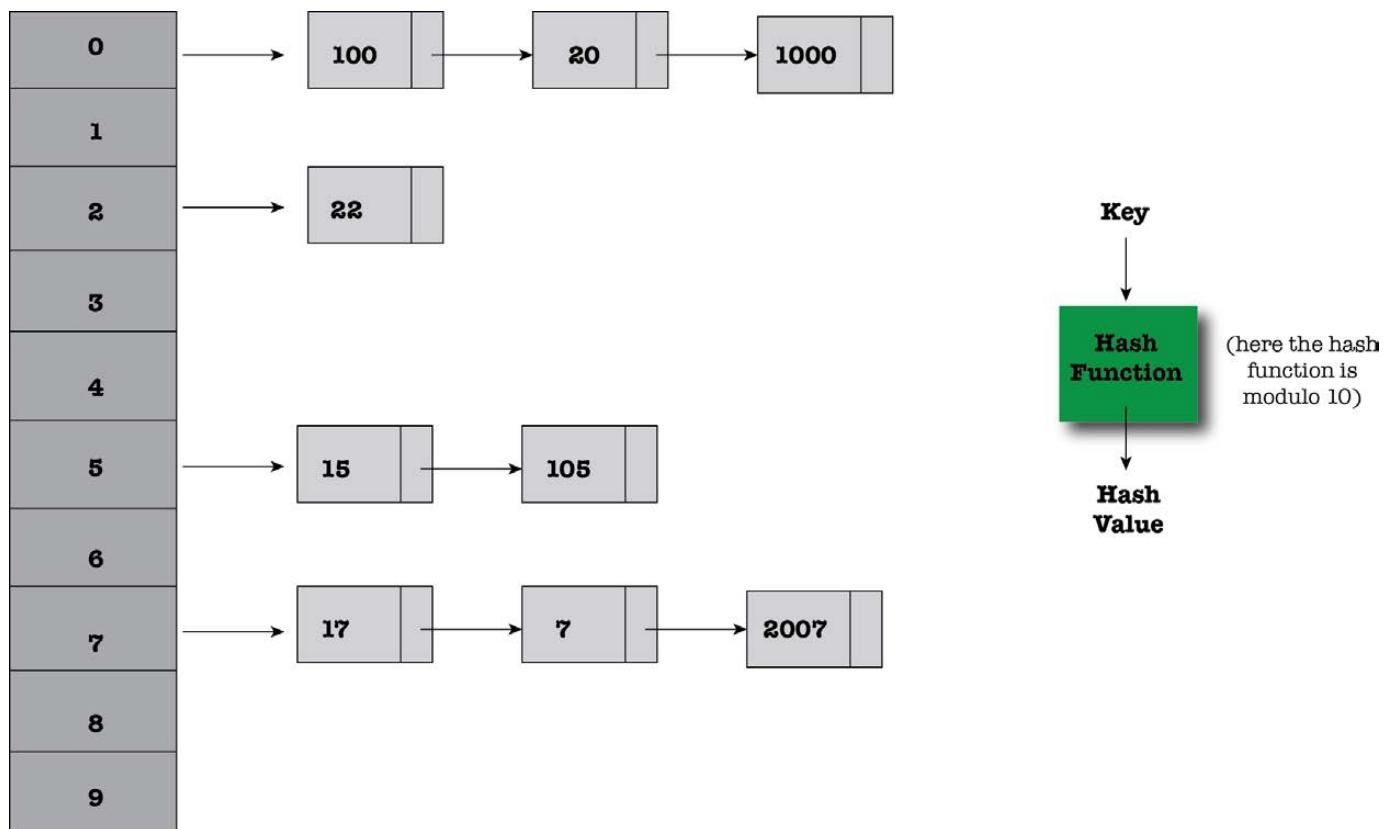


Figure 1. A Simple Hash Table

As I'm sure you understand, not every problem can be solved equally well with the help of a hash table. You always should consider and examine all your options before deciding what to use.

Figure 1 shows a simple hash table with keys and values shown. The hash function is the modulo 10 function; therefore, ten buckets are needed because only ten results can come from a modulo 10 calculation. Having only ten buckets is not considered very good, especially if the number of

values grows large, but it is fine for illustrative purposes.

To summarize, a hash table should follow these principles:

- Do not have too many buckets, just as many as needed.
- It is good for the hash function to take into account as much information provided by the key as possible. This is not a trivial task.
- The hash function must be able to hash similar keys to different

hash values.

- Each bucket should have the same number of keys or at least as close to being equal as possible (this is a very desirable property).
- Following some principles will make collisions less likely. First, you should use a prime number of buckets. Second, the bigger the size of the array, the smaller the probability of collisions. Finally, you should make sure that the hash function is smart enough to distribute its return values as evenly as possible.

Delete, Insert and Lookup

The main operations on a hash table are insertion, deletion and lookup. You use the hash value to determine where in the hash table to store a key. Later, you use the same hash function to determine where in the hash table to search for a given key.

Once the hash table is populated, searching is the same as doing an insertion. You hash the data you are searching for, go to that place in the array, look down the list that starts from that location, and see if what you are looking for is in the list. The number of steps is $O(1)$.

The worst-case search time for a hash table is $O(n)$, which can happen when all keys are stored in the same bucket. Nevertheless, the probability of that happening is so small that both the best and average cases are considered to be $O(1)$.

You can find many hash table implementations on the Internet or in several books on the topic. The tricky part is using the right number of buckets and choosing an efficient hash function that will distribute values as uniformly as possible. A distribution that is not uniform definitely will increase the number of collisions and the cost of resolving them.

A C Implementation

The first implementation will be stored in a file named ht1.c. The implementation uses separate chaining, because separate chaining is a reasonable choice. For simplicity, both input and output filenames are hard-coded inside the program. After finishing with the input and building the hash table, the program starts reading the second file, word by word, and starts checking whether a word can be found in the hash table.

Listing 1 shows the full C code of the ht1.c file.

Listing 1. ht1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define TABLESIZE 5

// Linked List
typedef struct node
{
    char *data;
    struct node *next;
} node;

// A Hash Function: the returned hash value will be the
// ASCII value of the first character of the string
// modulo the size of the table.
unsigned int hash(const char *str, int tablesiz)
{
    int value;

    // Get the first letter of the string
    value = toupper(str[0]) - 'A';

    return value % tablesiz;
}

static int lookup(node *table[], const char *key)
{
    unsigned index = hash(key, TABLESIZE);
    const node *it = table[index];

    // Try to find if a matching key in the list exists
    while(it != NULL && strcmp(it->data, key) != 0)
    {
        it = it->next;
    }
    return it != NULL;
}

int insert(node *table[], char *key)
{
    if( !lookup(table, key) )
    {
        // Find the desired linked list
        unsigned index = hash(key, TABLESIZE);
        node *new_node = malloc(sizeof *new_node);

        if(new_node == NULL)
            return 0;

        new_node->data = malloc(strlen(key)+1);

        if(new_node->data == NULL)
            return 0;

        // Add the new key and link to the front of the list
        strcpy(new_node->data, key);
        new_node->next = table[index];
        table[index] = new_node;
        return 1;
    }
    return 0;
}

// Populate Hash Table
// First parameter: The hash table variable
// Second parameter: The name of the text file with the
// words
int populate_hash(node *table[], FILE *file)
{
    char word[50];
    char c;

    do {
        c = fscanf(file, "%s", word);
        // IMPORTANT: remove newline character
        size_t ln = strlen(word) - 1;
        if (word[ln] == '\n')
            word[ln] = '\0';

        insert(table, word);
    } while (c != EOF);

    return 1;
}

int main(int argc, char **argv)
{
    char word[50];
    char c;
    int found = 0;

    // Initialize the hash table
    node *table[TABLESIZE] = {0};

    FILE *INPUT;
    INPUT = fopen("INPUT", "r");
    // Populate hash table
    populate_hash(table, INPUT);
    fclose(INPUT);
    printf("The hash table is ready!\n");

    int line = 0;
    FILE *CHECK;
    CHECK = fopen("CHECK", "r");

    do {
        c = fscanf(CHECK, "%s", word);

        // IMPORTANT: remove newline character
        size_t ln = strlen(word) - 1;
        if (word[ln] == '\n')
            word[ln] = '\0';

        line++;
        if( lookup(table, word) )
        {
            found++;
        }
    } while (c != EOF);

    printf("Found %d words in the hash table!\n", found);

    fclose(CHECK);
    return 0;
}
```

An Even Better C Implementation

The second implementation will be stored in a file named ht2.c. This implementation uses separate chaining as well. Most of the C code is the same as in ht1.c except for the hash function.

The C code for the modified hash function is the following:

```
int hash(char *str, int tablesize)
{
    int sum = 0;

    // Is it a valid string?
    if(str == NULL)

    {
        return -1;
    }

    // Calculate the sum of all characters in the string
    for( ; *str; str++)
    {
        sum += *str;
    }

    // Return the sum mod the table size
    return (sum % tablesize);
}
```

What this hash function does better than the other one is that it takes into account all the letters of the string instead of just the first one. Therefore, the produced number,

which corresponds to the position of the key in the hash table, is bigger, and this results in being able to take advantage of hash tables with a larger number of buckets.

Benchmarks

The presented benchmarks are far from accurate or scientific. They are just an indication of what is better, what works and what doesn't and so on. Keep in mind that finding the optimal hash table size is not always easy.

All programs were compiled as follows:

```
$ gcc -Wall program.c -o program
```

The trusty time command produced the following output after executing ht1 with four different hash table sizes:

```
$ grep define ht1.c
#define TABLESIZE 101
$ time ./ht1
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.401s
user    0m0.395s
sys            0m0.004s
$ grep define ht1.c
#define TABLESIZE 10
```

```
$ time ./ht1
The hash table is ready!
Found 59843 words in the hash table!

real    0m0.794s
user    0m0.788s
sys     0m0.004s

$ grep define ht1.c
#define TABLESIZE 5
$ time ./ht1
The hash table is ready!
Found 59843 words in the hash table!

real    0m0.404s
user    0m0.004s
sys     0m0.004s

$ grep define ht1.c
#define TABLESIZE 1001
$ time ./ht1
The hash table is ready!
Found 59843 words in the hash table!

real    0m1.454s
user    0m1.447s
sys     0m0.004s

real    0m0.410s
```

Figure 2 shows a plot of the execution times from the four

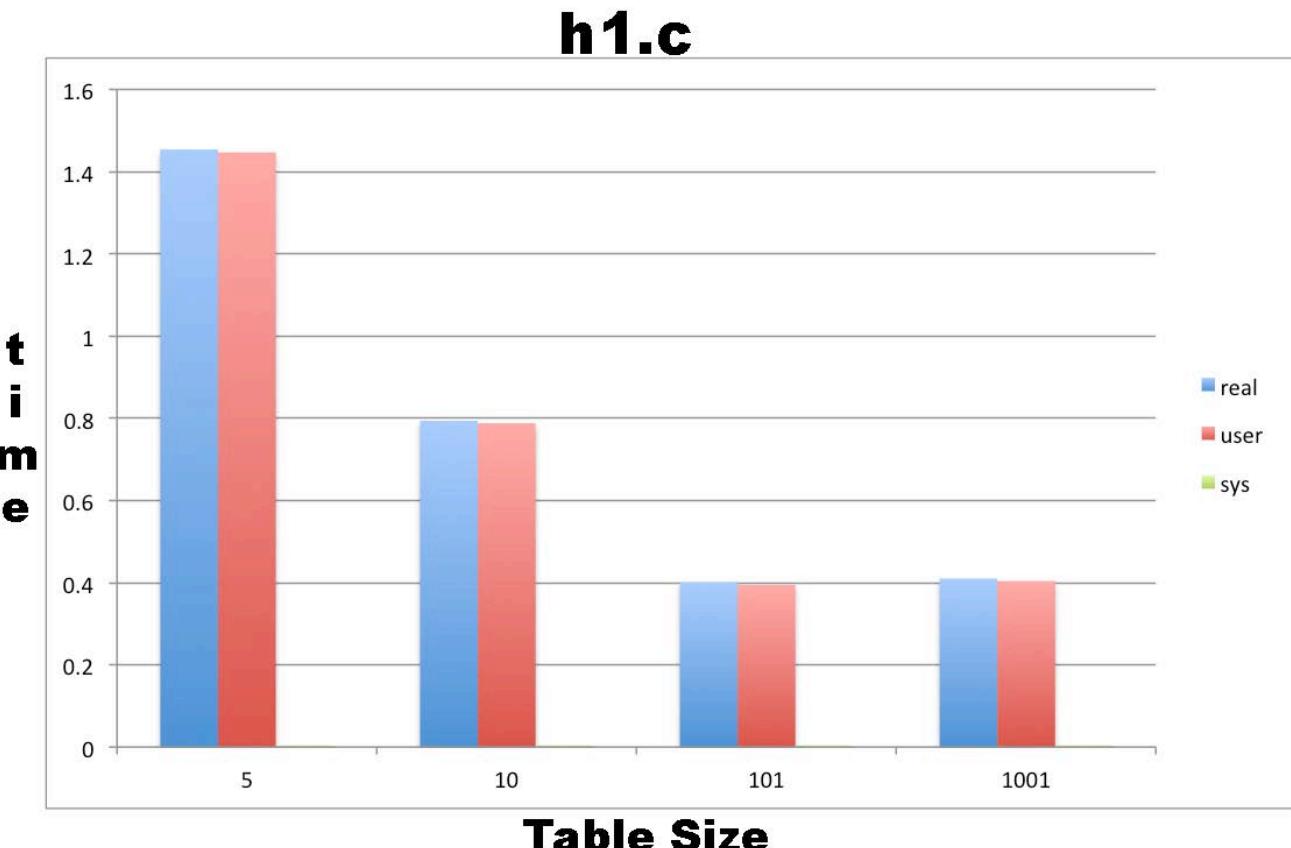


Figure 2. Execution Times from the Four Different Values of the TABLESIZE Variable of the ht1.c Program

different values of the TABLESIZE variable of the ht1.c program. The bad thing about ht1.c is that its performance with a hash table of 101 buckets is almost the same as with one with 1,001 buckets!

Next, here are the results from the execution of the ht2.c program:

```
$ grep define ht2.c
#define TABLESIZE 19
$ time ./ht2 INPUT CHECK
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.439s
user    0m0.434s
sys     0m0.003s
$ grep define ht2.c
#define TABLESIZE 97
$ time ./ht2 INPUT CHECK
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.116s
user    0m0.111s
sys     0m0.003s
$ grep define ht2.c
#define TABLESIZE 277
$ time ./ht2 INPUT CHECK
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.072s
user    0m0.067s
```

```
sys     0m0.003s
$ grep define ht2.c
#define TABLESIZE 997
$ time ./ht2 INPUT CHECK
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.051s
user    0m0.044s
sys     0m0.003s
$ grep define ht2.c
#define TABLESIZE 22397
$ time ./ht2 INPUT CHECK
The hash table is ready!
Found 59843 words in the hash table!
```

```
real    0m0.049s
user    0m0.044s
sys     0m0.003s
```

Figure 3 shows a plot of the execution times from the five different values of the TABLESIZE variable used in the ht2.c program. All hash table sizes are prime numbers. The reason for using prime numbers is that they behave better with the modulo operation. This is because a prime number has no positive divisors other than one and itself. As a result, the product of a prime number with another integer has fewer positive divisors than the product of a non-prime number with another integer.

h2.c

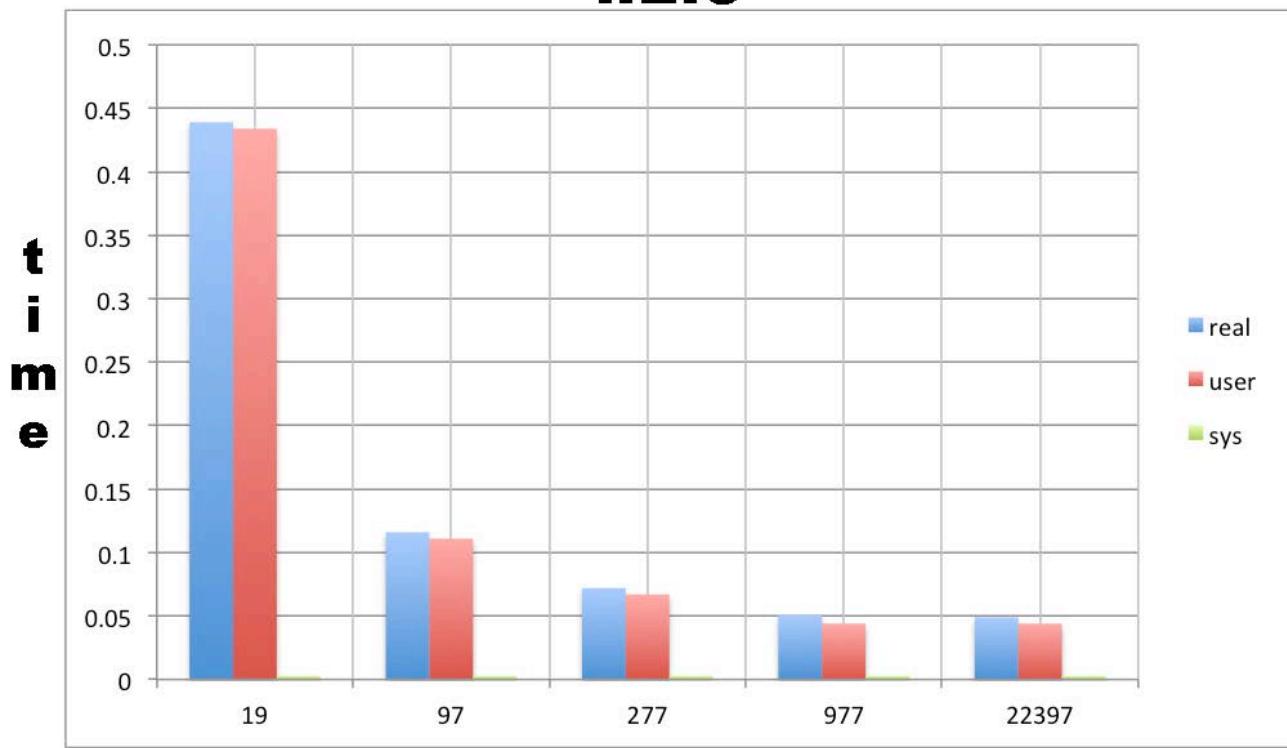


Figure 3. A Plot of the Execution Times from the Five Different Values of the TABLESIZE Variable Used in the ht2.c Program

As you can see, the new hash function performs much better than the hash function found in ht1.c. As a result, the use of more buckets greatly improves the performance of the hash table. Nevertheless, as long as the words in the text files are finite, there is no point in using more buckets than the number of unique words in the input file.

It is useful to examine the distribution of keys in the hash table for the ht2 implementation using two different number of buckets.

The following C function prints the number of keys in each bucket:

```
void printHashTable(node *table[], const unsigned int tablesize)
{
    node *e;
    int i;
    int length = tablesize;
    printf("Printing a hash table with %d buckets.\n", length);

    for(i = 0; i<length; i++)
    {
        // printf("Bucket: %d\n", i);
        // Get the first node of the linked list
```

```

// for the given bucket.

e = table[i];

int n = 0;

if (e == NULL)
{
    // printf("Null bucket %d\n", i);

}
else
{
    while( e != NULL )
    {
        n++;

        e = e->next;
    }
}

printf("Bucket %d has %d keys\n", i, n);
}
}

```

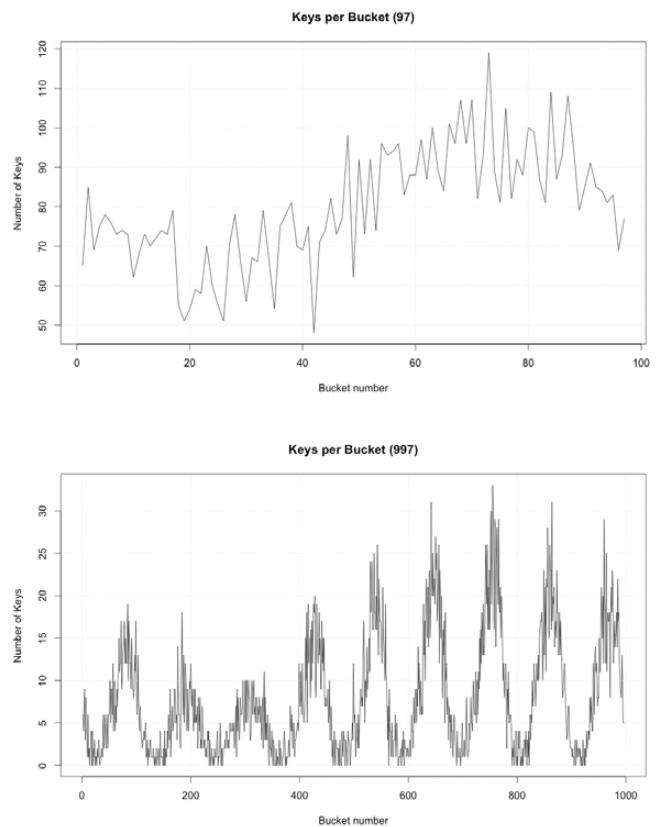


Figure 4. The Number of Keys in Each Bucket for Two Hash Tables with Different Numbers of Buckets

Figure 4 shows the number of keys in each bucket for two hash tables: one with 97 buckets and the other with 997 buckets. The hash table with 997 buckets appears to follow a pattern on how it fills its buckets, whereas the hash table with the 97 buckets is more evenly distributed. Nevertheless, the bigger hash table has a lower number of keys in each bucket which is what you really want because less keys in each linked list means less time searching it.

Summary

Hash tables are an important part of

computer science and programming. I hope this article helps you understand their importance and clarifies some things about them. ■

Mihalis Tsoukalos is a UNIX administrator, programmer (UNIX and iOS), DBA and mathematician. You can reach him at <http://www.mtsoukalos.eu> and @mactsouk.

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

WEBCASTS



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

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

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



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

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

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

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

WHITE PAPERS



White Paper: JBoss Enterprise Application Platform for OpenShift Enterprise

Sponsor: DLT Solutions

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

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

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

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

WHITE PAPERS



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

Sponsor: Red Hat | **Topic:** Linux Management

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

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



Standardized Operating Environments for IT Efficiency

Sponsor: Red Hat

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

Benefits of an SOE:

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

SOE leads to:

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

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

Using MySQL for Load Balancing and Job Control under Slurm

How to process a decade of satellite data without losing all of your mind.

STEVEN BUCZKOWSKI

Like most things these days, modern atmospheric science is all about big data. Whether it's an instrument flying in an aircraft taking sets of images several times a second and producing three quarters of a terabyte of data per flight day over a two-week campaign or a satellite instrument producing hundreds of gigs of spectral data daily over a 10–15 year lifetime, data volume is enormous. Simply analyzing a day's worth of data to keep track of basic instrument stability is CPU-intensive. Fully processing a day to retrieve the state of the atmosphere or looking at

trends across a decade's worth of data is exponentially so.

High-performance parallel cluster computing is the name of the game. For years I've done this on a very basic level by kicking off a handful of copies of my processing scripts on a couple computers around the lab, but after a recent move into a new lab, I got my first chance to work on a real cluster system, processing data from a satellite-borne hyperspectral sounder called AIRS (see Resources). AIRS is one of the instruments onboard NASA's AQUA satellite that was launched in late 2002 and has been

in continuous operation since. Data from AIRS and similar instruments is used to map out vertical profiles of atmospheric temperature and trace gases globally, but we have to be able to process it first.

The cluster computing game here is strictly to get a whole lot of computers doing the same thing to a whole lot of data so that we can process it faster than we collect it (much faster would be preferable). Since I was new to this game just a few months ago, I've had much to learn about cluster computing and how to design algorithms and processing software to take advantage of multiple CPUs for processing. This was my first experience where I had hundreds of CPUs at my disposal, and it really has changed how I process data in general. I started this article to describe how I was shown to parallelize this type of data processing and a method I put together that makes the process much cleaner.

Basic Slurm

The cluster system here consists of 240 compute nodes, each with dual, 8-core processors and 64GB of main memory running Red Hat Enterprise Linux. Cluster jobs are scheduled to run through the Slurm workload manager (see Resources). In a

Listing 1. Top-level Slurm/sbatch script.
This scripts specifies the request for CPUs, memory and time on the cluster and sets up the program that will be run on each processing node.

```
#!/bin/bash
# sbatch options
#SBATCH --job-name=RUN_CALFLAG
#SBATCH --partition=batch
#SBATCH --qos=short
#SBATCH --ntasks=20
#SBATCH --mem-per-cpu=18000
#SBATCH --cpus-per-task 1
#SBATCH --time=00:20:00

# matlab options
MATLAB=/usr/bin/matlab
MATOPT=' -nojvm -nodisplay -nosplash'

srun --output=~/run_calflag.log
→$MATLAB $MATOPT -r "run_calflag; exit;"
```

nutshell, Slurm is a suite of programs that works to allocate computer resources among users and compute jobs and enforce sharing rules to make sure everyone gets a chance to get their work in. The two most important programs in the suite for actually working on the system are `sbatch` and `srun`.

`sbatch` is the entry point to the Slurm scheduler and reads a high-level Bash control script that specifies job parameters (number of nodes needed,

memory per process, expected run times and so on) and spawns the requested number of identical jobs via calls to `srun`.

The script in Listing 1 asks Slurm to allocate 20 processors (`ntasks/cpus-per-task`), allocate 18GB of memory (`mem-per-cpu`) to each and run a job (`srun ...`) on each that will take around 20 minutes (`time`) to run. The `partition` and `qos` directives help the system manage its resources and set rules for the number of processors a user is allowed, CPU time limits and so on. The `job-name` directive puts a name to your task to help you separate your jobs in an `squeue` list of the system queue.

The `srun` request shown here is to run an instance of MATLAB on each of the allocated nodes and, in each instance, run the script `run_calflag` and exit. Any message output is sent to the file specified by the `output` parameter. `run_calflag` could be a simple “hello world” script, or it could be a loop to process a thousand files. It also doesn’t have to be MATLAB. MATLAB is our tool of choice here, and examples in this article use it in a background sort of way. There is no need to understand MATLAB to keep reading.

As long as this request doesn’t violate any cluster good behavior

rules by hogging processors, hogging memory and so on, Slurm queues the request until such time as processors are available to run it. When the resources are available, Slurm starts the processing by grabbing 20 CPUs and, then, starting a copy of `matlab/run_calflag` on each one. Once the control scripts are in order, this request is submitted to Slurm through the `sbatch` command:

```
sbatch run_calflag_batch.sh
```

Slurm also manages a set of environment variables that can be used to pass some job parameters into the processing scripts.

Basic Data Chunking

Listing 1 already shows enough to see one issue with parallelizing this kind of data processing: how does one chunk up the data to pass to each instance of the `run_calflag` started by `srun`? If I want to process one year, should I ask for 365 processors and do one day on each? 52 processors and one per week? One per month? How do I handle leap years? The cluster resource allocation rules prevent me from doing the 365 processor idea, but other than that, there is no clear, single answer.

Being new to cluster computing,

I looked at what my colleagues have done to set up their processing runs. Most have adopted a three-tier system to run these jobs:

- A bash script with `sbatch` directives and `srun` calls to kick off everything.
- A MATLAB script (called by `srun` in the previous script) that is run on each compute node that uses some node ID in the Slurm environment (usually `SLURM_PROC_ID`) to index processing into the range of years/days/files to process. The most common approaches are to request 12 CPUs and assign one month per CPU or request 52 and assign a week per CPU. This script then loops over the years/days/files “assigned” to this node. Within this loop, calls are made to a final MATLAB script that does the actual processing for each year/day/file in the sequence.

This approach certainly can work, but it has some significant issues:

- Ad hoc chunking of data: in general, how does one split “x” things to process over “y” nodes? In practice, this seems to mean you have to edit run scripts and tailor

them to just about every run you wish to do. (It is almost guaranteed that you will do multiple runs in this business: once to try to process a contiguous period of data and a second time to re-run the now non-contiguous set of days that failed for one reason or another.)

- Does not utilize allocated system resources well: parallelizing by month, the node processing February is pretty much guaranteed to sit idle for 8–10% of the total run time simply because it’s 2–3 days shorter than other months. Or, if the lower-level processing fails and takes out the entire process running on that CPU, the rest of that chunk will need to be reprocessed later *and* the processor sits idle while the rest of the processors finish.
- Does not utilize *my* time very well: any time I need to change either the number of processing jobs or the number of nodes to spread them over, I have to recalculate manually how to spread out the job and, likely, edit my control scripts. I absolutely hate having to keep a dozen scripts lying around all to do the same thing but each for some special edge case.

Okay, then, how do we get around this?

Job Control Stack

What is really needed is some sort of job scheduling stack where one could store whatever parameters are required for each atomic processing step. Whatever information would be needed to find the right data to process could be pushed onto this list, and the various scripts could then pop the next unprocessed item whenever a node becomes free. Let's say the data we need to process is archived one file per day and we want to process several years of it. The job control stack idea is that we can make a list of the files we want to process. This may be as simple as running `ls` or `find`. We then can push their details onto the stack and start processing them in parallel. We ask Slurm to provide us with some processors and start MATLAB on each along with our second-tier processing script. Each instance is a loop not unlike before, but instead of processing some ad hoc chunk of the data, each iteration of the loop queries the stack for the next available unprocessed data. If there is data to process, it gets passed to the low-level processing script. If not, the loop exits, and MATLAB terminates. The system

doesn't care if we have 365 days to process or 2,431, or whether we want to spread those across two or 200 processors. This means that we don't have to care either.

When the main job is done, and we find that some number of days didn't process properly, we just make a list of the failed files, push it into the job stack and run it again (after fixing the reason for the initial failure, of course). There should be very little reason to edit any of the scripts to do this.

This idea seems to solve all the big issues:

- No ad hoc chunking of data: one process, one file. Repeat as necessary.
- Node resource utilization is higher and more even: if a processor hits a series of days that process quickly, it just grabs more days to process until there are no more.
- If a node dies—either because of a system error or because of some other inadequately trapped error (missing files, short files/array indexing problems)—the processing balances over the remaining nodes. You lose and need to reprocess one day's worth of data, but the rest of

the data gets done elsewhere.

MySQL Implementation

It would be nice to be able to do this with a flat file for simplicity, but this approach led to some serious issues with collisions between different

processors grabbing the same data to process. The need to be able to lock data once a processor has grabbed it led me to implementing this job stack with a MySQL database table.

Listing 2 is a MySQL CREATE TABLE command to create the job control table as it is currently implemented. Most of the values in this table are populated when runs are pushed onto the stack. node_start and node_id are populated when the job is popped from it and begins active processing. node_end is filled in at the completion of a run. node_start, node_end and node_id aren't particularly necessary, but they collect useful statistics about runtime performance (although you can get the same information from Slurm's sacct command).

Jobs to be run are pushed to the stack by a routine like that shown in Listing 3. When a job is pushed to the stack, task_id takes on the process

Listing 2. SQL CREATE TABLE Script to Define the Basic Job Control Table

```
CREATE TABLE `JobManagement` ( `task_id` int(11) NOT
➥NULL, `entry_id` int(11) NOT NULL AUTO_INCREMENT,
➥`node_id` int(11) DEFAULT NULL, `node_start` timestamp
➥NOT NULL DEFAULT '0000-00-00 00:00:00', `node_end`
➥timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
➥`datapath` varchar(256) DEFAULT NULL, `task_name`
➥varchar(128) NOT NULL, PRIMARY KEY (`task_id`,`entry_id`) )
➥ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
```

Listing 3. Job control stack push function implemented in MATLAB. sDataPath is the absolute path to the data file to process. This gets pushed onto the job control table to make it available for processing.

```
function push_job_table(iTaskID, sJobName, sDataPath)

sMYSQL = 'mysql -h myhost -u myuser -p<password> myDB';

sSQL = sprintf(['echo "insert into JobManagement (task_id, year, '
    'doy, task_name) values (%d, \\\"%s\\\", \\\"%s\\\");"'],
iTaskID, ...
    sDataPath, sJobName);

[status, cmdout] = system([sSQL ' | ' sMYSQL]);
```

ID Slurm assigns to the overall processing run, and `entry_id` gets assigned a counter that starts at 1. This counter increments with each processing run added under that `task_id`. Taken together, `entry_id` and `task_id` are the primary key for the table and, thus, are unique for every record in the table. The actual information needed to retrieve data to process is stored in `datapath`. `datapath` specifies an absolute path to the primary data file that needs to

be processed and can be populated directly from `ls` or `find` in most cases but can also come from routines with more sophisticated logic if we need to match up multiple input files or check that files exist first and so on. When a processor queries the database for a new job, as in Listing 4, the value of `datapath` is the primary information that gets returned.

Listing 4 gives us the `pop` function for our stack. The idea in the `pop` is to select the next available row

Listing 4. Job control stack pop function implemented in MATLAB. Queries job control table for next available data file to process. If data is available, the path to it is returned. If none is available, returns an empty string causing further processing to end.

```
function stJobEntry = pop_job_table(taskid, entryid)

% determine which node we are on for table update (for performance
% tracking)

iNodeID = str2num(getenv('SLURM_PROCID'));

sMYSQL = 'mysql -h myhost -u myuser -p<password> myDB';

% select one entry from the job management table that hasn't been
% done yet. Immediately update with this processor's node_id to try
% to lock the row out from further requests.

sPOPSQL = sprintf(['echo "set @B=%d; set @C=%d;' ...
    'select @A:=entry_id as entry_id, datapath from ' ...
    'JobManagement where task_id = @B and entry_id = @C' ...
    'and node_id ' ...
    'is null limit 1 for update;" ...
    'update JobManagement set node_id = %d,' ...

    'node_start = now() where task_id = @B and ' ...
    'entry_id = @A;' ...
    'set @A=0;""], taskid, entryid, iNodeID);

% execute the table pop and get a day that is not being processed
[status, cmdout] = system([sPOPSQL ' | ' sMYSQL ' | head -2 | tail ' ...
    '-1']);

stJobEntry = struct('entry',[], 'datapath','');

if length(cmdout) > 0
    % parse cmdout to entry_id, datapath
    iTokens = str2num(cmdout);
    stJobEntry.entry = iTokens(1);
    stJobEntry.datapath = iTokens(2);
end
```

in the database and to lock it for update. The record is fully locked out by updating it with the `node_id` for the processor grabbing it and also the start time for processing. This is done in one command to minimize the CPU time in which another processor can grab this same record. Once the `node_id` has been set, the record never will come into consideration again.

The record retrieved or, really, the datapath stored within it, is passed out to a MATLAB structure

that gets returned to the calling function where it will be used to start processing. `push` and `pop` are all that is really necessary, but because we went to the effort to include `node_stop` to track runtime, a routine to close out the job table entry is needed. Listing 5 shows one version of this close-out that simply updates the record to add the end time in the `node_stop` field. No explicit locking is needed here.

Some readers probably are wondering why I am doing database

LINUX JOURNAL

now available
for the iPad and
iPhone at the
App Store.



Available on the
App Store

linuxjournal.com/ios



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

Listing 5. Once processing for a job table entry is complete, update the table with the time processing ended for basic runtime statistics.

```
function close_job_table_entry(iTaskID, iEntryID)

sMYSQL = 'mysql -h myhost -u myuser -p<password> myDB';

% close out day by adding processing end time to db table
sSQL = sprintf(['echo "update JobManagement set node_end=now() ' ...
    'where task_id = %d and entry_id = %d;"'], ...
    iTaskID, iEntryID);

[status, cmdout] = system([sSQL ' | ' sMYSQL]);
```

work through shell commands and `system()` escapes instead of some native database access. It's for three reasons really. First, I'm lazy, and I couldn't get any sort of MATLAB-native database connectivity to work. Second, this way I easily can develop and test at the command line and then paste into my code, and finally, I find this approach more easily portable. If we ever decide to become an IDL or Python shop, I can have these routines ported as quickly as I can look up their respective system calls. System shell escapes may be the slower, but in this use, they add only minutes onto runs that take tens of hours. There is not much point in chasing down those few minutes.

Wrap Up

Hopefully you now are convinced that a job control table is the best way to run jobs under Slurm. This approach has improved processing times significantly on several things we process routinely. Runs that used to take days now finish in one day, and I am not spending any time at all rewriting scripts for every run, so it definitely has worked for me. ■

As a chronic suffer of Multiple Avocation Disorder, Steven lives life in a desperate search for days with more hours in which to stuff new activities and splits his time as best he can between physics, software engineering, photography, cycling, beekeeping, ...

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

Resources

AIRS: <http://airs.jpl.nasa.gov>

Slurm: <http://slurm.schedmd.com/slurm.html>



Where every interaction matters.

break down your innovation barriers

power your business to its full potential

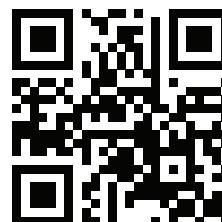
When you're presented with new opportunities, you want to focus on turning them into successes, not whether your IT solution can support them.

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

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

Want more on cloud?

Call: 844.855.6655 | go.peer1.com/linux | View Cloud Webinar:





DOC SEARLS

Three More Lessons

Some Linux-leveraged lessons for high school seniors.

[In June 2015, I gave a commencement address to the graduating class of High Mowing School in New Hampshire (<http://www.highmowing.org/page>). I wrote many drafts for the talk, all toward extemporizing the final thing. My experience with Linux and open-source hackers had an influence on it and gets credit as well. That's why I'm sharing it here.—Doc]

Many years ago the comedian Tom Novello played a character on *Saturday Night Live* named Father Guido Sarducci, the Vatican's gossip columnist. He'd stand on stage in his black robes and flat-brimmed hat, smoking a cigarette and sharing crazy ideas that somehow made sense.

One of them was the Five Minute University (<https://www.youtube.com/watch?v=kO8x8eoU3L4>). Students would be taught just the few things they'd remember long after they graduate. For example, if you take two years of Spanish and don't use it, after a while, all you'll remember are "Como estas?" and "Muy bien." If you take Economics,

all you'll remember are Supply and Demand. So just teach those. So I taught you two courses right there, in 15 seconds—funny stuff. Look it up on YouTube.

Better yet, look up this talk. It's called *Three More Lessons*. Five minutes each, because all I've got is 15 minutes. If I succeed, I'll give you three things to remember and maybe you can use. By the way, I'm giving everybody an A. So make that the first thing you forget. You don't need it anyway.

First lesson: *Humility*.

Last month I attended my high school's 50th reunion. Yes, I'm *that* much older than you guys.

I wish my school had been like High Mowing. But it wasn't.

My high school was a Lutheran academic correctional institution—or at least it was for me. It was called Concordia Prep. Concordia is to Lutherans what Loyola is to Catholics—something like that.

I was sent there because I was a bad student. I didn't pay attention, didn't do my homework, didn't follow rules...didn't do anything much, other than what I was interested in. It helped that my interests were mostly academic. I just wasn't tuned in to the subject of whatever class I hated being in at the time.

These days, they would have had a diagnosis for me and given me drugs. But in those days, there was no such luck. They had a single diagnosis for every failing student: you were **bad**. And, if you were bad and male, only two things could happen: 1) you'd get sent to a vocational-technical high school to learn a "trade" like auto mechanics or carpentry; or 2) you'd get sent away to a religious or military school. I got sent to a religious one.

One reason I was bad was really stupid: I actually thought I was smarter than everybody else.

There was no proof of that. But I believed it. I always thought I could learn anything just by reading up on

it. In the story of the tortoise and the hare, I identified with the hare. But I actually wasn't a hare. I was just a world-class procrastinator. I think by high school I was still putting off stuff they failed to teach me in fifth grade, like English grammar and long division.

About half the students at Concordia Prep were misfits like me, or so I thought at the time. The rest were young seminarians: boys committed to the ministry. But here's something many of them had in common: they were a lot smarter than me.

Take my roommate Paul. He was an outstanding student. Here's how good he was: when we were seniors, and the school's biology teacher went down with an injury, Paul came off the bench and taught the class, for *months*, as I recall. Could be he was actually better at the subject than the teacher he replaced. Paul was also the school's best musician, best writer and one of the wittiest guys I have ever known. He went on to write many books, serve as professor of homiletics (that's what I'm doing now, preaching) at the Yale Divinity School (where he also directed the Yale Institute of Sacred Music), and then to serve for decades as the Episcopal bishop of Bethlehem, Pennsylvania (<http://www.episcopalchurch.org/staff/rt-rev-paul-v-marshall>).

Another good friend, also named Paul, was a year behind me—a Junior when I was a senior. This Paul was a great guy, and also very brainy. I remember sitting next to him when I took the SAT test for the third or fourth time, hoping to bring my scores up to a level where some college, somewhere, might accept me. Paul was taking it for the first time, I think.

I remember sitting in the room there, with our tests and answer sheets. Paul was in the seat next to me. When the first test started, I was on about the third question when Paul was already on the next page, answering every question like he was being asked the time. All the answers were obvious to him, while I'm thinking, "Okay, I'm sure B is wrong, but A, C and D look like they might be right...."

I did finish on time, sort of. See, I had once space left at the end of the answer sheet, even though I had gone through all the questions on the test. Then I remembered that I had skipped a question on the test, but not on the answer sheet. Then the voice came down from the front of the room: "Stop. Put down your pencils...."

Afterward I hung around while Paul took the Chemistry test as well. He had been taking Chemistry only for two months, but he was sure he

could ace it, because he read the book. Afterwards I asked how he thought he did.

"There was one question that could have gone either way", he said. "So I went with what I thought the people who made the test would have thought was right." As I recall, he aced the test, along with the other SAT sections.

But here's the thing that mattered. He was humble about it. Humility, C. S. Lewis said, is the opposite of pride, which is a sin. In case you're wondering, Paul is now the pastor of a church in California (http://locator.lcms.org/nchurches_frm/c_detail.asp?C359145).

I got another lesson in humility from the rest of my classmates, who voted a large percentage of themselves to one distinction or another: Most Popular, Most Likely to Succeed, Wittiest, Best Dressed, Best Athlete, Best Looking, Best All Around and so on. I was none of those and didn't deserve to be. If the other guys could have voted for Least Likely to Succeed, I might have won.

Now, 50 years later, I had a lot of successes to talk about at the reunion. You know...if it came up. I was prepared to be humble about those successes, but I also made sure I had plenty of my old Harvard business

cards to hand out. Just a *little* bit of pride there—that's right: one of the seven deadlies.

Guess how many of the guys showed up for the reunion?

One. Uno. Me.

Also present, sort of, were five dead guys whose yearbook pictures they posted on something called the Memorial Wall. That was their distinction, finally: being dead.

And here I was, being humbled, involuntarily.

But they are still alive: in me. So were a lot of other friends at the school. Here's how.

My friend Bob helped me get up the courage to ask a girl for a date for the first time—and how to dress for it. My friend Steve taught me how to wrestle. I sucked at it, but I learned from it. My friend Bill and I together created our own version of what today we call Ultimate Frisbee, using the football field. My friend AJ taught me all kinds of stuff about music. My roommate Paul taught me how to write and how to be funny. The other Paul taught me to kick ass at chess—every ass but his, that is. I beat him once. He didn't see my rook. It was huge.

But the biggest gift came one day when we were sitting around talking about what we would do in life. What

it was that each of us brought to the table. My roommate Paul said, "What David has...(That's my real name, by the way. David.)...is insight."

I hadn't known that. Really. That was new to me. But I also knew it was true. And that set me on my path.

Sound familiar? The High Mowing tagline is "Find your own path." It's the one way my school and High Mowing turned out to be the same.

I assume you are sitting here among the best friends you've ever had. I expect you all to continue supporting each other in the coming years. In fact, I want to see you here at *your* 50th reunion, in the year 2065. Promise me that.

If you keep that promise, you'll illustrate....

Second lesson: Relationships are priceless.

One evening in the year 2000, I was standing at a gate in Los Angeles, ready to board a flight to New York. As often happens, there was a problem. The airline needed to change planes and gates. They told us to hurry over to a far gate and just get on board and take any seat we could. I was at the front of the original line, but at the back of the line at the other gate—and then the last one on the plane. The only seat left was one in the back row.

What followed were five of the most remarkable hours of my life. My seatmate's name was Sayo Ajoboye, and he was a pastor from Nigeria who had just finished translating the Bible to his native language of Yoruba. He also spoke many other languages, including English, which he knew better than I did.

At the time I was flying to New York, on a speaking tour for a book I had co-written that was already a business bestseller. He asked to see it, and I showed him a chapter I wrote called "Markets are Conversations". He told me that one-liner was pretty good for a privileged guy from the first world. When I asked him what he meant by that, he gave me a Socratic lesson. "Imagine you're in a public market in a third-world country like mine", he said. "And you see something you like in one of the stalls. Let's say it's a colorful coat. What's the first thing you would say to the seller?"

I said, "What does it cost?"

He said, "Yes, you would say that, because where you come from price is everything. But where I come from it's not. There is much more going on. So let's say you two talk for a while. You tell him about your life and what you're doing there. And he tells you about his life, and the coat you

like: who made it, the dyes used, the weave and so on. After 20 minutes of that, what happens to the price?"

I said, "Maybe he wants to charge less and I want to pay more."

"Exactly", Sayo said. "Why?" I didn't have an answer.

He replied, "A relationship", and he went on to explain that relationship is one of three things that happen in a marketplace. One is transaction. Another is conversation. And the third is relationship. Then he added one more thing: "*Relationships are priceless.*"

He meant that literally. Relationships, he said, are based on the morality of love, which is about giving, rather than exchange. This, he said, is the deepest moral system in the world. In it you can only give.

He also said the morality of love was more important than the moral system we call justice. Because justice is mostly about accounting, and accounting requires prices. The moral books need to be squared up. Made even. That's why we owe favors, and pay for crimes. It's how revenge works. Getting revenge is about getting justice. We even illustrate justice with the blind lady holding the scales. Love doesn't do that. Relationships aren't about that. With love, all you can do is give.

Here is an interesting thing I learned by hanging with Eric and others who contribute to Linux and other open-source code bases: most of them are self-taught or taught by each other.

Now, lest you think this was the kind of wisdom that can only come from a preacher, about three weeks after my encounter with Sayo I found myself in a conversation with Eric S. Raymond (<http://www.catb.org/esr>), the hacker who is almost single-handedly responsible for getting the world talking about open source. As we were talking about markets being conversations, Eric added, "Markets are about three things: transaction, conversation and relationship." Eric is an atheist.

Here is an interesting thing I learned by hanging with Eric and others who contribute to Linux and other open-source code bases: most of them are self-taught or taught by each other. When I speak to groups of these guys (and they are mostly guys, alas...we need to change that), I sometimes ask if they program in languages they were taught in school. The vast majority say no. The paths they follow are ones others blazed, and they blaze as well, by sharing both code and know-how. While all this has what

Eric calls "use value", it too is an act of generosity without expectation of return—of creating and appreciating what is priceless in the world.

Third lesson: **Saving the world.**

In the fall of 2003 I was driving somewhere when I came up with a great idea I wanted to share with my mother. We usually talked about everyday stuff, but Mom was a schoolteacher and liked to go deep. She'd say, "Small minds talk about people, average minds talk about things, and great minds talk about ideas."

But then I remembered, she was dead, which was a new thing for me. She passed in August of that year at age 90. But I still heard her speak, right there, in the car. Not as a ghostly presence, but in a clear voice, the embodiment of her soul in mine. Mothers do that for us. Their voices never leave us. Maybe you've noticed that.

Here is what she said. "Give your idea to somebody else. That's why I'm gone. Your ideas are only useful for

the living. Love is for the living too. I gave love to you so you could give it to others." She was talking about what Sayo taught me three years earlier.

And here was my idea: *Save the world with our personal differences, and the network that bridges them.*

There are three parts to that, so let me unpack them for you.

First, the **world** that needs saving.

From the perspective of the planet, our species is a pestilence. A parasite. Maybe worse. Ever since industry won the industrial revolution, we have been mining, drilling and fracking with wanton abandon: not just for oil and coal, but for helium, tungsten, uranium and other substances made by stars and living things that have been extinct for most of history. This is stuff that cannot be replaced in the short term—or, in some cases, ever.

Our influence is so plain and widespread that geologists are seriously weighing a decision to rename our current geologic epoch the Anthropocene (<https://en.wikipedia.org/wiki/Anthropocene>).

We need to reverse this, before the planet reverses it for us, the hard way.

Second, our **personal differences**.

Most people in the world are very different from how we are. Throughout history, and long before,

our differences have driven us apart. The human diaspora was surely caused—at least in part (though we'll never know) by our differences. One tribe running away from another, or being driven away—to the far corners of the globe—all by differences in appearance, language, religion, custom and other stuff humans have always fought over.

Yet nothing makes us more human than the fact that we **are all different**—not only from each other, but from how we were ten minutes ago. That is because we all have our own souls, and we learn constantly as a matter of course. We are learning animals.

How we learn best is from each other. There's that relationship thing again.

To explain how important this is, consider two common words: *information* and *authority*.

Information is derived from the verb *inform*, which is derived from the verb *to form*. Think about that. We are *formed* by what we learn from other people.

We are what we know. When you tell me something I didn't know, I am, literally, formed by that. I am made larger. This is an amazing and wonderful grace we all take for granted, but it's really freaking huge.

Here is another way to look at it:

Advertiser Index

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

ADVERTISER	URL	PAGE #
ContainerCon	http://events.linuxfoundation.org/ events/containercon	23
Drupalize.me	http://www.drupalize.me	107
EmperorLinux	http://www.emperорlinux.com	29
Percona Live MySQL	http://www.percona.com/live	57
Peer 1	http://go.peer1.com/linux	97
SPTechCon Boston	http://www.sptechcon.com	45
SUSE	http://suse.com	3
Texas Linux Fest	https://2015.texaslinuxfest.org	31
Usenix Security	www.usenix.org/sec15	7

ATTENTION ADVERTISERS

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

we are all authors of each other. What we call *authority* is the right we grant others to form who we are by enlarging and changing what we know.

Over the next few years you will be growing your own authority: the right you earn to change other people, and form them with your ideas, your work, your art.

Third, **our network.**

One reason I'm standing here is that I've been studying the Internet since before it was born in its current form, in April 1995—a year or two before you guys all showed up. I watched the Internet coming the way an astronomer watches an incoming asteroid. And I've been following its impact ever since, and participating in it as well.

Everything I am known for is stuff I have done since the Internet arrived. So, in that one small way, we are all just as young. The difference is that I remember what life was like before the Internet. So let me tell you about that.

It was like civilization before fire. Or stone tools. Or weaving. Or printing. It's that big a deal.

Here's a personal example.

In 1978, I was living in the North Carolina woods, and I was broke. I had been a journalist and a radio personality, but where I lived, all those

jobs were filled. So I took whatever jobs I could get, which included working in my landlord's sawmill, because I had a hard time making the rent.

My dream back then was getting a job writing comedy for *Saturday Night Live*. So I sent Lorne Michaels, the show's producer, a heap of my material, some of which was actually good, at least as I recall.

Lorne sent the whole thing back with a note saying they don't accept unsolicited material, but they do accept unsolicited nude photographs. I was going to send him one, with a special message written on it, but in those days getting a nude photo processed in a conservative southern state was to invite a jail sentence.

I did eventually find a good job, co-founding a successful advertising agency. It was a good job that I loved and worked hard at, but it wasn't writing for *Saturday Night Live*.

The point of that story is that I had to settle. This is what human beings have always done when choices are few and contained by geography and a limited number of contacts.

Those limits are not there on the Net—or at least are severely reduced. The reason is that the Net puts everything on it at functional distance apart of zero—and at zero cost—or close enough. This is huge.

It might even be huge enough to save the world.

Because now you can share ideas and create relationships, at any distance, at little cost—with pretty much anybody, anywhere in the world, including countless people who are nothing like you. Together you can use your differences to overcome the ones we've been fighting over for the duration, and work in common cause for the whole planet and not just our one species, or any of our many tribes.

Yes, you can do bad things too, but as my friend the NYU professor Clay Shirky (<http://shirky.com>) says, a sure sign of a good technology is that it's easy to think of bad ways to use it.

So two things give me hope that we can save the world and make it better. One is the Internet, and the other is you. Thanks for the privilege of speaking to you today, and congratulations on graduating from the best high school in the world. ■

Doc Searls is Senior Editor of *Linux Journal*. He is also a fellow with the Berkman Center for Internet and Society at Harvard University and the Center for Information Technology and Society at UC Santa Barbara.

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



Instant Access to Premium Online Drupal Training

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

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

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

