LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

FEBRUARY 2015 | ISSUE 250 | www.linuxjournal.com

WEB DEVELOPMENT

PLUS

Kickstart Development Easily with DevAssistant

SQL INJECTION
AND THE DRUPAGEDDON VULNERABILITY

USE JAVASCRIPT
FOR SERVER AND CLIENT PROGRAMMING

A LOOK AT DJANGO'S NEW AND INTERESTING FEATURES

Secure Cloud Deployment Tips

Every website needs a host.
Not every host is a good one.

#HostResponsibly

blackmesh.com
24/7/365 Managed Hosting

CONTENTS

FEBRUARY 2015
ISSUE 250

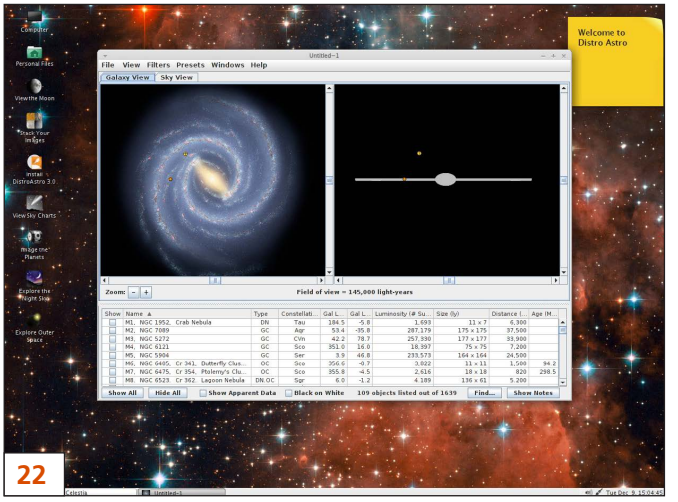
WEB DEVELOPMENT

FEATURES

ON THE COVER

- SQL Injection and the Drupageddon Vulnerability, p. 80
- Use JavaScript for Server and Client Programming, p. 62
- A Look at Django's New and Interesting Features, p. 32
- Secure Cloud Deployment Tips, p. 46
- Kickstart Development Easily with DevAssistant, p. 92

INDEPTH

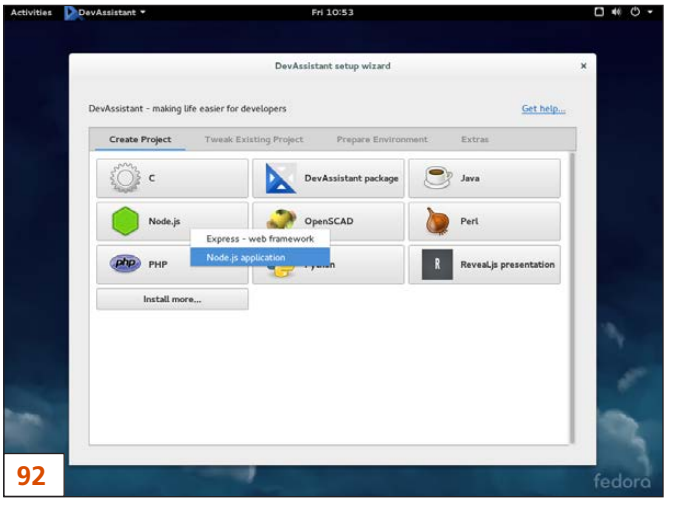


22

COLUMNS


50

IN EVERY ISSUE



92

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

LINUX JOURNAL

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

Contributing Editors

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

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

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

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

SILICON MECHANICS

SERVERS • STORAGE • HPC

Are you attending SCaLE 13x?
Email *SCaLE@siliconmechanics.com* to receive **40% off** your registration, and don't forget to stop by our booth for a chance to **win a Raspberry Pi!**

Are you tired of dealing with proprietary storage?

zStax®
ZFS Unified Storage
Powered by NexentaStor

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

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

zStax StorCore 64

The **zStax StorCore 64** utilizes the latest in dual-processor Intel® Xeon® platforms and fast SAS SSD for caching. The zStax StorCore 64 platform is perfect for:

- small-medium office filers
- streaming video hosts
- small archives

zStax StorCore 104

The **zStax StorCore 104** is the flagship of the zStax product line. With its highly available configurations and scalable architecture, the zStax StorCore 104 platform is ideal for:

- backend storage for virtualized environments
- mission critical database applications
- always available active archives

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

SHAWN POWERS

Under Construction

I think it was in the late 1990s, possibly into the 2000s, when it was common to put a cutesy graphic on the bottom of your Web page letting everyone know your site wasn't finished. Generally the graphic was an animated GIF file of a little construction guy shoveling a pile of gravel. Mind you, this was before animated GIF files were the most annoying thing on the Internet, and long before they started getting cool again. The thing that makes me smile isn't how clever we were to make such graphics, it's the naiveté of the concept that a Web site might ever be truly finished.

The Internet exists in a constant state of change. The most we can hope for is reaching milestones, but even those are far from static. Thankfully, with the wide range of development platforms available, the Web is evolving faster and better all the time. This issue focuses on Web development and how Linux plays such a vital role. Our resident developer Reuven M. Lerner starts things off with a fresh look at Django. Although it's been around for years, Django is a great Python framework. Reuven walks through setting up a basic project and writing a function.

Dave Taylor takes us back to the digital casino this month as we start a new command-line game. As with all Dave's projects, this one promises to be enjoyable, and if we're not careful, we might learn some Bash scripting along the way! Kyle Rankin moves into part two of his series on deploying servers in the cloud. Hosting your servers in someone else's data center can save time and money, but it's not without its security risks. Kyle describes mitigating those concerns as he continues his series.

For my column this month, I sort of take a left turn. Although I get a lot of e-mail concerning the topics I write

VIDEO:
Shawn Powers runs through the latest issue.

about, I most often am asked about my work environment and lifestyle. No, there's not a mob of paparazzi in my front lawn trying to get a glimpse of me in bunny slippers, but I do get a steady stream of e-mail. So in this issue, I try to answer the questions I get asked most often.

Back to the Web development theme, Federico Kereki delves into the world of JavaScript—yes, that JavaScript, but it's far more than it used to be. With the advent of Node.js and the like, JavaScript has become an entire development platform. Federico explores the development process and shows how to create an application. If you've heard of Angular.js and Node, but are still stuck in the world of LAMP applications, be sure to check out his article.

Unfortunately, along with great advances in Web development there are great advances in the security concerns of those Web sites. Shea Nangle explains the SQL injection vulnerability Drupal folks have been dealing with, along with an explanation of how best to deal with compromises. Most important, he also discusses how to avoid such vulnerabilities in the future. There's no magic bullet to secure every Web site, but with planning ahead and adopting a multipronged approach, hopefully we can avoid having the frequency of vulnerabilities increase as fast as the number of Web sites do!

Finally, Tomas Radej introduces us to DevAssistant. Although it's a powerful tool full of features any developer will appreciate, at its core it automates the part of coding that developers dislike—namely, the parts that aren't coding! Regardless of your language or framework, DevAssistant is designed to save time and make time spent writing code more productive.

Although there's an underlying assumption that every Web site is "under construction", that doesn't mean we never get anything done. It just means that with every advance and improvement we make, there's always room for newer and better. Whether you spend your free time writing code, or if your free time is the only time you don't write code, this issue of *Linux Journal* is full of helpful information. As someone who recently started to delve into the world of programming, this issue was extremely interesting for me. I hope you enjoy it as much as we enjoyed putting it together!■

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

letters

Cloud Computing Basics Article

Thank you very much for the article "Cloud Computing Basics—Platform as a Service (PaaS)" from Mitesh Soni in the February 2014 issue of *Linux Journal*.

I'm a PhD student, and my work is about security models for PaaS, so this text was very useful reading.
—Jose Luis Faria

Ideal Backups with zbackup—Data Integrity?

I very much enjoyed David Barton's article "Ideal Backups with zbackup" in the November 2014 issue.

One of the main concerns I have (which is mentioned in the article) is how to mitigate data corruption to which the deduplicated data store of a zbackup is particularly vulnerable.

The author hints that the filesystem snapshots can be used to help but is not really explicit as to his methods. I was wondering if he would be willing provide some more detail on that point.

I could envision snapshotting the backup volume every once in a while to basically do a "full" backup, but these snapshots would have to be replicated to other machines for it to really be worthwhile. Is this what the author was hinting at, or is there another technique I'm missing here?

Another partial solution, as the author suggests, might be to `rsync --ignore-existing` (and also maybe `-c`) the zbackup data store when replicating it offsite. This would prevent a corrupted store from being blindly over written across offsite media. Combined with regular restoration tests, this might be sufficient for some workflows.

Any further insight that I might be missing would be much appreciated.
—Chris Wills

Dr Hjkl Feedback

As a Vim fanatic myself, I was excited to see Kyle Rankin's article Dr Hjkl in the December 2014 issue. But I was shocked after reading it. All the command-line shortcuts you listed are actually bash/readline *Emacs* shortcuts (*gasp*)! Try adding the following to ~/.inputrc:

```
set editing-mode vi
```

Now a lot of your Vim favorites—^, $, w, f and so on—work verbatim! So all the Vim muscle memory you've acquired is directly applicable in Bash.

What's more, .inputrc is actually a config file for the GNU readline library, not Bash. Bash is only one of the many consumers of readline—you'll have the same vi keybindings in gdb, pdb, database shells and may other CLI applications.

The only caveat is that Bash won't show you when you're in insert vs. normal mode like Vim will. I tend to press Esc (or Ctrl-[) when in doubt.

Also, if you happen to be on OS X, be aware that some BSD applications use the editline library instead of readline, so you'll want to `echo "bind -v" >> ~/.editrc`.

Thanks to you and all the *LJ* team for all the great articles!
—Chris

Regarding `find`, `xargs` and Spaces

In Dave Taylor's article "Power Shell History and the find Command" in the December 2014 issue, he wrote: "Next time, I'll talk about the `find|xargs` command pipe pair and the substantial problem with files and directories that contain spaces."

There is another, in my view even more severe, problem with typical uses of "find | xargs". The `xargs` command (except in the `xargs -0` case) parses for matching single and double quotes. So if an incoming filename contains an odd number of such characters, the `xargs` command will fail.

For example, in an empty test directory,

execute the following commands:

```
touch a 'b"c' d
find . -print | xargs ls -ltd
```

The result will be the following error and failure with exit by `xargs`:

```
xargs: unmatched double quote; by default quotes are \
special to xargs unless you use the -0 option
```

Both the space issue, and this quote issue, make the typical use of this otherwise quite useful pipe *unacceptable* in production shell scripts, in my view, and likely present a serious security risk, if a root-enabled script uses "find | xargs" over a portion of the filesystem space in which non-root tasks can create files (almost anywhere).

Sometime ago, I wrote a small wrapper for `xargs` that has served me well, and that always invokes `xargs` with the `-0` (minus zero, for nul separator) argument, the *only* safe and reliable form of `xargs`.

In my view, the initial `xargs` command-line interface options and design (dating back to at least Western Electric's UNIX System V, January 1983, the oldest UNIX manual I still possess) are irreparably flawed and should *never* be used in production code. The only safe `xargs` option, `-0`, was added later in the history of `xargs`, and it is not in the System V variant.

You can find a copy of my "x.c" program at **pauljackson.us/x.c** available under the GPLv2 license. It might be worth recommending to your readers.

The `x` command compiled from "x.c" is used exactly like `xargs`, except that the incoming filename stream from find must be newline-separated, the natural `find` default. Instead of saying

```
find ... | xargs ...
```

rather say:

```
find ... | x ...
```

So, for example, the test case above that failed using `xargs` works fine using `x`.

My `x` command does still have the same limitations as the underlying `xargs` command regarding multibyte language locales for filenames, plus the problem with not correctly handling the nul to newline conversion. It is useful only in single-byte ASCII-like

locales, so far as I know.
—Paul Jackson

Dave Taylor replies: *Thanks for your thoughtful note. I am always aware of the lurking horror of weird characters in filenames, but there's a balance because part of what makes the shell a great learning environment for scripting and programming is that it's simple and straightforward. Many shell scripts could be more robust and bulletproof if written as C programs or Perl scripts, and in some instances, that's critical. In other instances, however, simplicity and an understanding of how Linux works, and the UNIX underpinnings, is more important.*

And really, how often do you find filenames in your Linux system that have a single or double quote as part of the filename?

Paul Jackson replies: I have lots of such filenames on my Linux disks, as they include some Windows filesystems, which run inside VirtualBox emulators.

But the more important fact is that `xargs` in any form, except the `xargs -0` form (minus zero, for nul-separated) is dangerous in any root-enabled script, if fed paths from a `find` command over any portion of a filesystem that non-root processes can write, as they can construct, deliberately, filenames that will be incorrectly parsed by `xargs`.

Readers' Choice Awards Request

As I read through the Readers' Choice Awards 2014 (in the December 2014 issue), I was looking for and hoping to see a category for best IDE, since many, or most readers probably also are programmers. Would you consider Best IDE as a category in the future?
—Joe Farkas

Absolutely. In our attempt to streamline the categories, "Best IDE" ended up on the cutting-room floor. If it's something folks still are interested in, we can certainly add it back next year. Thanks for the feedback, and if you think there are other categories we should be including, please send e-mail to ljeditor@linuxjournal.com.
—Shawn Powers

Linux Journal App on Kindle Fire HDX 8.9" Tablet

I found how to increase the text size of the articles in the *Linux Journal* app on my Kindle Fire HDX 8.9" tablet, but how can I increase the text size of the table of contents?
—Mike

I don't actually have a Kindle Fire, but if the font size doesn't increase on the table of contents using the same method to increase the font size elsewhere, maybe it's not supported in the device. I think it would be worth contacting Amazon's tech support to find out.—Shawn Powers

Index to Past Articles?

Your back issues contain a vast wealth of information, much of which is still relevant. Do you ever publish an index to past articles?
—John

The annual DVD (see lj.mybigcommerce.com/linux-journal-archive-dvd-1994-2014*) contains every issue ever published, and as such is fairly popular for that very reason. It's a convenient way to read the archived information as well, since looking at back issues isn't quite the same as browsing through a new one.—Shawn Powers*

WRITE *LJ* A LETTER

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

PHOTO OF THE MONTH

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

LINUX JOURNAL

At Your Service

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

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

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

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

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

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

Big Data Gets Real in Boston!

People are talking about BigData TechCon!

"Big Data TechCon is a great learning experience and very intensive."
—Huaxia Rui, Assistant Professor, University of Rochester

"Get some sleep beforehand, and divide and conquer the packed schedule with colleagues."
—Paul Reed, Technology Strategy & Innovation, FIS

"Worthwhile, technical, and a breath of fresh air."
—Julian Gottesman, CIO, DRA Imaging

"Big Data TechCon is definitely worth the investment."
—Sunil Epari, Solutions Architect, Epari Inc.

BigData TECHCON

April 26-28, 2015

Seaport World Trade Center Hotel

Choose from 55+ classes and tutorials!

Big Data TechCon is the HOW-TO technical conference for professionals implementing Big Data solutions at their company

Come to Big Data TechCon to learn the best ways to:

- Process and analyze the real-time data pouring into your organization
- Learn how to extract better data analytics and predictive analysis to produce the kind of actionable information and reports your organization needs.
- Come up to speed on the latest Big Data technologies like Yarn, Hadoop, Apache Spark and Cascading
- Understand HOW to leverage Big Data to help your organization today

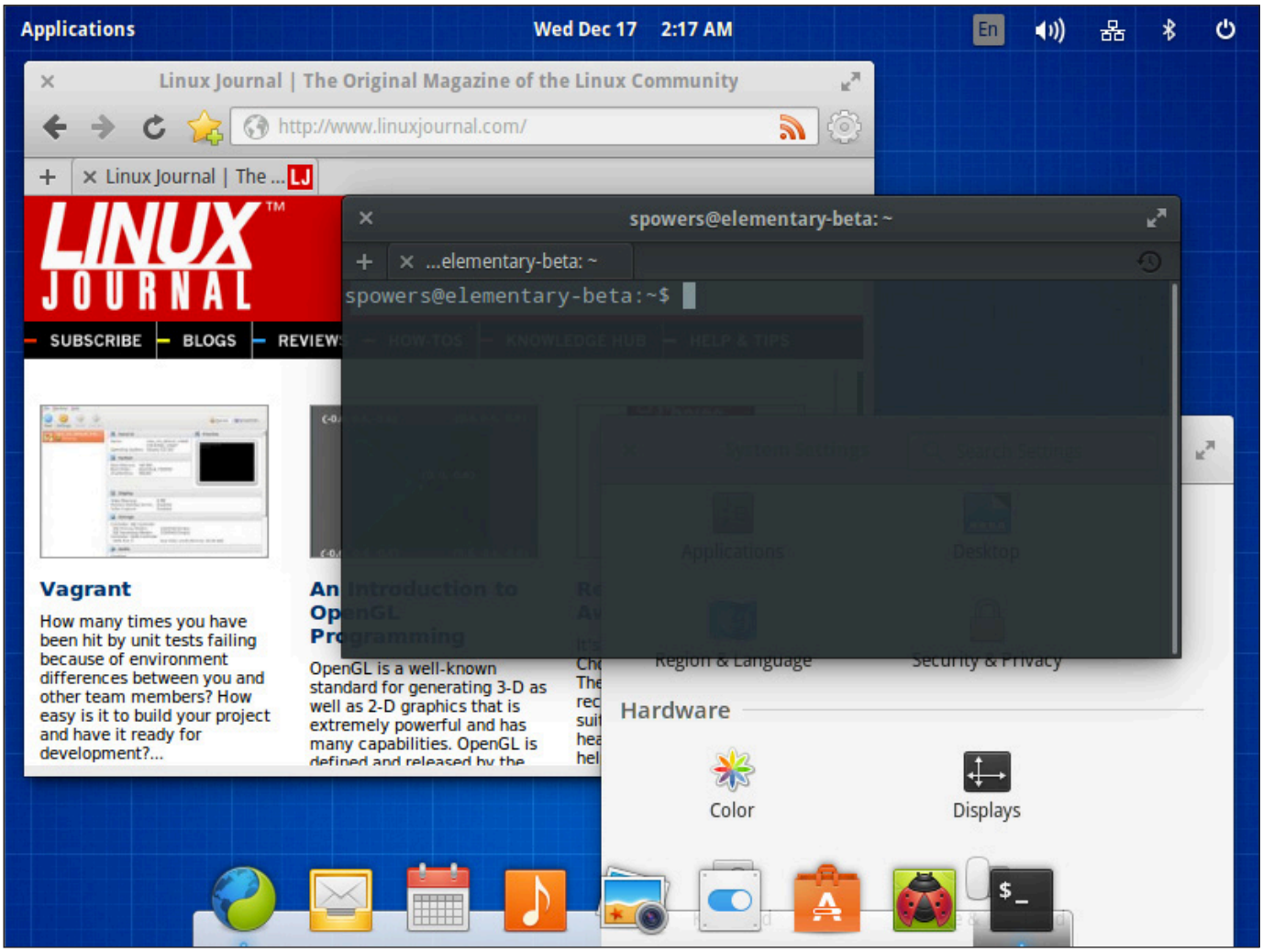
www.BigDataTechCon.com

A BZ Media Event

Elementary, My Dear Linux User

I suspect there are as many Ubuntu-based Linux distributions as there are all other distributions combined. Many of them are designed with a specific purpose in mind. Whether the desire is for a different looking desktop, custom kernel or just pre-installed packages, there's probably a version of *buntu out there to fit every need. Elementary OS is just another in a long list of variants, but what it does, it does very well.

Upon first boot, Elementary



obviously is designed to look and function like OS X. Although that might turn the stomachs of die-hard Linux fans, it provides an interesting platform for folks who appreciate the clean, functional layout Apple provides on its flagship OS. Elementary OS includes the Ubuntu Software Center, and like most variants, it can install any program in the Ubuntu repositories. Out of the box, however, it's a clean, fast operating system that people familiar with OS X will recognize right away.

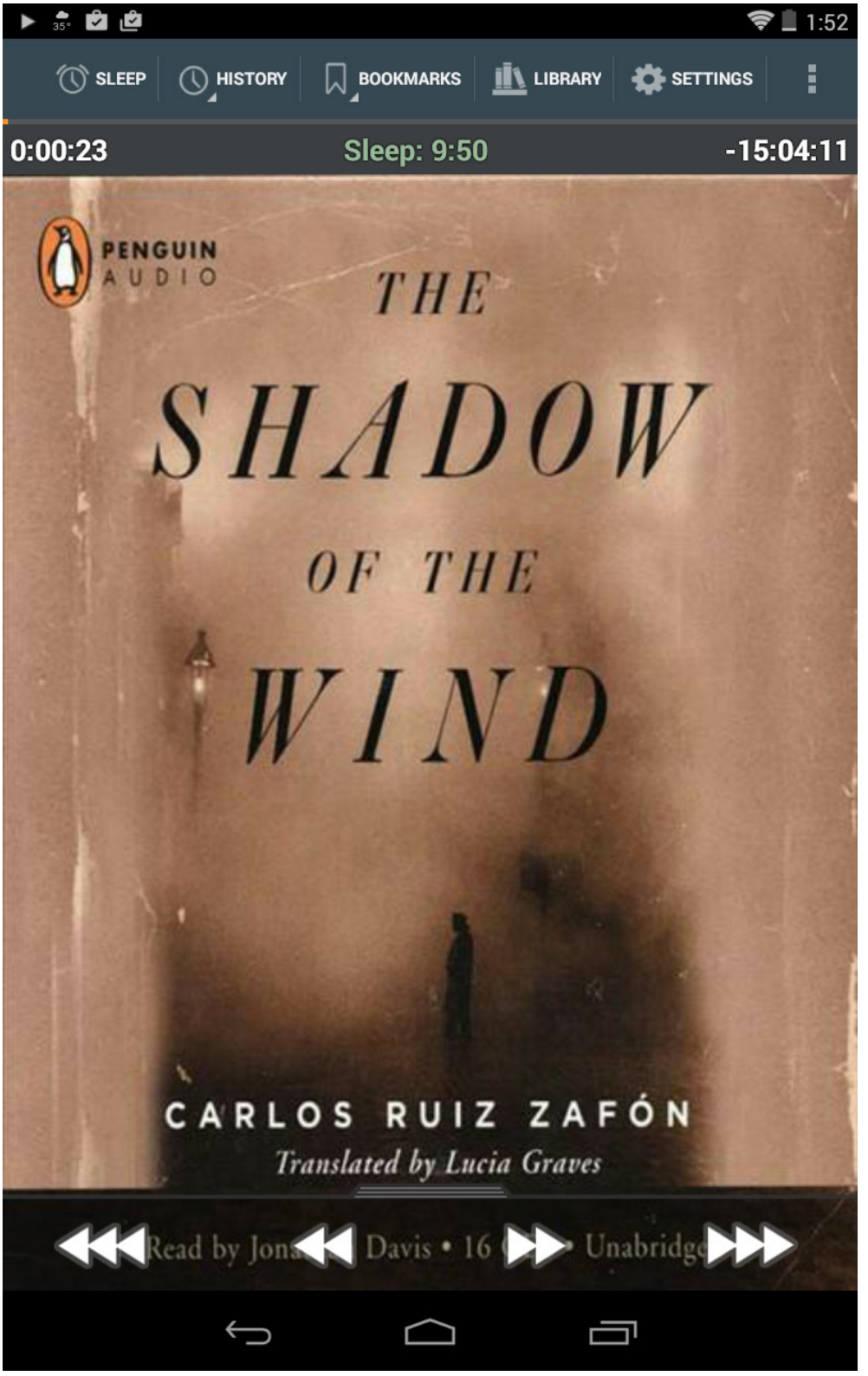
Elementary OS doesn't do anything particularly special compared to other Linux distributions, but when I tried it, I really liked it. The graphics are beautiful, and the default applications are lightning fast. If you just want a simple, functional desktop, I urge you to give it a try. At the time of this writing, the version based on Ubuntu 14.04 still is in beta, but the stable version is available today and looks very similar. Give it a spin at http://elementaryos.org.

—SHAWN POWERS

Android Candy: Listen!

I don't listen to music very often. It's not that I don't like music, it's just that I love books. If I have an opportunity to listen to entertainment, I almost always go for an audiobook. For several years, I've been using Smart AudioBook Player on my Android devices. I've written about it before, and although it's still incredible, I have a new favorite—namely, Listen Audiobook Player.

Along with the normal bookmark features you'd expect from an audiobook



program, Listen provides:

- Multi-device playback position sync via Dropbox.
- Support for m4b, m4a, mp3, opus and ogg files (DRM-free).
- Cover art downloading.
- Variable speed playback with global default and individual book settings.
- Automatic sleep timer with audio fade.
- Bluetooth control support (play, pause, skip).
- Widgets.
- Elaborate appearance customization (colors, buttons and so on).

Although Listen isn't free, the $0.99 cost is more than fair for the smooth-functioning, feature-rich application. When combined with FolderSync to upload audiobook files automatically, I'm hard-pressed to think of a better way to experience audiobooks. Since the author has a seven-day refund policy, there's no reason not to give it a try (http://snar.co/listen).

—SHAWN POWERS

They Said It

People fail forward to success.
—Mary Kay Ash

A finished person is a boring person.
—Anna Quindlen

The important thing is this: To be able at any moment to sacrifice what we are for what we could become.
—Charles DuBois

A mind, like a home, is furnished by its owner, so if one's life is cold and bare he can blame none but himself.
—Louis L'Amour

Men are born with two eyes, but only one tongue, in order that they should see twice as much as they say.
—Charles Caleb Colton

Non-Linux FOSS: Homebrew

```
nermal:~ spowers$ brew install wget
==> Downloading https://downloads.sf.net/project/machomebrew/Bottle
Already downloaded: /Library/Caches/Homebrew/wget-1.16.1.mavericks.
bottle.tar.gz
==> Pouring wget-1.16.1.mavericks.bottle.tar.gz
🍺  /usr/local/Cellar/wget/1.16.1: 9 files, 940K
nermal:~ spowers$ wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
nermal:~ spowers$
```

I use OS X quite often during my day job. I'm able to tolerate it largely due to the terminal. If I couldn't do my work with green text on a black background, I think I'd go crazy (or crazier). Unfortunately, OS X doesn't come with all the command-line tools I need. That's where Homebrew comes in to save the day.

Homebrew acts like the package management system OS X is lacking. Using commands very similar to apt-get, it allows the installation of hundreds of applications. A perfect example is the wget program. I was surprised to find that OS X doesn't include wget, but with Homebrew, it's a simple one-liner away.

The best part is that Homebrew installs everything in the /usr/local file space. There's no reason to worry about Homebrew corrupting your system, because it doesn't touch anything outside of /usr/local. OS X system updates won't overwrite your programs, and because /usr/local/bin is already in the PATH, installed Homebrew apps just work!

Homebrew uses Ruby to manage its packages and functions, but it doesn't require any programming knowledge to use. And the installation procedure is literally a copy/paste on the command line. If you use OS X, but you wish you could install packages as easily as in Linux, give Homebrew a try: **http://brew.sh**.

—SHAWN POWERS

Linux for Astronomers

I've looked at specialty distributions that were created for engineers and biologists in previous articles, but these aren't the only scientific disciplines that have their own distributions. So in this article, I introduce a distribution created specifically for astronomers, called Distro Astro (**http://www.distroastro.org**). This distribution bundles together astronomy software to help users with tasks like running observatories or planetariums, doing professional research or outreach.

From the very first moment of booting up Distro Astro, you will notice that this distribution is aimed

Figure 1. Even the boot splash screen has an astronomy theme.

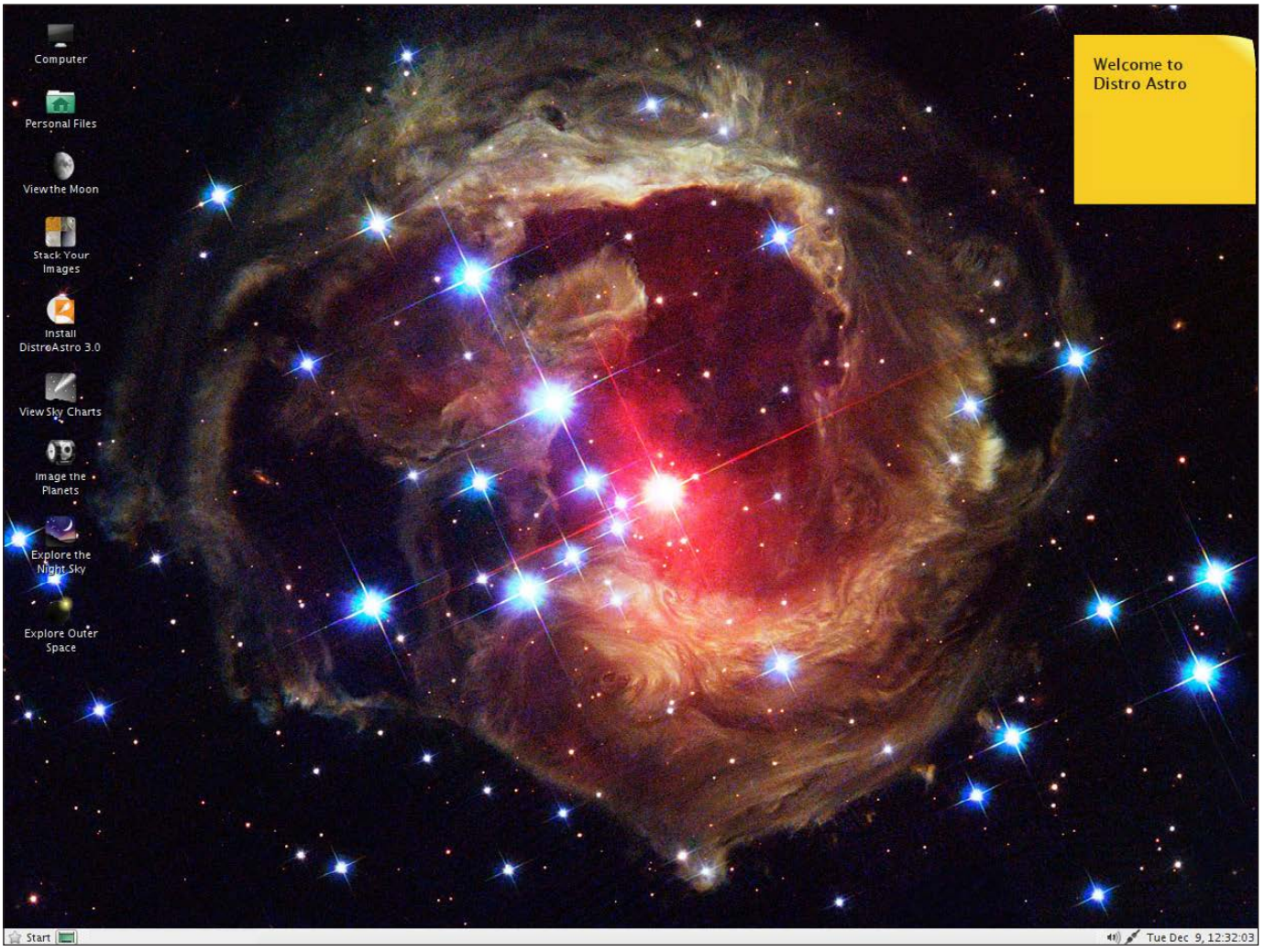


Figure 2. Once you boot up, you have desktop shortcuts pointing to a series of different tasks.

at astronomers. The look and feel of items, from the boot splash screen to wallpapers and screensavers, have all been given an astronomical theme. Even the default wallpaper is a slideshow of Hubble images.

There is also a unique feature called Nightvision Mode that lets you switch the color theme to red-based colors to help maintain your night vision when using the computer during observations. Clicking on the menu item will toggle this mode on or off.

The advantage of having a domain-specific distribution is that all of the software you likely will use is bundled together and installed. Ideally, you shouldn't have to install anything other than the distribution itself.

The first task in astronomy is to

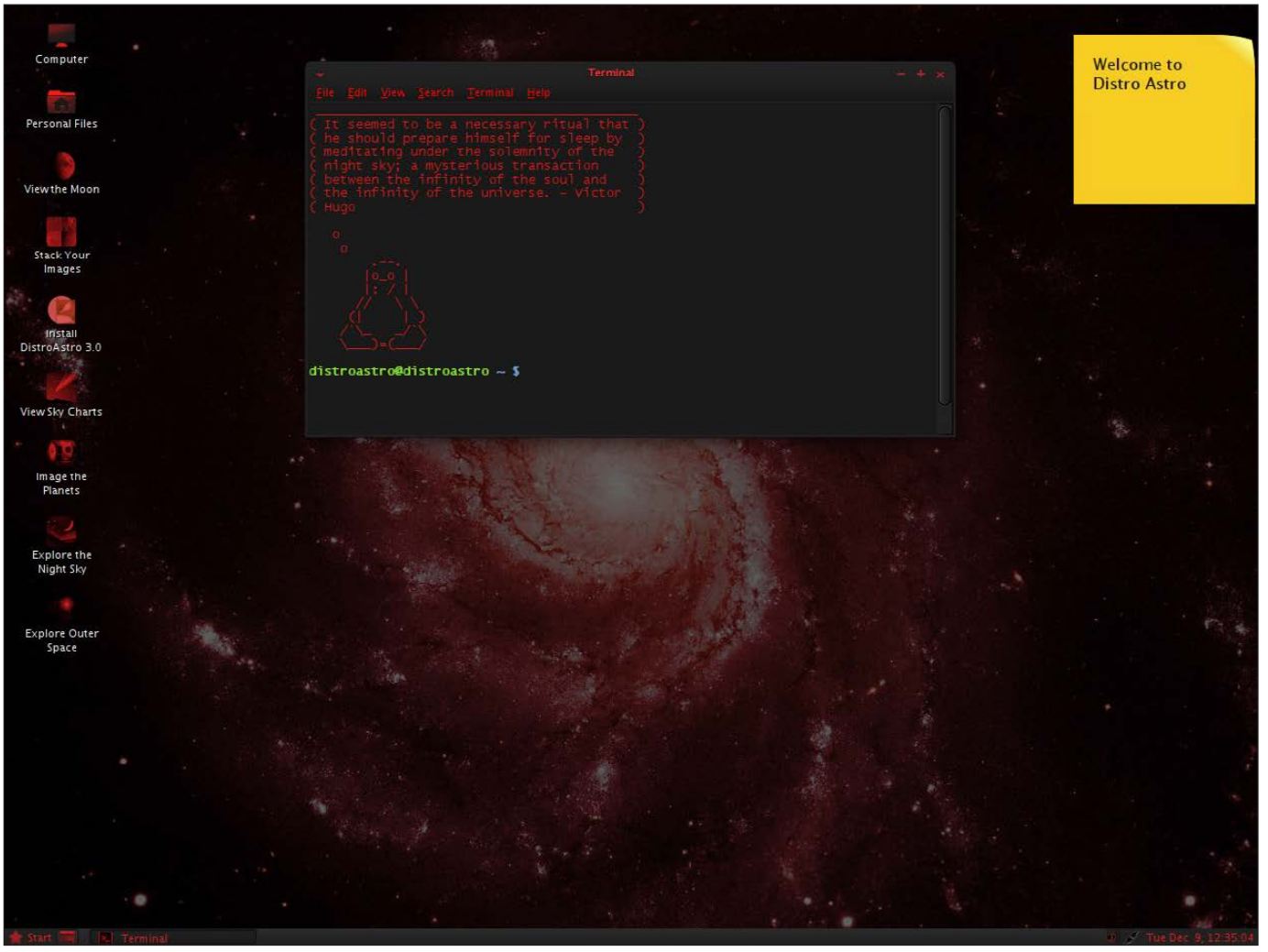


Figure 3. For night observations, you can turn on Nightvision Mode.

collect images of the sky. In Distro Astro, the INDI library is pre-installed for you. This library is the main interface that is used in telescope control. With it, you can control telescopes from Meade, Celestron and Orion, among others. If you are lucky enough to have a domed observatory, you even can control commercial domes, such as those from Sirius Observatories.

To be able to make your observations, you need a front end to talk to your hardware. This access is provided through clients like KStars, XEphem and Cartes du Ciel, which use the INDI library to talk to your telescope.

After aiming your telescope, you need to collect some images or do some astrophotography. While you can do some of this with software like KStars, you have software specifically designed to do image

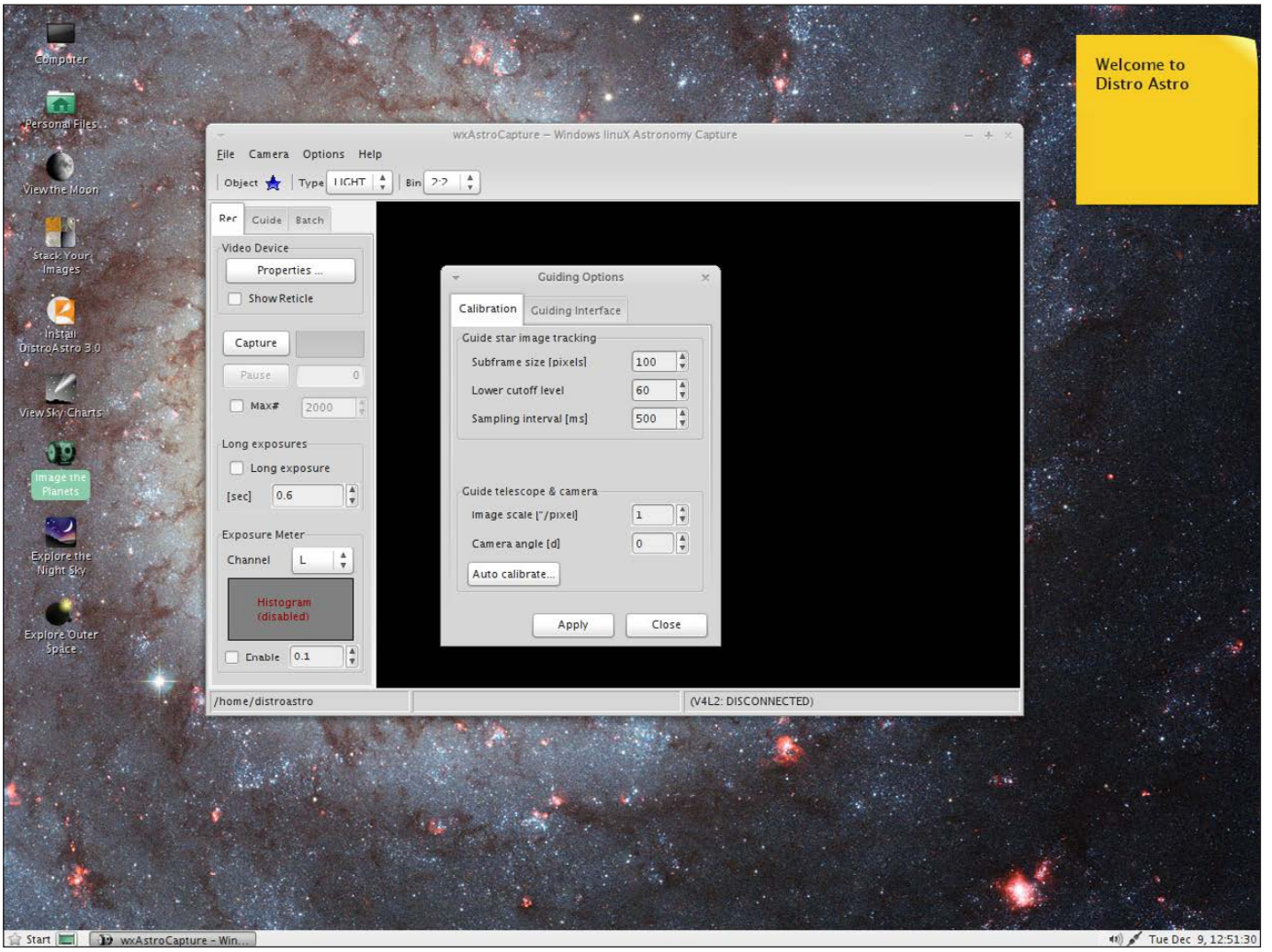


Figure 4. You can do image capture from your telescope with wxAstroCapture.

capture. Some, like wxAstroCapture, are specifically written for use in astronomy. With it, you can set up automatic guiding and batch image collection. You then can go have a nice hot cup of coffee while your telescope collects your data. To help you keep track of all of these observations, you can use the Observation Manager, a logging program to maintain your records.

Once you have made your observations, you need to process them and do some science with them. The first step is to do image processing. This might be some filtering or some sort of feature identification routine. One common task is image stacking, so much so that Distro Astro includes the RegiStax package. With it, you can set alignment points and do image analysis on full stacks of images. You even can set up batch processes

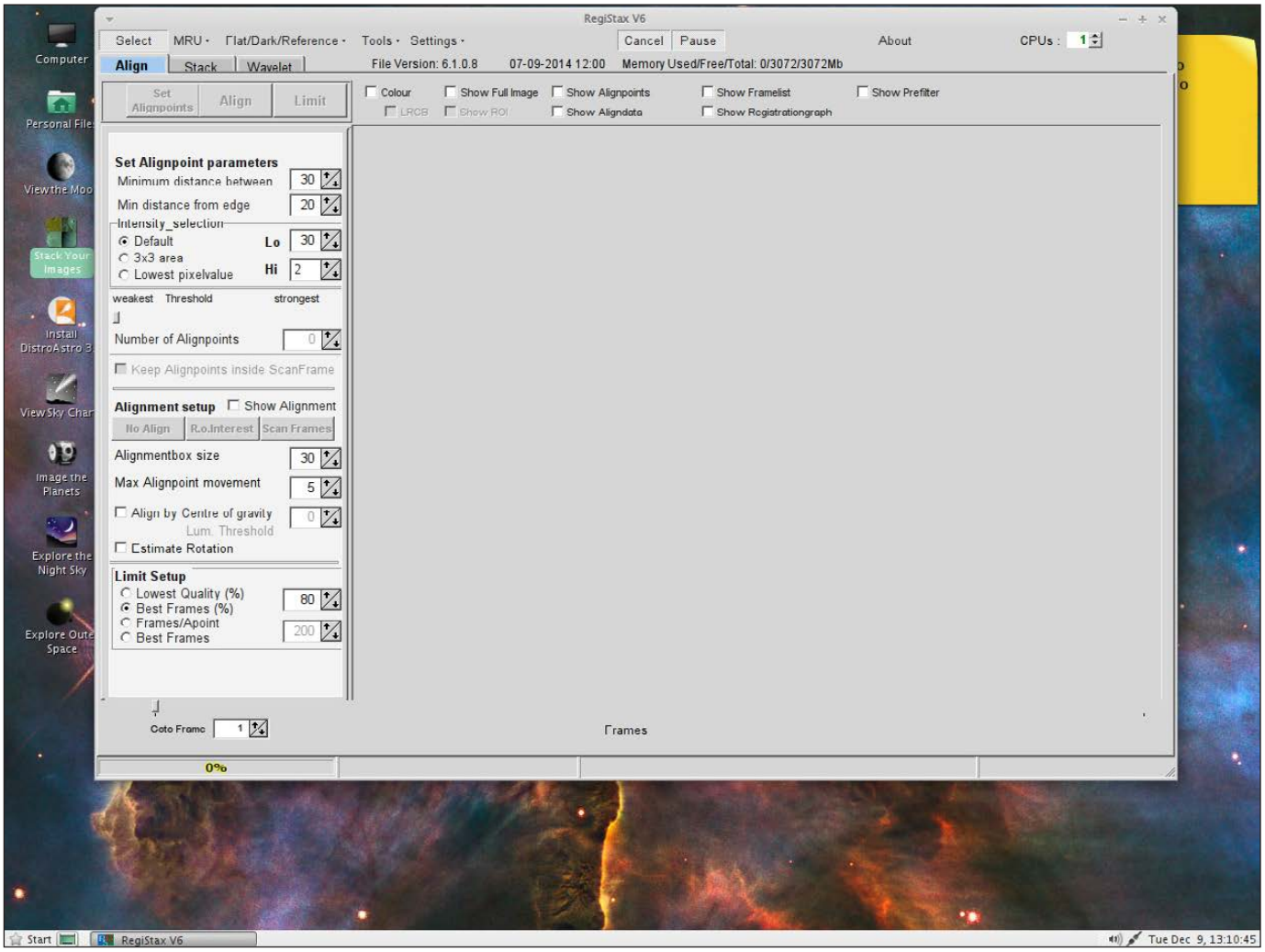


Figure 5. With RegiStax, you can do image processing on full stacks of images.

and run on multiple CPUs, if you have them.

You can do more complex image analysis with tools provided by IRAF (Image Reduction and Analysis Facility). Sometimes, however, the functions already available aren't enough, especially when you are doing leading-edge research. In those cases, you are forced to write your own algorithms to do this work. ImageJ is a very popular choice that is available in Distro Astro. With it, you can develop and write your own algorithms in Java.

Unfortunately, the weather doesn't always allow for observations. So, what can you do if you have been stuck away from the telescope for days on end? Distro Astro includes a number of sky chart programs to allow you to plan out your future observations. They also are really useful for budding

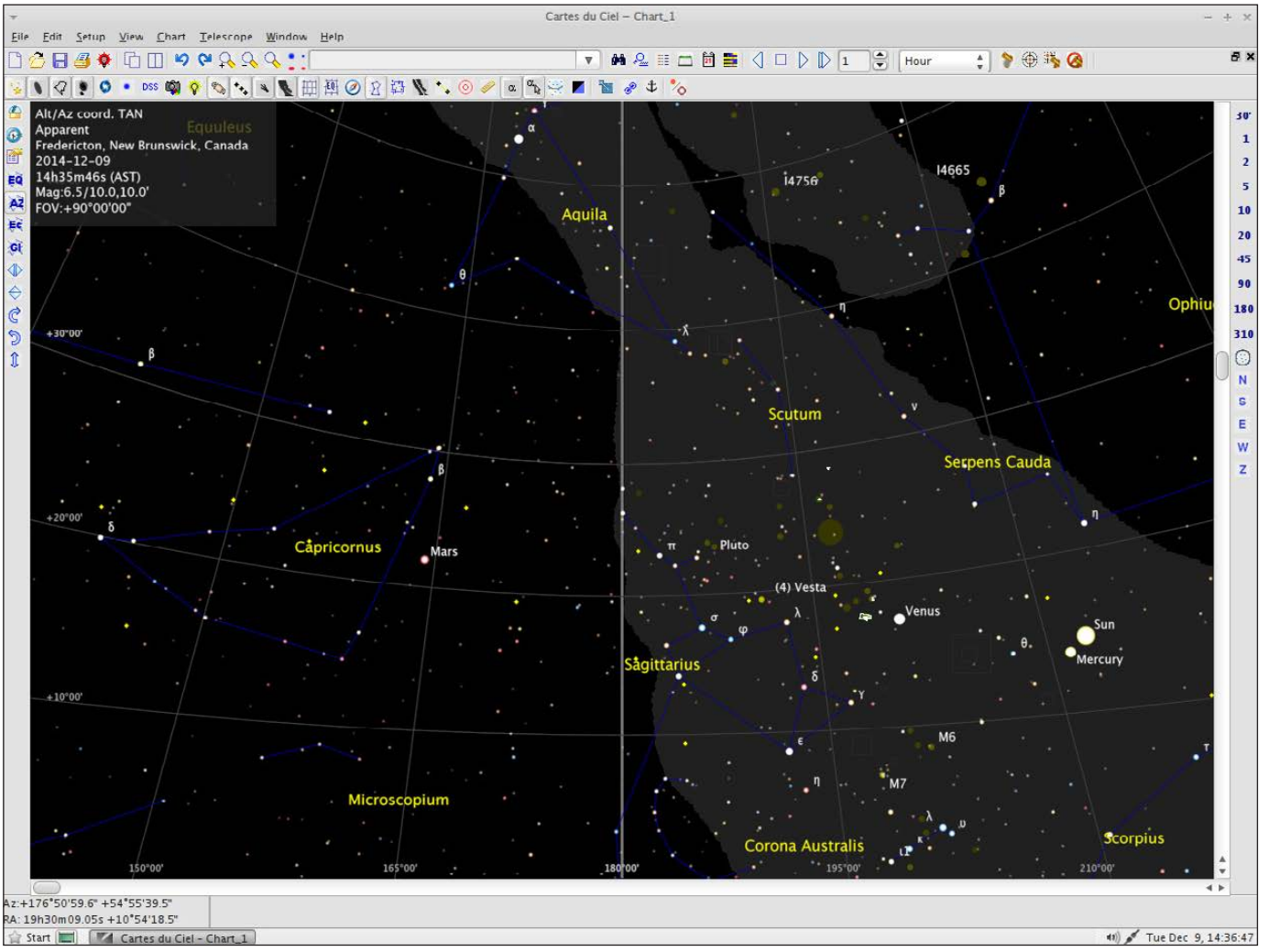


Figure 6. With Carte du Ciel, you can learn more about the sky.

astronomers. You can use packages like Cartes du Ciel to learn about the sky above your head. And there are several other options, including Stellarium and Celestia. If you are more interested in deep sky objects, there is a package called Where is M13?. This package shows objects from several catalogs, such as the Messier, Caldwell, Collinder or NGC catalogs. The unique feature of this software is that it can locate these deep sky objects in three dimensions. That way, you can start to get a feel for where in space these objects actually are.

Part of science is outreach, sharing your findings with others and inspiring them. One way of doing this is by putting on a planetarium display. This is a way to expose a large number of people to the wonders of the sky all at once. Several different software

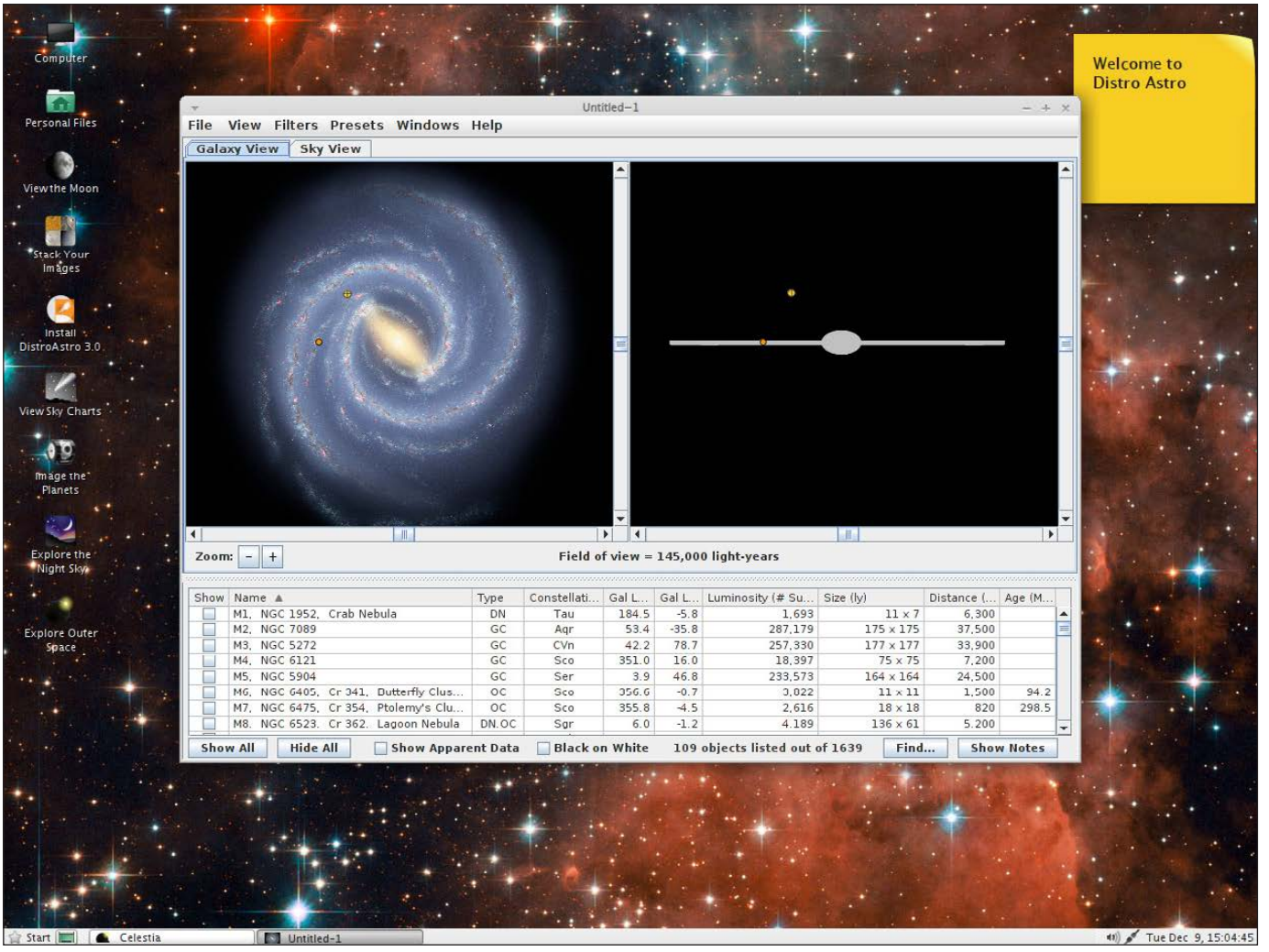


Figure 7. Where is M13? lets you see where deep sky objects are located around you in three dimensions.

packages are included in Distro Astro that can talk to planetarium projectors. Packages like Celestia and OpenUniverse can be used with non-fisheye projectors. If you have a more professional fisheye projector, there is a modified fork of Stellarium called Nightshade Legacy that is included in Distro Astro. The original version of Stellarium also is included, giving you some choice in what is available.

With Distro Astro, you now easily can set up that observatory computer or configure a machine to do your research runs on. And, with a live CD option available, you have no excuse to keep you from downloading an ISO and giving it a try. And with all of the packages included, you should be able to burn a CD and run with nothing else to install.—**JOEY BERNARD**

Many Drives, One Folder

RAID is awesome, and LVM is incredibly powerful, but they add a layer of complexity to the underlying hard drives. Yes, that complexity comes with many benefits, but if you just want to

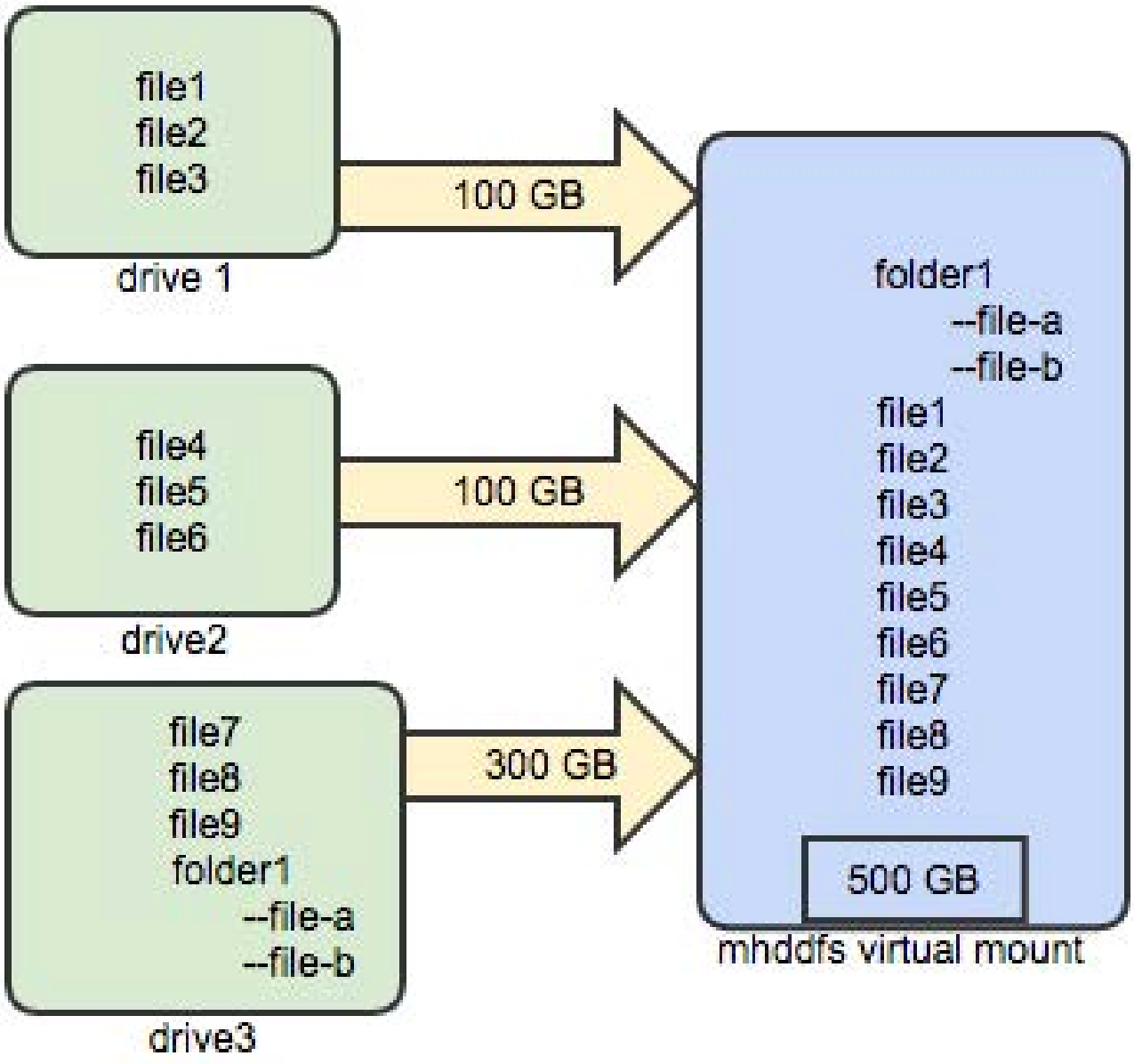


Figure 1. The System in Action

```
spowers@docboy:/mnt$ mhddfs /mnt/serv1,/mnt/serv2,/mnt/serv3 /mnt/everything/
mhddfs: directory '/mnt/serv1' added to list
mhddfs: directory '/mnt/serv2' added to list
mhddfs: directory '/mnt/serv3' added to list
mhddfs: mount to: /mnt/everything/
mhddfs: move size limit 4294967296 bytes
spowers@docboy:/mnt$ df -h
Filesystem                      Size  Used Avail Use% Mounted on
/dev/sda1                       111G  4.3G  101G   5% /
192.168.1.10:/serv2             17T   15T   1.6T  91% /mnt/serv2
192.168.1.8:/serv1              5.4T  2.0T  3.5T  36% /mnt/serv1
192.168.1.6:/serv3              1.9T  1000G 863G  54% /mnt/serv3
/mnt/serv1;/mnt/serv2;/mnt/serv3 24T   18T   5.9T  76% /mnt/everything
spowers@docboy:/mnt$
```

Figure 2. Using mhddfs

spread your files across multiple storage locations, there's a much easier way.

The mhddfs program is a userspace application that creates a virtual mountpoint. Using standard drives (USB, network, RAID, whatever), it concatenates the storage into a single volume. When you write files to the virtual volume, it just saves them to the underlying drives until one of them fills up, then it moves to the next. Looking at the virtual mountpoint, you can't tell where one drive ends and the next starts, but because mhddfs writes directly to the underlying drives, you easily can look at them (and even remove them!) to see the individual file locations.

There's no RAID involved, there's no striping or mirroring, there's simply a single virtual mountpoint that shows the contents of all your drives in a single folder. It's a simple concept, but incredibly convenient if you need to have a single view of all your data spread across multiple drives. (See Figure 1 for a visual representation of the system in action, and Figure 2 to see it in use.)

Although mhddfs might be simple, it's incredibly useful. In fact, we found it so ingenious, we gave it this month's Editors' Choice award. You probably can install it from your distribution's repositories, but if not, visit the home page at **http://mhddfs.uvw.ru**. (The page is in Russian, but an English version is available.)**—SHAWN POWERS**

REUVEN M. LERNER

Django

Have you looked at Django lately? The leading Python Web framework has many new and interesting features for old and new users alike.

So, you want to write a Web application? Well, the good news is that you aren't lacking for languages or frameworks. Nearly every modern language supports a number of different frameworks, allowing you to choose from one that best matches your development needs and style. If you are a Python user, chances are you have heard of Django, a project launched in 2005 that often is compared with Ruby on Rails.

Django is not the only Python Web framework. Besides Flask, a microframework that I have written about here in the past, there are several other ways you can use Python to create a Web application. However, there's no doubt that Django is the best-known framework, as well as the most popular one, in the Python world. It is used to power large sites (such as Pinterest and Instagram) as well as numerous small and internal ones. Just as important, the Django community has demonstrated that it is both active and strong, moving development from a BDFL (benevolent dictator for life) model to the Django Software Foundation, a nonprofit entrusted with organizing, funding and promoting Django development.

Django often is compared with Ruby on Rails, and for good reason. Indeed, anyone who knows both Ruby and Python, and who knows Rails well, should be able to start getting work done in Django within less than a day. At the same time, I would say that Django adopts and promotes Python conventions and attitudes wholeheartedly. This contrasts with Rails, whose developers often have attitudes and conventions that go against what the Ruby language has promoted.

If you have a firm understanding of Python's modules and packages, much of Django will make perfect sense to you and will allow you to make use of that knowledge.

So in this article, I begin to cover Django and how you can use it in your Web projects. I plan to look at various

parts of Django during the coming months, including many of the recent updates to this framework that make it particularly flexible and powerful, as well as a generally good choice for many organizations—especially those that already are using Python.

Starting with Django

Most people, when they want to develop a Web application, think of it as just that—an "application". (Or, if I want to pretend to be cooler than I actually am, I could just call it a "Web app".) In Django, however, you think in terms of a "project". Each Django project, however, contains a number of applications, and each of them has its own Python code, as well as its own business logic and templates for displaying data.

This division of a Django project into separate applications took me some time to get used to. However, I think that it's a great way to operate and makes it possible to leverage existing code in new Django projects easily. It also forces me, as a developer, to break my code into small parts, which increases the chances that I'll be able to re-use that code in future projects.

To create a new Django project, you first must download and install this package from PyPi, the Python Package Index. The easiest way to do so is to use the `pip` command, which I'm delighted to say has been incorporated into the latest (2.7.9 and 3.4) versions of Python. Thus, you can type:

```
pip install django
```

and ensure that Django is installed. By default, Django will use SQLite3, which is installed on the computer by default, and for which libraries come along with Python. (Depending on your computer's configuration, you might need to execute the `pip` command as root, using `sudo`.) As I write this, the latest stable version of Django is 1.7.1, which includes a number of new and interesting features.

Once you have installed Django, you need to start a project. The installation of Django creates, among other things, a command on your computer called django-admin.py, which can be used to administer various parts of a Django project. For example, to create a new project, you can say:

```
django-admin.py startproject atfproject
```

Assuming that nothing named "atfproject" already exists in the current directory, this will create a new directory named atfproject, which will serve as your Django project. If you're

using Git, you immediately should enter into this directory, initialize a new repository and then commit the files. This will ensure that even if you mess something up, you'll always be able to return to the initial, stable state that Django gave you out of the box.

If you enter the atfproject directory, you'll see that it contains two different things. First, it contains a manage.py program, which you will use (not surprisingly) to manage your Django project. Second, it contains a subdirectory with the same name as the project's main directory (atfproject), which contains the project's main configuration and code. I find it a bit confusing that there are two directories, one within the next, named the same thing, but I have gotten used to it over time.

The inner project directory is a Python package, complete with an __init__.py file, as well as three other files: settings.py, urls.py and views.py.

The first, settings.py, contains the overall settings for the Django project. This file defines, for example, whether you want to be in debugging mode (`DEBUG = True`), what database driver you want to use (`DATABASES`) and what applications should be added to your project (`INSTALLED_APPS`). You already can see, if you look at settings.py, that `INSTALLED_APPS` is a tuple of strings, in which each string describes a Python package containing a Django app.

The second file, views.py, contains the definitions of view functions. I'll get back to those in a moment, when I describe creating your first application and then writing some view functions within it.

The third file, urls.py, provides Django with a mapping between URLs and the view functions. It allows you to say that if a request comes in for the URL /abc, you'll want to execute function "abc" as a result. (You'll soon use this URL configuration.)

You might think, given the number of times I have written that "I'll discuss" a certain topic, that there's a great deal more configuration to do in order to run your Django project. But that's actually not the case. Even without any code, you already can start the Django application running on your computer. Inside the top-level project directory, invoke:

```
./manage.py runserver
```

As the name suggests, this starts a simple HTTP server and then connects your Django application to that server. By default, the server listens on port 8000. This means that by pointing your Web browser to http://localhost:8000,

you will see the home page of your new Django project. Not surprisingly, it doesn't say or do much other than tell you that you have successfully installed and run a new project.

Creating an Application

Let's add an application to the Django project. This also will give you a chance to create a simple view function, as well as a URL path that allows people to invoke your function.

The first step in creating an application is to use the manage.py program in the top-level directory of your Django project. Simply invoke:

```
./manage.py startapp atfapp
```

And, you'll soon have a new Django application (atfapp) set up inside your project. What does a Django application contain? Well, if you look inside of the atfapp directory, you'll find a Python package. The directory contains an empty __init__.py file, but then contains four other files: admin.py, models.py, tests.py and views.py.

For now, let's concentrate on views.py, which is where you can define the "view functions" for your application. Django calls itself an MTV framework, in which MTV stands for model, template and view. This is slightly different from the MVC (model-view-controller) model of application design used by Rails and several other frameworks. In MVC, the controller receives the request, communicates with models that represent the business logic and then displays things using views.

In Django, things work a bit differently. The HTTP request is directed to a view function by the urls.py file in the Django project's package directory. Based on the URL object, a view function then is invoked. This view function then communicates with one or more models and then renders a template. If you're coming from an MVC framework like Rails, there's a clear overlap (but still a distinction) between MVC controllers and MTV view functions.

Let's look at how one of these might work. Open up views.py in the application's directory (that is, atfapp/views.py). Add the following to it:

```
from django.http import HttpResponse

def hello(request):
return HttpResponse("hello, world")
```

Let's start with the function. Every view function takes at least one parameter, traditionally called "request". This request object represents the HTTP request that was

submitted to the HTTP server. If and when you want to inspect or display elements of the request, you can do so, but for now let's ignore it.

In fact, let's ignore any potential inputs and just return a basic, simple HTTP response. You do this by returning not a string, but an HttpResponse object, which you create by passing the class a string ("hello, world"). Of course, you also need to make sure that HttpResponse is defined, so you use Python's "from - import" statement to bring in the name from the django.http package.

Once this function is defined in views.py, you're just about set to go. But before it can be invoked via the Web, you need to tell Django what URL will reach it. For this, you must return to the urls.py file in the project's package directory and tell Django that you want to have the root URL (that is, /) lead to your "hello" view function.

To do this, you first need to import your application's "views" module into urls.py. Once again, you can see how Django uses Python's standards in an intelligent way; if the application is called atfapp and the views file is views.py inside of that directory, you merely have to say:

```
from atfapp.views import hello
```

at the top of urls.py. Once that is done, the name "hello" is defined in the global namespace for urls.py, and you can reference your function by that short name.

You then have to tell Django to point to your function when it receives an HTTP request for /. You do this by creating a URL object and passing it two parameters: a string containing a regular expression and a function. Note that the regular expression can (and often should) contain the ^ and $ characters to anchor the URL to the start and finish of the URL. Moreover, the leading / character of a URL doesn't appear by default in these URL mappings. Thus, to point / to your "hello" function, you configure Django as follows:

```
url(r'^$', hello)
```

Sticking this into the call to the "patterns" function in urls.py that Django installed by default, you end up with the following definitions:

```
urlpatterns = patterns('',
    url(r'^$', hello),
    url(r'^admin/', include(admin.site.urls)),
)
```

Because Django is running in debug mode, you don't even need to restart

your project. Django notices the change to urls.py and restarts the server. Point your browser to http://localhost:8000/, and sure enough, you'll see "hello world".

Named Parameters

That's not bad for a tiny bit of code! But let's go one step further. Say you don't want the URL to be /, but want it to be /hello/NAME instead, where NAME can be someone's name. You then want the view function to notice that name and use it when displaying output.

This means you'll not only have to adjust your URL, but that the URL will need to have a variable second part (that is, NAME), which then will be passed to your view function.

Amazingly, Django makes this very simple, but only if you're comfortable with some of the more advanced features of regular expressions. If you have any experience with regexps, you probably know that you can group characters using parentheses. You probably also know that you can use a question mark (?) to indicate that the preceding character

is optional. So in theory, it should be illegal for a regexp to contain a question mark just inside a pair of parentheses, as in:

```
(?foo)
```

Python, however, allows for this to happen and treats it specially. If, after the question mark, you put the letter P and then a name inside <angle brackets>, the group is captured within the regular expression. Better yet, the captured text can be referred to by name. So if you change your URL to be:

```
^hello/(?P<name>\w+)$
```

it will capture one or more alphanumeric characters (\w+) following "hello/" in the URL. These will be turned into the named group, "name". In urls.py, this will look like the following:

```
url(r'^hello/(?P<name>\w+)$', hello),
```

Once you have done that, you can rewrite your "hello" function, such that it now takes a second parameter, named (not surprisingly) "name". That's right; the named group in your regular expression is then passed to your function as a parameter. I love this feature and find it to be quite elegant—if you can get over the learning curve that people have for regular expressions.

Now change your view function, such that it takes a second parameter:

```
def hello(request, name):
    return HttpResponse("hello,
{}".format(name))
```

And sure enough, after the server reloads, you can visit /hello/Reuven and get a personalized greeting.

Conclusion

In this article, I describe setting up a very simple Django project with an application and wrote a basic view function. In my next article, I'll look at Django's templates and show how you can pass data to them as well as modify that data from within the template without any code.■

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

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

VAULT

LINUX STORAGE AND FILESYSTEMS CONFERENCE

MARCH 11 - 12, 2015

REVERE HOTEL - BOSTON, MA

Vault will bring together the leading developers in file systems and storage in the Linux kernel with related projects to forge a path to continued innovation and education.

WHY YOU SHOULD ATTEND

- 40 technical sessions on the hottest trends and technologies in storage and file systems.
- Keynote presentations by top technical leaders from Facebook, NetApp, Red Hat, and SanDisk.
- A roundtable discussion with the track leaders from the invitation-only Linux Storage Filesystem and MM Summit.

REGISTER TODAY

http://go.linuxfoundation.org/vault2015

LINUX FOUNDATION

DAVE TAYLOR

Let's Play Cards with Acey-Deucey

Time to get your gambling on as Dave implements the simple betting card game Acey-Deucey.

I've been looking at the fundamentals of Bash shell script writing and programming in my past few articles, so it's time to get back to some game coding. In this article, I'm writing about a simple card game called Acey-Deucey. You might have played this with your kids—it's a really simple betting game popular with people who don't get poker (mostly just kidding there).

The idea is easy. The dealer flips up two cards, one on each side of the (face down) deck, then players bet on whether the next card to be flipped up is going to be numerically between the two exposed cards.

For example, if I flipped up a 6 of diamonds and a jack of hearts, you would then bet on whether the next card would be in the range of 7–10. Suit doesn't matter; it's all about rank.

This means, yes, you can do the math easily enough. If you have the entire deck available each time, the chance of any given card coming up is 1/52. Suit doesn't matter, so each card rank has a 1/13 chance of being chosen. In the above example, 7–10 is four possible winning ranks, so the chances are 4/13 or 30%.

But this isn't a math game, so I apologize for the slight sidetrack! This is a programming problem, and it starts where all card games start, with simulating a deck of cards.

To make things interesting, I'm going to approach the problem by having two decks, one sorted one unsorted. The "shuffle" then randomly selects from the sorted deck and adds it to the unsorted (that is, shuffled) deck. Then cards can be "pulled" from the unsorted/shuffled deck sequentially, a simulation that

adds a bit of arguably unnecessary computation, but lets us debug by previewing the top cards on the deck and so on. It also opens up the possibility of cheating, but that's another story entirely.

To start, let's create the sorted deck, modeling it as an array:

```
card=1
while [ $card -le 52 ]        # 52 cards in a deck
do
  deck[$card]=$card
  card=$(( $card + 1 ))
done
```

To create the shuffled deck—herein called "newdeck"—we'll need to have a new function `pickCard` that randomly tries to grab a card out of the ordered deck. Note that this has some nuances, but let's start with the base code:

```
while [ $errcount -lt $threshold ]
do
  randomcard=$(( ( $RANDOM % 52 ) + 1 ))
  errcount=$(( $errcount + 1 ))

  if [ ${deck[$randomcard]} -ne 0 ] ; then
    picked=${deck[$randomcard]}
    deck[$picked]=0          # picked, remove it
    return $picked
  fi
done
```

The `threshold` variable is because although the first 80% of card picks are easy, the last few end up problematic, because we could, for example, end up with an array of 51 empty slots and one available card that we're trying to find with random selections. So `threshold` limits the number of random picks made. If we don't get a valid card within that count, the script will drop down to the less random sequential search for an available card:

```
randomcard=1
while [ ${newdeck[$randomcard]} -eq 0 ]
 do
    randomcard=$(( $randomcard + 1 ))
 done
 picked=$randomcard
 deck[$picked]=0              # picked, remove it
 return $picked
```

It's not so glamorous, but it's functional, and really, by the time we hit this code, we're down to the last half-dozen cards in the deck anyway, and a lack of randomness is unlikely to be noticed, certainly not in Acey-Deucey where we use the entire deck each time we play.

Now that we have a card picking function, we can create the shuffle deck function, which instantiates the array `newdeck` with a randomly

chosen ordering of the sorted cards in `deck`:

```
while [ $count -le 52 ]
do
  pickCard
  newdeck[$count]=$picked
  count=$(( $count + 1 ))
done
```

Still with me? These functions are mnemonically named `initializeDeck`, `shuffleDeck` and `pickCard`, meaning that all we need to do to be ready to start writing some game playing code is this sequence:

```
initializeDeck
shuffleDeck
```

Nice. Now the first step of the actual game is to pick those first two cards, right? Right!

The core code is straightforward, but we have to deal with the fact that numerically we're representing a king as a rank of zero (so that all the other rank cards are their own value). So we need to keep fixing that each time a card is picked:

```
function dealCards
{
    # Acey Deucey has two cards flipped up...
    card1=${newdeck[1]}      # since deck is shuffled, we take
    card2=${newdeck[2]}      # the top two cards from the deck
    card3=${newdeck[3]}      # and pick card #3 secretly

    rank1=$(( $card1 % 13 ))     # get the rank values
    rank2=$(( $card2 % 13 ))     # to make subsequent
    rank3=$(( $card3 % 13 ))     # calculations easy

    # fix to make the king, default rank = 0, have rank = 13

    if [ $rank1 -eq 0 ] ; then
      rank1=13;
    fi
    if [ $rank2 -eq 0 ] ; then
      rank2=13;
    fi
    if [ $rank3 -eq 0 ] ; then
      rank3=13;
    fi

    # now organize them so that card1 is always lower value
    # than card2

    if [ $rank1 -gt $rank2 ] ; then
      temp=$card1; card1=$card2; card2=$temp
      temp=$rank1; rank1=$rank2; rank2=$temp
    fi

    showCard $card1 ; cardname1=$cardname
    showCard $card2 ; cardname2=$cardname

    showCard $card3 ; cardname3=$cardname
    echo "I've dealt:"
    echo "   $cardname1" ; echo "   $cardname2"
}
```

There's a lot going on in that particular function, but don't be too bewildered, the main work is here in these two lines:

```
card1=${newdeck[1]}
rank1=$(( $card1 % 13 ))
```

On first glance, the omission of a `$RANDOM` might cause anxiety, but remember we've already created the shuffled deck as `newdeck[]` in the script. Now we can just grab the top card and save it as `$card1`, then calculate its rank by doing the old mod 13 trick. This, of course, is why a king with a value of 13 ends up being a zero, so we need to fix it a bit later in the function.

The eagle-eyed among you will notice that we've already picked the third card in this function, so we could actually just say "winner!" or "loser!" immediately, dispensing with the entire betting sequence, but what's the point in that?

There's one more function

that's needed for the playing card library in the shell, and that's `showCard`. You can see it referenced above, so let's have a look. Fundamentally, it's easy: pull the rank and suit from the card value, then translate each as needed to make the output pretty.

Here's my shot at that, demonstrating my zeal for case statements:

```
function showCard
{
   # This uses a div and a mod to figure out suit and rank, though
   # in this game only rank matters. Still, presentation is
   # important too, so this helps make things pretty.

   card=$1

   if [ $card -lt 1 -o $card -gt 52 ] ; then
     echo "Bad card value: $card"
     exit 1
   fi

   # div and mod. See, all that math in school wasn't wasted!

   suit="$(( ( ( $card - 1) / 13 ) + 1))"
   rank="$(( $card % 13))"

   case $suit in
     1 ) suit="Hearts"    ;;
     2 ) suit="Clubs"     ;;
     3 ) suit="Spades"    ;;
     4 ) suit="Diamonds" ;;
     * ) echo "Bad suit value: $suit"; exit 1
   esac

   case $rank in
     0 ) rank="King"    ;;
     1 ) rank="Ace"     ;;
     11) rank="Jack"    ;;
     12) rank="Queen"   ;;
   esac

   cardname="$rank of $suit"
}
```

You also could do the same thing, probably quicker, by setting up an array of rank names and an array of suit names, but that can be your version of the code. Me? I like case statements.

Now we have everything we need for the basic game, but I'm going to hold off on the overall game logic and some demonstrations of the game in play until my next article.■

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

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

KYLE RANKIN

Secure Server Deployments in Hostile Territory, Part II

The cloud: where security isn't easy, but it's necessary.

In my last article, I started a series on some of the challenges related to spawning secure servers on Amazon EC2. In that column, I discussed some of the overall challenges EC2 presents for security compared to a traditional infrastructure and elaborated on how I configure security groups and manage secrets. In this article, I finish up the topic with a few more practices I put in place when deploying servers to EC2. As with the previous article, although my examples are with EC2, most of these practices are something you easily could adapt to any cloud environment or even your own infrastructure in most cases.

Puppet-Specific Practices

I know that everyone has their own favorite configuration management system. Without getting into a holy war, we use Puppet for configuration management, so I thought it would be worthwhile to highlight a few security practices related to Puppet itself. I'll keep all of the Puppet-specific tips in this section, so you can skip ahead to the next section if this doesn't interest you.

The first thing to cover in a Puppet deployment is how to spawn servers. I'm a big fan of using vanilla off-the-shelf Amazon AMIs instead of rolling my own. Maintaining my own system images adds a lot of overhead, particularly as things change, and I'd much rather let the Debian team manage that workload. I take the official Debian AMIs and add a userdata script that installs Puppet, and that's it.

In a client/server model, when the client first attempts to check in to the Puppetmaster, it will present a certificate it has generated and request that the Puppetmaster sign it. Once the Puppetmaster signs it, it will trust that the client who presents that certificate is who it says it is. Although Puppet allows for automatic signing of all certificate requests, that's generally regarded as an insecure practice. Instead, what I do is have my spawning script launch another simple script that just checks for a new certificate from the host I spawned for the next ten minutes. If the host checks in within that ten-minute window, it gets signed; otherwise, the script exits. In this way, I create a very small window for an attacker to impersonate a server, and even then, I'd see the duplicate certificate request that I didn't sign on the Puppetmaster. This gives me a lot of the benefits of auto-signing without the risks.

I know a lot of Puppet advocates favor Puppet deployments without a Puppetmaster. In that model, the Puppet configurations are shipped to the hosts, and they apply them directly. Although there can be benefits to this approach, it has a few drawbacks. First, you lose the ability to deploy something like hiera-gpg with ease. More important to me, you lose the handy feature that every Puppet client has generated a valid internal certificate that has been signed by an internal trusted CA (the Puppetmaster). I heavily re-use these certificates internally whenever I need to set up TLS on any server, and we require TLS on all external network services. When another non-root service needs access to the certificate (such as an Nginx server or Postgres), I just make a copy of it that the particular service can see and store it elsewhere on the filesystem.

We also have modified Puppet so that certificates are generated with a Subject Alt Name of role.domainname in addition to the standard hostname.domain each certificate already would have. Because we cluster our internal services for fault tolerance, we refer to services by role, and that role actually could point to three or more servers, each with its own domain name. By adding the role name as a Subject Alt Name to the certificate, it's easy to re-use these certificates for all internal services.

Manage Dynamic IPs

I'll be honest, dynamic IPs are a pain. That said, if you are using public EC2 services, you don't have much of a choice. All IPs are assigned dynamically and change whenever you halt the server. This adds an additional level of complexity to the environment,

because you don't know ahead of time what IP a server will have. To combat this, I have set up an internal DNS server that has been configured to allow dynamic DNS updates. For a host to update its DNS record, however, it needs to have a shared secret that gets generated on the DNS server and exported to Puppet. Once Puppet starts configuring the server, it creates the local key file that contains the secret and builds a local nsupdate script that updates any records the host needs. From that point on, every time Puppet runs, the host confirms all of its dynamic DNS records are accurate. I've also added a hook so that when I decommission a host, Puppet will remove all of those records.

Overall Best Practices

There are a few other general security practices I put in place. First, as I mentioned before, because each host has a certificate signed by an internal trusted CA for Puppet, we take advantage of those certs to require TLS for all network communications between hosts. Given that you are sharing a network with other EC2 hosts, you want to make sure nobody can read your traffic as it goes over this network. In addition, the use of TLS helps us avoid man-in-the-middle attacks.

We also take additional steps with package management. We host internal package repositories that mirror only upstream packages we actually use. We sign any internal packages we create, and hosts validate the signatures before they install packages. Finally, we go to extra steps to restrict access to our environments. Our production environment in particular is restricted to a small number of sysadmins and is accessed via a separate VPN. We require key-based SSH authentication and have an internal policy for sysadmins to password-protect their SSH keys (read my "Secret Agent Man" article in the December 2013 issue for some tips related to SSH agent and password-protected key management).

Although this isn't a complete list of security practices I use (or that you should use), hopefully it's enough to give you an overview of the sorts of considerations you should make if you want to host a secure environment in the cloud.■

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

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

SHAWN POWERS

Geek, Hack Thyself

Being a geek means more than sitting at a keyboard, although admittedly that's part of it.

I've been writing for *Linux Journal* since 1997. Through the years, I've gotten hundreds of e-mails from readers and fellow geeks, but never do I get as much e-mail as when I talk about the geek lifestyle. Whether I'm discussing my network-connected health monitors or my latest attempt at emulating console games from the 1990s, there's something about everyday geek life that is just fun to discuss. For this article, I decided it would be interesting to start a discussion about health and lifestyle, then follow up a little later in the month via Twitter, Facebook, Reddit—or whatever we decide makes the most sense. I must warn you, I'm not limiting myself to technology here. Geeks are people too, and it's fun to see how our geekiness leaks into the rest of our lives!

Work

First off, let's be honest. We spend a lot of time working. Even when we're not at work, we tend to be working on work stuff—work work work. Thankfully for most of us, technology is what we enjoy, so even though we spend a lot of time doing it, work is a labor of love.

My day job is that of a teacher/trainer. I create video-based training for Linux and open-source tools. Because that often means integrating with multiple systems, it means from my desk I'm within arm's reach of:

- An 11" MacBook Air running OS X Mavericks. (I had Yosemite, but the Wi-Fi constantly failed, so I went back to Mavericks—so far, so good.)
- A Wacom Companion tablet computer running Windows 8.1. I use Camtasia to record the training videos and VMware Workstation to run virtual instances of whatever systems I'm using to train.
- An Intel NUC machine running ESXi.

Figure 1. The motorized lifting desk means I can stand or sit, or just play!

For some dumb reason, the folks at Wacom disabled the VT-X capability of the i7 processor on my Companion tablet, so in order to emulate 64-bit machines, I have to have ESXi running on a separate server.

- A headless tower running Ubuntu Server 14.04—because everyone needs a Linux server handy. I use it locally and over SSH when I'm programming or testing new software.
- The Polywell i2303 nettop computer I reviewed in June 2012 (**http://www.linuxjournaldigital.com/linuxjournal/201206/?pg=56**) running Xubuntu 14.04. This computer is still screaming fast, and I use it as my main Linux desktop.

I also have a Workrite Ergonomics desk that allows me to stand up or sit down while working (Figure 1). Sometimes I just raise it and lower it for fun, because it's truly awesome to press a button and have your entire workspace rise into the air!

Figure 2. The Swopper chair takes some getting used to, but it really helps with posture.

When I'm sitting, I use a Swopper Classic chair to help keep me active. You know those animal toys at the park mounted on giant springs so kids can rock back and forth? The Swopper chair is basically an adult version of that (Figure 2). I can bounce, spin and sway all day long, and the instability keeps my core muscles active and forces good posture—or at least that's what I tell myself. I just like to bounce all day!

And of course, no office would be complete without hot caffeinated beverages. This time of year, I tend to drink more tea than coffee. For

Figure 3. The Breville One-Touch adjusts temperature, steeping time and automatically removes tea leaves.

Christmas two years ago, my wife bought me the Breville One-Touch Tea Maker (Figure 3). It makes a perfect pot of tea every time, and it makes brewing tea as simple as brewing a pot of coffee. I never imagined how much of a difference it would make, but truly it makes perfect tea every time, even with delicate white teas.

Play

I stink at video games. I really do. It doesn't make sense that someone with such a geeky lifestyle wouldn't be a gamer, but I'm really not. The only exception to that rule is classic games. I love arcade games, and I love

old-school console games. Thanks to the miracle of emulation, I can play the NES and SNES games I grew up with on modern hardware. In fact, in the next couple issues, I'll be writing about my new console emulation device built inside a cigar box. I can't wait to share it with you!

Most of my playtime is spent reading. My favorite genres are science fiction and fantasy (big surprise, right?). In fact, I'm already connected with many of you on-line at **http://www.goodreads.com**. If you're a science fiction or fantasy reader and want to share book insights, please connect with me (**http://snar.co/goodreads**).

If you've been reading *Linux Journal* for a while, you know that I'm a big fan of audiobooks. I cover my current favorite audiobook player in the UpFront section of this issue (the Listen Audiobook Player). I also read traditional books, but admittedly not as often. For those, I either buy the dead-tree version or read on my Kindle Paperwhite. I keep my non-Amazon e-books organized with Calibre and send them to my Kindle via e-mail whenever I need to load one.

Keeping the Doctor Away

Anyone who follows me on Twitter or reads my personal blog knows that during the past year I've had many health problems. The worst of which was "acute renal failure". That was exactly as scary as it sounds. My kidneys haven't completely failed, and right now, they're functioning at a steady "sorta crappy" level. Thankfully, they don't seem to be getting any worse. Needless to say, it's forced me to take my health a lot more seriously than ever before. Since I'm a geek, that means I'm trying to include as much science and technology into the process as possible.

Step 1: Weight Loss

Somewhere around my mid 20s, I started another growth spurt. Unfortunately, it wasn't the sort of growth a person hopes to accomplish. I grew both in width and gravitational attraction. One of the most important body hacks I'm working on is losing weight. While having a healthy BMI (body mass index) is a good idea anyway, my kidney problems make it vital that my blood pressure stays under control. Being overweight increases blood pressure, and since high blood pressure destroys kidneys, it means I have to lose weight.

I probably should take a second to say right now that I'm not advocating that everyone should do what I do. Our size doesn't define who we are, and I'm not some vain athlete who is worried about what I'll look like on the beach this spring. The concept of me being an

athlete or wanting to go to the beach is just insane. Still, I get a lot of e-mail about my process, so I'm sharing.

The science for losing weight is easy. Burn more calories than you consume. Sure, there are lots of nuances about carbs and fiber and such, but the math is the math. If you burn more than you consume, you will lose weight. There are two variables in that equation. Either you exercise more, eat less, or both. I'm trying desperately to do both. I used to be really good at eating less, but recently, I've been better at exercising. I've never managed to be good at both simultaneously.

The easiest way I've found to succeed with eating less is simply to track what I eat. There's something about consciously recognizing what you eat that helps moderate it. Even if you don't plan to decrease the amount of food you eat, if you record everything, you probably will find yourself eating less. The best tool I've found for recording what I eat is MyFitnessPal. It's a free on-line tool that allows you to add food from its extensive database and have the calories calculated automatically. It even allows you to scan the barcode on packages and import the nutrition information automatically. The social aspect of MyFitnessPal is also helpful, because having accountability partners can make a world of difference. If you'd like to try it with me, and possibly with other *Linux Journal* readers, feel free to connect to me: **http://www.myfitnesspal.com/shawnp0wers**.

When it comes to exercise, there are obviously many ways to burn calories. Some exercises burn more than others, but for me, the best fat-burning activity is walking. Sure, there are other exercises that burn more calories per minute, but I find it difficult to swim or jog very far before tiring out. Walking, however, is an exercise I can do for hours. It burns far more calories to walk for an hour than it does to run for five minutes. And personally, I think it's easier to do the hour of walking.

I try to walk a minimum of 10,000 steps per day. It's easiest to have a pedometer to keep track of your steps during the day. My favorite pedometer is one of the models from **http://www.fitbit.com**. I love the geeky Bluetooth syncing, graphing on so on. Plus, like with MyFitnessPal, there's a social aspect to the FitBit. You can join forces (or compete against) your friends on-line. Sometimes a quick look at the leaderboard will be enough motivation to get out and walk for a while. In fact, ever since we started wearing FitBit trackers, my wife and I have started parking at the far side of the parking lot when we go

shopping. That way, we get a few more steps in while we're getting to the store!

If you'd like to connect with me on FitBit, head over to the *Linux Journal* group page and join (**https://www.fitbit.com/group/22M75H**). The service is free, but the FitBit devices are unfortunately on the pricey side. Still, I think it's possible to join even without a FitBit device. If we get enough people to join, it could turn into quite a group!

Step 2: Keeping Old Age Away!

Walking and weight loss have developed into something that I never in a million years expected. I still have weight to lose, but I've lost enough that I have a little more stamina. After reading a book by Matthew Inman on running, I decided to give it a try. Even if you have no intention of ever running, I highly recommend his book. Most of it is available on-line at his Web site: **http://theoatmeal.com/comics/running**.

Running is a surprisingly controversial activity. Some people say it is hard on joints. Some people say it is good for joints and strengthens them. Many think running helps keep you young, while many others claim running ages you faster. Some people claim it will cause arthritis, while other say it's a great way to build bone density and prevent osteoporosis. The one thing everyone agrees on, however, is that running is a very effective way to get a cardio workout. I never was able to run as a kid, and now that I've been diagnosed with asthma and can treat it, running is possible for the first time in my life.

I'm not a good runner. I'm slow. I can't go very far at once. I concentrate so hard on breathing that I end up spitting and drooling. I jiggle—a lot. Still, I'm slowly getting better at it. After a month or so of working on it, I'm actually starting to enjoy it. It's *very* hard work, but it's the sort of hard work that is rewarding in and of itself. If you've never run before, or if you're even remotely interested in starting, here are a few of my tips. Keep in mind I'm still a beginner, but the following tips got me through the miserable beginning stages.

- If you have health issues, talk to your doctor. Without treating my asthma, I'd never be able to run. Even if you are being treated for asthma, it's good to see your doctor anyway. Oh and if you're a fellow asthmatic who uses an inhaler? I beg you to read this: **http://snar.co/asthma**.

- Buy good shoes. I used to think people who spent lots of money on shoes were crazy. (I still think that's

the case with high heels.) When it comes to running, however, high-quality shoes that have been fitted for your running style are vital. It will help you avoid injury and make the experience far less miserable. Spending upward of $100 on running shoes isn't uncommon. The most important part, however, is to go to a shoe store that specializes in running shoes so you can be properly fitted. Don't go to a general sporting goods store—they won't know what they're doing.

- Start with a program like Couch-2-5K (C25K). There are several Android apps that give you a training schedule and even help you time your sessions to build up to the full 5K. The best part about using one of the C25K programs is that you're forced to build slowly. Doing too much too fast will lead to misery and injury.

- Make a commitment to keep it up until you can run 30 minutes straight. That won't seem possible at first—really. Running one minute at the beginning was almost impossible for me. After several months, however, I finally got to the point where I could run (very slowly) for a full 30 minutes. As if it were some magical rite of passage, when I reached the 30-minute mark, I didn't hate running anymore. Mind you, it's still very hard, it hurts, and it's not exactly "fun", but once you reach that 30 minute mark (which incidentally is about how long it takes to run a 5K), something changes.

Right now I'm to the point where I want to increase my distance and increase my speed. I can run for 30 minutes, but just barely. There is a lot of advice out there for brand-new runners, but oddly, not so much once you reach that magical 5K/30-minute mark. Because I want to keep getting better, I recently thought it would be a good challenge to run at 6mph instead of my regular 5mph. As it's a measly 1mph difference, I didn't think it would be all that bad.

I. Was. Wrong.

I learned through the school of hard knocks that increasing your speed 20% overnight is really dumb. I went from being able to run three miles to almost collapsing after one. It was frustrating. It hurt. I got discouraged and honestly considered quitting the whole running thing. Thankfully, just like with dieting and walking, there are some incredible social tools for runners. And if there's one thing runners are good

at, it's encouraging other runners—especially new runners. I suspect it's because they remember being a beginner, and know just how horrible and difficult it is at first.

By the time this issue goes out, I hope to be one of those crazy running guys who goes out and runs 5–6 miles on a regular basis. It's still hard for me to imagine that being the case, but I really hope it happens. If you want to start running and could use some encouragement, I urge you to check out a site like **http://dailymile.com**. It's a great way to record your runs, but more important, it's a community of runners. Have I mentioned how encouraging runners can be to other runners, especially beginners? Look me up on the Dailymile Web site—maybe we can encourage each other (**http://www.dailymile.com/people/shawnp0wers**).

Life Hacking

This article was largely focused on my day-to-day life in geek world. Normally I'd say drop me an e-mail with your particular twist on living the geek life, but my e-mail box has been largely untamable lately. You still can e-mail me if you like, but please don't hesitate to reach out via Twitter, Facebook or Google+. I have lots of ideas for future lifestyle columns, like:

- Souped up battery backup units.
- Home automation.
- Photo consolidation and management.
- Home networking.
- Car-mounted technology.

But, I'd love to hear ideas and requests from you. I had a lot of fun talking about the geek life here, and I can't wait to hear from all of you on-line!■

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.

Resources

Twitter: @shawnp0wers

Facebook: **http://facebook.com/linuxjournalshawn**

Google+: **http://snar.co/googleplus**

TRENDnet's THA-101 Home Smart Switch with Wireless Extender

So your attic is the only place where real work happens, Junior is hooked on Angry Birds Transformers, and the router is in the basement. What to do? TRENDnet seeks to keep you productive in any of your home's Wi-Fi-free corners with its new THA-101 Home Smart Switch with Wireless Extender, a combination smart electrical outlet and powerful N300 wireless extender. Simply place the THA-101 in a location where you get at least two bars worth of home wireless signal from an existing wireless network. Next, plug in the THA-101 and follow a quick app-based installation. The powerful built-in N300 wireless extender then broadcasts a strong wireless signal to an area of your home with low or no wireless coverage. Then, plug anything in to the THA-101's electrical outlet and use the free mobile app to control the outlet from a mobile device with an Internet connection, including an on/off schedule. The app also displays a delicious feast of geek-friendly data, namely real-time current, voltage, power and total energy consumption.
www.trendnet.com

AdaCore's GNATdashboard

The Ada programming language is geared toward industries where software is mission-critical and often safety- or security-critical as well. Ergo, quality assurance monitoring in Ada, which is what AdaCore's new GNATdashboard is all about, is of utmost importance. GNATdashboard is AdaCore's new tool that serves as a control panel for monitoring and improving the quality of Ada software. The new tool feeds code-quality information from AdaCore's tools to the open-source SonarQube code quality management platform and Squoring Technologies' SQUORE quality and performance decision support solution. This helps quality assurance managers and project leaders understand or reduce their software's technical debt, eliminating the need for manual input. GNATdashboard fits naturally into a continuous integration environment, says AdaCore, providing users with metrics on code complexity, code coverage, conformance to coding standards and more.
www.adacore.com

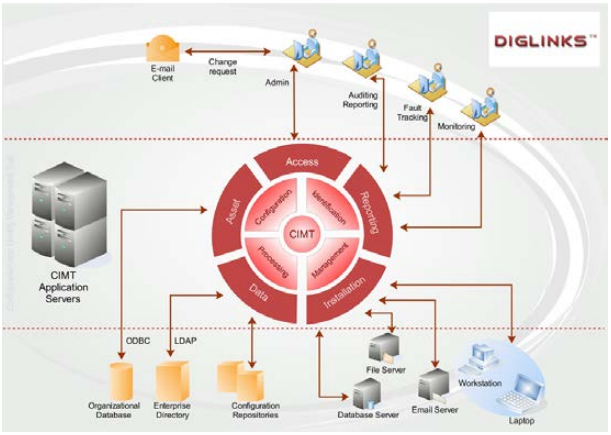
Violin Memory's Oracle Performance Analysis Service (O-PAS)

Poor database and application performance easily can result in lost revenue, extra IT and database manpower expenses, and unacceptable operational and capital expenses. Solving this bundle of challenges is the inspiration for Violin Memory's new Oracle Performance Analysis Service, or O-PAS, a free tool designed to remove the complexity of reporting, analyzing and decision-making around Oracle databases. The goal with O-PAS is to assist database administrators, application owners, IT executives and resellers in making informed decisions and selecting the right solution for achieving significant performance improvements and cost savings. The Violin O-PAS identifies core performance bottlenecks in end users' Oracle environments and determines the suitability of Flash-based storage solutions to increase IOPS, reduce latency and lower overall costs substantially. The tool generates comprehensive reports that identify areas in Oracle implementations where massive performance improvements can be gained by using all-Flash arrays and the associated cost savings that they enable.
www.violin-memory.com

Ron Fuller, David Jansen and Matthew McPherson's *NX-OS Configuration Fundamentals LiveLessons* (Cisco Press)

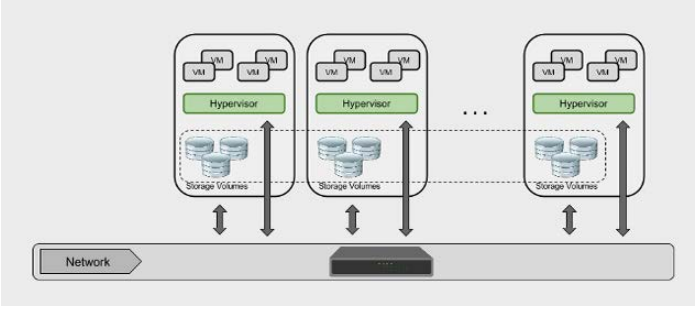
NX-OS is an embedded Linux-based network OS from Cisco Systems that is designed to support the high-performance, high-reliability server-access switches found in data centers. Network engineers or operators who want to gain a solid understanding of NX-OS technologies would be wise to investigate a new video training product called *NX-OS Configuration Fundamentals LiveLessons* from Cisco Press. Authors Ron Fuller, David Jansen and Matthew McPherson offer 12 hours of instruction on NX-OS technologies and configuration, covering 13 individual video lessons, which are in turn subdivided into 88 sub-lessons. The lessons guide students from an introduction to the product families and operating system to Layer 2 and 3 capabilities before moving on to multicast and security. High availability, unique embedded serviceability features, and Unified Fabric and the Nexus 1000v are covered as well, followed by QoS, OTV, MPLS and LISP.
ciscopress.com

DigLinks' CIMT



Problems like server breakdowns, errors related to hardware and software management and typical technical issues that stymie everyday business operations can all add unplanned costs. To help businesses minimize these costs, system and network solution provider DigLinks created the Configuration and Identity Management, or CIMT, a new open-source IT management solution that allows for central, user-friendly management of the entire IT infrastructure in small- to large-scale enterprises. DigLinks states that CIMT can be integrated into IT landscapes easily and with minimal effort—irrespective of the number of systems, types of operating systems, hardware components or network connections. In addition, a customized, scalable service package is available, which frees companies from time-consuming routine work on their technical operations and focuses on their core business. CIMT is written in Java for portability and covers all aspects of IT management—users, workstations, IPs, network devices or servers, hardware and software—from a single application.

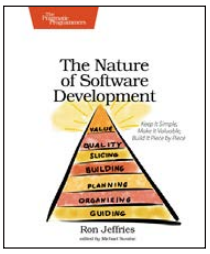
www.diglinks.com



Inverness Data's RapidStor

Whether you are operating on spinning magnetic disk or solid-state drives or hosting data for your company or for the cloud, Inverness Data invites you to utilize the new stable release of its flagship product RapidStor. RapidStor is a complete fault-tolerant and high-performance data storage infrastructure. The solution is software-defined and capable of maintaining and managing data volumes and virtual machines from a single node to an entire cluster of multiple nodes. RapidStor is designed to meet constantly evolving requirements by scaling both up and out, all of which runs transparently to the user under a single management interface. Because RapidStor runs on commodity hardware, Inverness notes the advantage of freedom from hardware vendor lock-in.

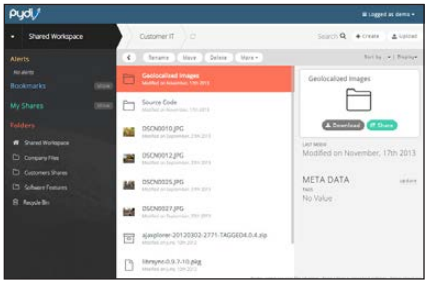
inverness-data.com



Ron Jeffries' *The Nature of Software Development* (Pragmatic Bookshelf)

The only things you'll need to take advantage of Ron Jeffries' *The Nature of Software Development* are "your Standard Issue Brain, a bit of curiosity and a desire to build your own understanding". Subtitled *Keep It Simple, Make It Valuable, Build It Piece by Piece*, Jeffries' book is intended to help you get value from your software project, perhaps not "free, now and perfect" as you wish but at least "cheaper, sooner and better". Jeffries leads readers from the desire for value down to the specific activities that help good Agile projects deliver better software sooner and at a lower cost. Using simple sketches and a few words, Jeffries invites readers to follow his path of learning and understanding from his engagement with Agile methods from their very beginning. Jeffries also illustrates why Agile methods ask for what they do, as well as why a shallow implementation of Agile can lead to only limited improvement. The ultimate goal is to embark on a long-term journey to make a project a bit more perfect each day and ultimately end up with a deep understanding of the nature of software development done well.

pragprog.com



Pydio

Companies currently are exposed to massive liability when employees use uncontrolled consumer services like Dropbox to share and store sensitive company information. Breathe a sigh of relief ye IT departments and developers, knowing that solutions like the new-and-improved Pydio 6.0 exist, a controlled, open-source alternative to consumer cloud services. Formerly called AjaXplorer, Pydio 6.0 is described as "the first open-source file sharing solution to deliver tight control of information on the scale demanded by enterprises and service providers, and to combine this with the same intuitive ease of use found in modern consumer apps and cloud services". The platform's wealth of deeper, enterprise-focused features still are available for power users and system administrators, but these have been moved to a second level to avoid confusing the majority of users. The company also touts Pydio as the perfect solution for developers looking to build intuitive multiplatform file sharing into their own products or cloud service offerings. Finally, Pydio enables the integration of different client applications (Linux, Android, Windows, MacOS, iOS and so on) into a unified, multidevice platform.

pyd.io

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.

JavaScript All the Way Down

Use JavaScript for server and client programming.

FEDERICO KEREKI

There is a well known story about a scientist who gave a talk about the Earth and its place in the solar system. At the end of the talk, a woman refuted him with "That's rubbish; the Earth is really like a flat dish, supported on the back of a turtle." The scientist smiled and asked back "But what's the turtle standing on?", to which the woman, realizing the logical trap, answered, "It's very simple: it's turtles all the way down!" No matter the verity of the anecdote, the identity of the scientist (Bertrand Russell or William James are sometimes mentioned), or even if they were turtles or tortoises, today we may apply a similar solution to Web development, with "JavaScript all the way down".

What's in a Name?

JavaScript originally was developed at Netscape in 1995, first under the name Mocha, and then as LiveScript. Soon (after Netscape and Sun got together; nowadays, it's the Mozilla Foundation that manages the language) it was renamed JavaScript to ride the popularity wave, despite having nothing to do with Java. In 1997, it became an industry standard under a fourth name, ECMAScript. The most common current version of JavaScript is 5.1, dated June 2011, and version 6 is on its way. (However, if you want to use the more modern features, but your browser won't support them, take a look at the Traceur compiler, which will back-compile version 6 code to version 5 level.)

Some companies produced supersets of the language, such as Microsoft, which developed JScript (renamed to avoid legal problems) and Adobe, which created ActionScript for use with Flash.

There are several other derivative languages (which actually compile to JavaScript for execution), such as the more concise CoffeeScript, Microsoft's TypeScript or Google's most recent AtScript (JavaScript plus Annotations), which was developed for the Angular.JS project. The asm.js project even uses a JavaScript subset as a target language for efficient compilers for other languages. Those are many different names for a single concept!

If you are going to develop a Web site, for client-side development, you could opt for Java applets, ActiveX controls, Adobe Flash animations and, of course, plain JavaScript. On the other hand, for server-side coding, you could go with C# (.Net), Java, Perl, PHP and more, running on servers, such as Apache, Internet Information Server, Nginx, Tomcat and the like. Currently, JavaScript allows you to do away with most of this and use a single programming language, both on the client and the server sides, and with even a JavaScript-based server. This way of working even has produced a totally JavaScript-oriented acronym along the lines of the old LAMP (Linux+Apache+MySQL+PHP) one: MEAN, which stands for MongoDB (a NoSQL database you can access with JavaScript), Express (a Node.js module to structure your server-side code), Angular.JS (Google's Web development framework for client-side code) and Node.js.

In this article, I cover several JavaScript tools for writing, testing and deploying Web applications, so you can consider whether you want to give a twirl to a "JavaScript all the way down" Web stack.

Why JavaScript?

Although stacks like LAMP or its Java, Ruby or .Net peers do power many Web applications today, using a single language both for client- and server-side development has several advantages, and companies like Groupon, LinkedIn, Netflix, PayPal and Walmart, among many more, are proof of it.

Modern Web development is split between client-side and server-side (or front-end and back-end) coding, and striving for the best balance is more easily attained if your developers can work both sides with the same ease. Of course, plenty of developers are familiar with all the languages needed for both sides of coding, but in any case, it's quite probable that they will be more productive at one end or the other.

Many tools are available for JavaScript (building, testing, deploying and more), and you'll be able to use them for all components in your system (Figure 1). So, by going with the same single set of tools, your experienced JavaScript developers will be able to play both sides, and you'll have fewer problems getting the needed programmers for your company.

Of course, being able to use a single language isn't the single key point. In the "old days" (just a few years ago!), JavaScript lived exclusively in browsers

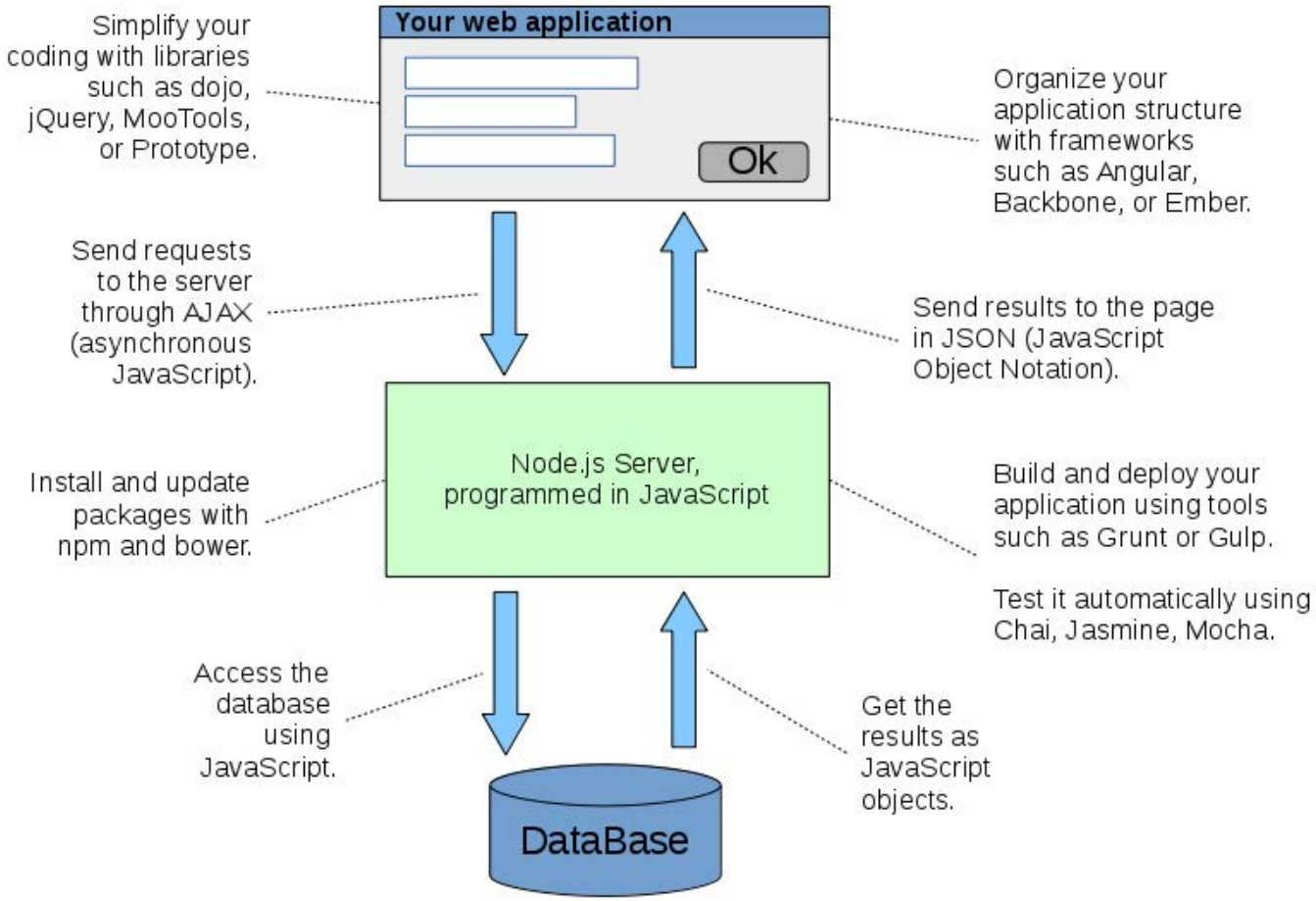


Figure 1. JavaScript can be used everywhere, on the client and the server sides.

to read and interpret JavaScript source code. (Okay, if you want to be precise, that's not exactly true; Netscape Enterprise Server ran server-side JavaScript code, but it wasn't widely adopted.) About five years ago, when Firefox and Chrome started competing seriously with (by then) the most popular Internet Explorer, new JavaScript engines were developed, separated from the layout engines that actually drew the HTML pages seen on browsers. Given the rising popularity of AJAX-based applications, which required more processing power on the client side, a competition to provide the fastest JavaScript started, and it hasn't stopped yet. With the higher performance achieved, it became possible to use JavaScript more widely (Table 1).

Some of these engines apply advanced techniques to get the most speed and power. For example, V8

Table 1. The Current Browsers and Their JavaScript Engines

Browser	JavaScript Engine
Chrome	V8
Firefox	SpiderMonkey
Opera	Carakan
Safari	Nitro

compiles JavaScript to native machine code before executing it (this is called JIT, Just In Time compilation, and it's done on the run instead of pre-translating the whole program as is traditional with compilers) and also applies several optimization and caching techniques for even higher throughput. SpiderMonkey includes IonMonkey, which also is capable of compiling JavaScript code to object code, although working in a more traditional way. So, accepting that modern JavaScript engines have enough power to do whatever you may need, let's now start a review of the Web stack with a server that wouldn't have existed if it weren't for that high-level language performance: Node.js.

Node.js: a New Kind of Server

Node.js (or plain Node, as it's usually called) is a Web server, mainly written itself in JavaScript, which uses that language for all scripting. It originally was developed to simplify developing real-time Web sites with push capabilities—so instead of all communications being client-originated, the server might start a connection with a client by itself. Node can work with lots of live connections, because it's very lightweight in terms of requirements. There are two key concepts to Node: it runs a single process (instead of many), and all I/O (database queries, file accesses and so on) is implemented in a non-blocking, asynchronous way.

Let's go a little deeper and further examine the main difference between Node and more traditional servers like Apache. Whenever Apache receives a request, it starts a new, separate thread (process) that uses RAM of its own and CPU processing power. (If too many threads are running, the request may have to wait a bit longer until it can be started.) When

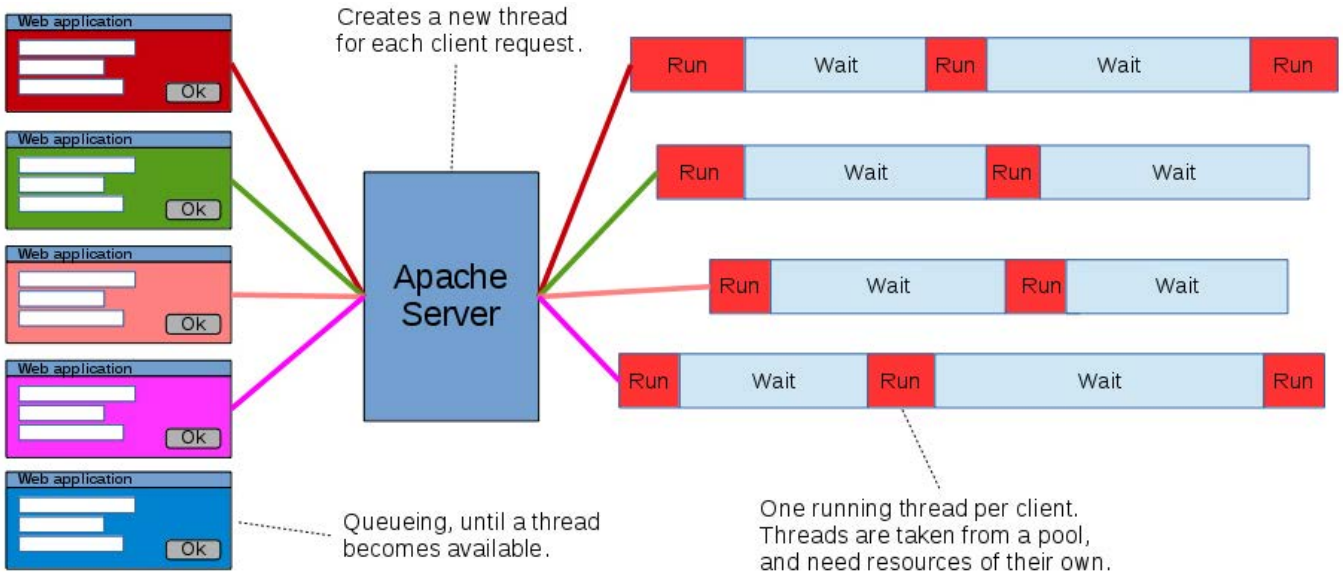


Figure 2. Apache and traditional Web servers run a separate thread for each request.

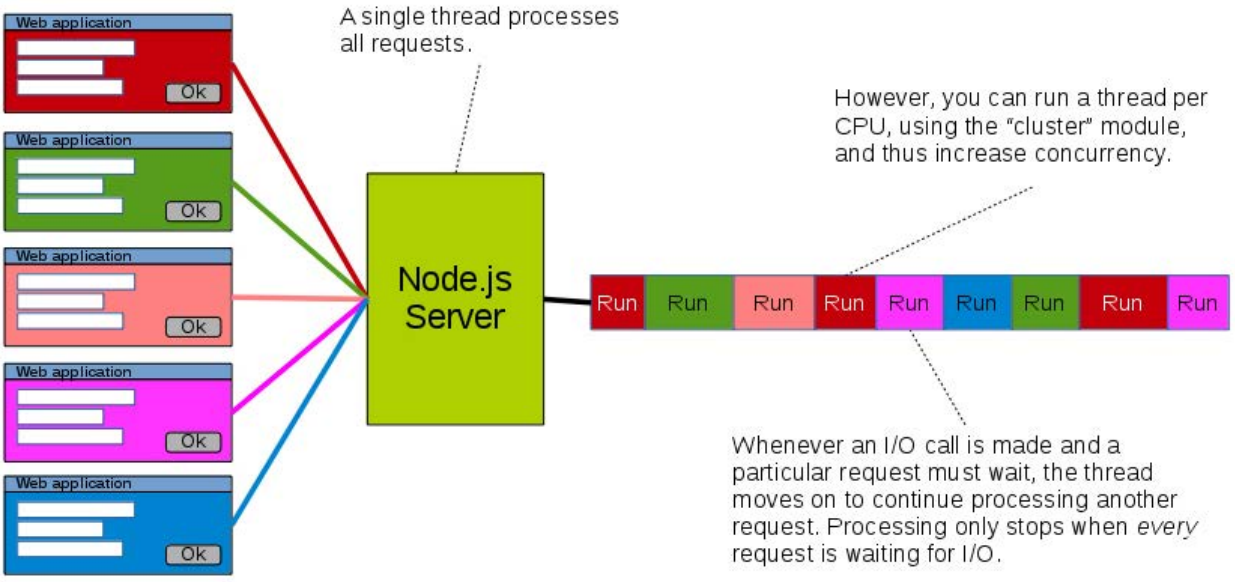


Figure 3. Node runs a single thread for all requests.

the thread produces its answer, the thread is done. The maximum number of possible threads depends on the average RAM requirements for a process; it might be a few thousand at the same time, although numbers vary depending on server size (Figure 2).

On the other hand, Node runs a single thread. Whenever a request is received, it is processed as soon as it's possible, and it will run continuously until some I/O is required. Then, while the code waits for the I/O results to be available, Node will be able to

Table 2. Some widely used Node.js modules that will help your development and operation.

Module	Description
async	Simplifies asynchronous work, a possible alternative to promises.
cluster	Improves concurrency in multicore systems by forking worker processes. (For further scalability, you also could set up a reverse proxy and run several Node.js instances, but that goes beyond the objective of this article.)
connect	Works with "middleware" for common tasks, such as error handling, logging, serving static files and more.
ejs, handlebars or jade, EJS	Templating engines.
express	A minimal Web framework—the E in MEAN.
forever	A command-line tool that will keep your server up, restarting if needed after a crash or other problem.
mongoose, cradle, sequelize	Database ORM, for MongoDB, CouchDB and for relational databases, such as MySQL and others.
passport	Authentication middleware, which can work with OAuth providers, such as Facebook, Twitter, Google and more.
request or superagent	HTTP clients, quite useful for interacting with RESTful APIs.
underscore or lodash	Tools for functional programming and for extending the JavaScript core objects.

process other waiting requests (Figure 3). Because all requests are served by a single process, the possible number of running requests rises, and there have been experiments with more than one million concurrent connections—not shabby at all! This shows that an ideal use case for Node is having server processes that are light in CPU processing, but high on I/O. This will allow more requests to run at the same time; CPU-intensive server processes would block all other waiting requests and produce a high drop in output.

A great asset of Node is that there are many available modules (an estimate ran in the thousands) that help you get to production more quickly. Though I obviously can't list all of them, you probably should consider some of the modules listed in Table 2.

Of course, there are some caveats when using Node.js. An obvious one is that no process should do heavy computations, which would "choke" Node's single processing thread. If such a process is needed, it should be done by an external process (you might want to consider using a message queue for this) so as not to block other requests. Also, care must be taken with error processing. An unhandled exception might cause the whole server to crash eventually, which wouldn't bode well for the server as a whole. On the other hand, having a large community of users and plenty of fully available, production-level, tested code already on hand can save you quite a bit of development time and let you set up a modern, fast server environment.

Planning and Organizing Your Application

When starting out with a new project, you could set up your code from zero and program everything from scratch, but several frameworks can help you with much of the work and provide clear structure and organization to your Web application. Choosing the right framework will have an important impact on your development time, on your testing and on the maintainability of your site. Of course, there is no single answer to the question "What framework is best?", and new frameworks appear almost on a daily basis, so I'm just going with three of the top solutions that are available today: AngularJS, Backbone and Ember. Basically, all of these frameworks are available under permissive licenses and give you a head start on developing modern SPA (single page applications). For the server side, several packages (such as Sails, to give just one example) work with all frameworks.

AngularJS (or Angular.JS or just plain Angular—take your pick) was developed in 2009 by Google, and its current version is 1.3.4, dated November 2014. The framework is based on the idea that declarative programming is best for interfaces (and imperative programming for the business logic), so it extends HTML with custom tag attributes that are used to bind input and output data to a JavaScript model. In this fashion, programmers don't have to manipulate the Web page directly, because it is updated automatically. Angular also focuses on testing, because the difficulty of automatic testing heavily depends upon the code structure. Note that Angular is the A in MEAN, so there are some other frameworks that expand on it, such as

MEAN.IO or MEAN.JS.

Backbone is a lighter, leaner framework, dated from 2010, which uses a RESTful JSON interface to update the server side automatically. (Fun fact: Backbone was created by Jeremy Ashkenas, who also developed CoffeeScript; see the "What's in a Name?" sidebar.) In terms of community size, it's second only to Angular, and in code size, it's by far the smallest one. Backbone doesn't include a templating engine of its own, but it works fine with Underscore's templating, and given that this library is included by default, it is a simple choice to make. It's considered to be less "opinionated" than other frameworks and to have a quite shallow learning curve, which means that you'll be able to start working quickly. A deficiency is that Backbone lacks two-way data binding, so you'll have to write code to update the view whenever the model changes and vice versa. Also, you'll probably be manipulating the Web page directly, which will make your code harder to unit test.

Finally, Ember probably is harder to learn than the other frameworks, but it rewards the coder with higher performance. It favors "convention over configuration", which likely will make Ruby on Rails or Symfony users feel right at home. It integrates easily with a RESTful server side, using JSON for communication. Ember includes Handlebars (see Table 2) for templating and provides two-way updates. A negative point is the usage of `<script>` tags for markers, in order to keep templates up to date with the model. If you try to debug a running application, you'll find plenty of unexpected elements!

Simplify and Empower Your Coding

It's a sure bet that your application will need to work with HTML, handle all kinds of events and do AJAX calls to connect with the server. This should be reasonably easy—although it might be plenty of work—but even today, browsers do not have exactly the same features. Thus, you might have to go overboard with specific browser-detection techniques, so your code will adapt and work everywhere. Modern application users have grown accustomed to working with different events (tap, double tap, long tap, drag and drop, and more), and you should be able to include that kind of processing in your code, possibly with appropriate animations. Finally, connecting to a server is a must, so you'll be using AJAX functions all the time, and it shouldn't be a painful experience.

Listing 1. A simple jQuery example, showing how to process events, access the page and use AJAX.

```
var myButtonId = "#processButton");
$(myButtonId).click(function(e) {          // when clicked...
    $(myButtonId).attr("disabled", "disabled");  // disable button
    $.get("my/own/services", function(data) {    // call server service
      window.alert("This came back: " + data); // show what it returns
      $(myButtonId).removeAttr("disabled");      // re-enable the button
    }
});
```

The most probable candidate library to help you with all these functions is jQuery. Arguably, it's the most popular JavaScript library in use today, employed at more than 60% of the most visited Web sites. jQuery provides tools for navigating your application's Web document, handles events with ease, applies animations and uses AJAX (Listing 1). Its current version is 2.1.1 (or 1.11.1, if you want to support older browsers), and it weighs in at only around 32K. Some frameworks (Angular, for example) even will use it if available.

Other somewhat less used possibilities could be Prototype (current version 1.7.2), MooTools (version 1.5.1) or Dojo Toolkit (version 11). One of the key selling points of all these libraries is the abstraction of the differences between browsers, so you can write your code without worrying if it will run on such or such browser. You probably should take a look at all of them to find which one best fits your programming style.

Also, there's one more kind of library you may want. Callbacks are familiar to JavaScript programmers who need them for AJAX calls, but when programming for Node, there certainly will be plenty of them! You should be looking at "promises", a way of programming that will make callback programming more readable and save you from "callback hell"—a situation in which you need a callback, and that callback also needs a callback, which also needs one and so on, making code really hard to follow. See Listing 2, which also shows the growing indentation that your code will need. I'm omitting error-processing code, which would make the example even messier!

The behavior of promises is standardized through the "Promises/

Listing 2. Callback hell happens when callbacks include callbacks, which include callbacks and so on.

```
function nurseryRhyme(...) {
  ..., function eeny(...) {
    ..., function meeny(...) {
      ..., function miny(...) {
        ..., function moe(...) {
          ...
        }
      }
    }
  }
}
```

A+" open specification. Several packages provide promises (jQuery and Dojo already include some support for them), and in general, they even can interact, processing each other's promises. A promise is an object that represents the future value of an (usually asynchronous) operation. You can process this value through the promise `.then(...)` method and handle exceptions with its `.catch(...)` method. Promises can be chained, and a promise can produce a new promise, the value of which will be processed in the next `.then(...)`. With this style, the callback hell example of Listing 2 would be converted into more understandable code; see Listing 3. Code, instead of being more and more

Listing 3. Using promises produces far more legible code.

```
nurseryRhyme(...).
  .then(function eeny(...) {...})
  .then(function meeny(...) {...})
  .then(function miny(...) {...})
  .then(function moe(...) {...});
```

indented, stays aligned to the left. Callbacks still are being (internally) used, but your code doesn't explicitly work with them. Error handling is also simpler; you simply would add appropriate `.catch(...)` calls.

You also can build promises out of more promises—for example, a service might need the results of three different callbacks before producing an answer. In this case, you could build a new single promise out of the three individual promises and specify that the new one will be fulfilled only when the other three have been fulfilled. There also are other constructs that let you fulfill a promise when a given number (possibly just one) of "sub-promises" have been fulfilled. See the Resources section for several possible libraries you might want to try.

I have commented on several tools you might use to write your application, so now let's consider the final steps: building the application,

testing it and eventually deploying it for operation.

Testing Your Application

No matter whether you program on your own or as a part of a large development group, testing your code is a basic need, and doing it in an automated way is a must. Several frameworks can help you with this, such as Intern, Jasmine or Mocha (see Resources). In essence, they are really similar. You define "suites", each of which runs one or more "test cases",

Listing 4. Suites usually include several test cases.

```
describe("Prime numbers tests", function() {
  it("Test prime numbers", function() {
    expect(isPrime(2)).to.be.true();
    expect(isPrime(5)).to.be.true();
  });

  it("Test non-prime numbers", function() {
    expect(isPrime(1).to.be.false();
    expect(isPrime(4).to.be.not.true();   // just for variety!
    expect(isPrime(NaN)).to.throw(err);
  });
});
```

Listing 5. Some examples of the many available matchers you can use to write assertions.

```
expect(someFunction(...)).to.be.false();  // or .true(), .null(),
                                          // .undefined(), .empty()
expect(someFunction(...)).to.not.equal(33);  // also .above(33),
                                             // .below(33)
expect(someFunction(...)).to.be.within(30,40);
expect(someObject(...)).to.be.an.instanceOf(someClass);
expect(someObject(...)).to.have.property("key", 22);
expect(someResult(...)).to.have.length.above(2);
expect(someString(...)).to.match(/^aRegularExpression/);
expect(failedCall(...)).to.throw(err);
```

which test that your code does some specific function. To test results and see if they satisfy your expectations, you write "assertions", which basically are conditions that must be satisfied (see Listing 4 for a simple example).

Listing 6. A sample test with Zombie (using promises, by the way) requires no actual browser.

```
var browser = require("zombie").create();
browser.localhost("your.own.url.com", 3000);
browser.visit("/")
  .then(function() { // when loaded, enter data and click
    return browser
      .fill("User", "fkereki")
      .fill("Password", ¬°")
      .pressButton("Log in");
  })
  .done(function() { // page loaded
    browser.assert.success();
    browser.assert.text("#greeting", "Hi, fkereki!");
  });
```

A Slew of DDs!

Modern agile development processes usually emphasize very short cycles, based on writing tests for code yet unwritten, and then actually writing the desired code, the tests being both a check that the code works as desired and as a sort of specification in itself. This process is called TDD (Test-Driven Development), and it usually leads to modularized and flexible code, which also is easier to understand, because the tests help with understanding. BDD (Behavior-Driven Development) is a process based on TDD, which even specifies requirements in a form quite similar to the matchers mentioned in this article. Yet another "DD" is ATDD (Acceptance Test-Driven Development), which highlights the idea of writing the (automated) acceptance tests even before programmers start coding.

You can run test suites as part of the build process (which I explain below) to see if anything was broken before attempting to deploy the newer version of your code.

Tests can be written in "fluent" style, using many matchers (see Listing 5 for some examples). Several libraries provide different ways to write your tests, including Chai, Unit.js, Should.js and Expect.js; check them out to decide which one suits you best.

If you want to run tests that involve a browser, PhantomJS and Zombie provide a fake Web environment, so you can run tests with greater speed than using tools like Selenium, which would be more appropriate for final acceptance tests.

Building and Deploying

Whenever your code is ready for deployment, you almost certainly will have to do several repetitive tasks, and you'd better automate them. Of course, you could go with classic tools like `make` or Apache's `ant`, but keeping to the "JavaScript all the way down" idea, let's look at a pair of tools, Grunt and Gulp, which work well.

Grunt can be installed with `npm`. Do `sudo npm install -g grunt-cli`, but this isn't enough; you'll have to prepare a gruntfile to let it know what it should do. Basically, you require a package.json file that describes the packages your system requires and a Gruntfile.js file that describes the tasks you need. Tasks may have subtasks of their own, and you may choose to run the whole task or a specific subtask. For each task, you will define (in JavaScript, of course) what needs to be done (Listing 7). Running `grunt` with no parameters will run a default (if given) task or the whole gamut of tasks.

Gulp is somewhat simpler to set up (in fact, it was created to simplify Grunt's configuration files), and it depends on what its authors call "code-over-configuration". Gulp works in "stream" or "pipeline" fashion, along the lines of Linux's command line, but with JavaScript plugins. Each plugin takes one input and produces one output, which automatically is fed to the next plugin in the queue. This is simpler to understand and set up, and it even may be faster for tasks involving several steps. On the other hand, being a newer project implies a smaller community of users and fewer available plugins, although both situations are likely to work out in the near future.

You can use them either from within a development environment (think Eclipse or NetBeans, for example), from the command line or as "watchers",

Listing 7. A Sample Grunt File

```
module.exports = function(grunt) {
  grunt.initConfig({
    concat: {
      dist: {
        dest: 'dist/build.js',
        src: ['src/**.js'],
      },
      options:    {
      separator: ';',
      stripBanners : true,
      },
    },

    uglify: {
      dist: {
        files:    { 'dist/build.min.js':
           ➥['<%= concat.dist.dest %>'] },
      }
    },
  });

  grunt.loadNpmTasks('grunt-contrib-concat');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.registerTask('default', ['concat', 'uglify']);
};
```

setting them up to monitor specific files or directories and run certain tasks whenever changes are detected to streamline your development process further in a completely automatic way. You can set up things so that templates will be processed, code will be minified, SASS or LESS styles will be converted in pure CSS, and the resulting files will be moved to the server, wherever it is appropriate for them. Both tools have their fans, and you should try your hand at both to decide which you prefer.

Getting and Updating Packages

Because modern systems depend on lots of packages (frameworks, libraries, styles, utilities and what not), getting all of them and, even worse, keeping them updated, can become a chore. There are two tools for this: Node's own `npm` (mainly for server-side packages, although it can work for client-side code too) and Twitter's `bower` (more geared to the client-side parts). The former deals mainly with Node packages and will let you keep your server updated based on a configuration file. The latter, on the other hand, can install and update all kinds of front-end components (that is, not only JavaScript files) your Web applications might need also based on separate configuration metadata files.

Usage for both utilities is the same; just substitute `bower` for `npm`, and you're done. Using `npm search for.some.package` can help you find a given package. Typing `npm install some.package` will install it, and adding a `--save` option will update the appropriate configuration file, so future `npm update` commands will get everything up to date. In a pinch, `npm` also can be used as a replacement for `bower`, although then you'll possibly want to look at `browserify` to organize your code and prepare your Web application for deployment. Give it a look just in case.

Conclusion

Modern fast JavaScript engines, plus the availability of plenty of specific tools to help you structure, test or deploy your systems make it possible to create Web applications with "JavaScript all the way down", helping your developers be more productive and giving them the possibility of working on both client and server sides with the same tools they already are proficient with. For modern development, you certainly should give this a thought.■

Federico Kereki is a Uruguayan systems engineer with more than 25 years of experience developing systems, doing consulting work and teaching at universities. He currently is working as a UI Architect at Globant, using a good mixture of development frameworks, programming tools and operating systems—and FLOSS, whenever possible! A couple years ago, he wrote the *Essential GWT* book, in which you also can find some security concerns for Web applications. You can reach Federico at fkereki@gmail.com.

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

Resources

Keep up to date with JavaScript releases, features and more at **https://developer.mozilla.org/en-US/docs/Web/JavaScript**.

Get the Traceur compiler at **https://github.com/google/traceur-compiler**. CoffeeScript can be found at **http://coffeescript.org**, and TypeScript is at **http://www.typescriptlang.org**. You can read a draft version of the AtScript Primer at **https://docs.google.com/document/d/11YUzC-1d0V1-Q3V0fQ7KSit97HnZoKVygDxpWzEYW0U/mobilebasic**. Finally, for more details on asm.js, go to **http://asmjs.org**.

Common client-side frameworks are AngularJS at **https://angularjs.org**, Backbone at **http://backbonejs.org** and Ember at **http://emberjs.com**, among many others. With Backbone, you also might consider Chaplin (**http://chaplinjs.org**) or Marionette (**http://marionettejs.com**) for large-scale JavaScript Web applications. MEAN.JS is at **http://meanjs.org**, MEAN.IO is at **http://mean.io**, and Sails is at **http://sailsjs.org**.

You certainly should use libraries like jQuery (**http://jquery.com**), MooTools (**http://mootools.net**), Dojo (**http://dojotoolkit.org**) or Prototype (**http://prototypejs.org**).

Use “promises” to simplify callback work in Node; you can choose among Q (**https://github.com/kriskowal/q**), when (**https://github.com/cujojs/when**), bluebird (**https://github.com/petkaantonov/bluebird**) or kew (actually, an optimized subset of Q, at **https://github.com/Medium/kew**), among many more. You can find the standard “Promises/A+” documentation at **https://promisesaplus.com**. An alternative to promises could be async (**https://github.com/caolan/async**).

For server-side package management, use npm from **https://www.npmjs.org**. For client-side packages, either add browserify from **http://browserify.org**, or get bower from **http://bower.io**.

Working with CSS is simpler with Sass (**http://sass-lang.com**) or {less} (**http://lesscss.org**); note that the latter can be installed with npm.

Use testing frameworks, such as Intern (**http://theintern.io**), Jasmine (**http://jasmine.github.io**) or Mocha (**http://mochajs.org**). Chai (**http://chaijs.com**), Should.js (**https://github.com/tj/should.js**), Expect.js (**https://github.com/mjackson/expect**) and UnitJS (**http://unitjs.com**) are complete assertion libraries with different interfaces to suit your preferences. (UnitJS actually includes Should.js and Expect.js to give you more freedom in choosing your preferred assertion writing style.) PhantomJS (**http://phantomjs.org**) and Zombie.js (**http://zombie.labnotes.org**) allow you to run your tests without using an actual browser, for higher speeds, while Selenium (**http://www.seleniumhq.org**) is preferred for actual acceptance tests.

Deploy your systems with Grunt (**http://gruntjs.com**) or Gulp (**http://gulpjs.com**).

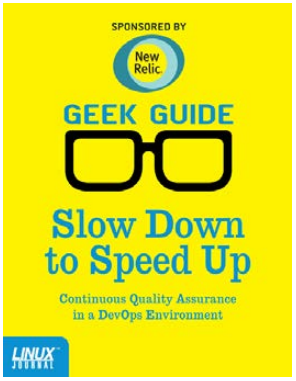
NEW!

Linux Journal eBook Series

GEEK GUIDES

FREE Download NOW!

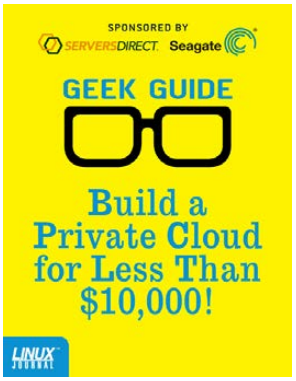
Slow Down to Speed Up:
Continuous Quality Assurance in a DevOps Environment



By Bill Childers

DevOps is one of the newest and largest movements in Information Technology in the past few years. The name DevOps is a portmanteau of "Development" and "Operations" and is meant to denote a fusion of these two functions in a company. Whether or not your business actually does combine the two functions, the lessons and tools learned from the DevOps movement and attitude can be applied throughout the entire Information Technology space. This eBook focuses on one of the key attributes of the DevOps movement: Quality Assurance. At any point, you should be able to release your product, code or configuration—so long as you continue keeping your deliverables in a deployable state. This is done by "slowing down" to include a Quality Assurance step at each point in your workflow. The sooner you catch an error or trouble condition and fix it, the faster you can get back on track. This will lower the amount of rework required and keep your team's momentum going in a forward direction, enabling your group to move on to new projects and challenges.

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



By Mike Diehl

This eBook presents a compelling argument as to why you should consider re-architecting your enterprise toward a private cloud. It outlines some of the design considerations that you need to be aware of before implementing your own private cloud, and it describes using the DevCloud installer in order to install OpenStack on an Ubuntu 14 server. Finally, this eBook will familiarize you with the features and day-to-day operations of an OpenStack-based private cloud architecture, all for less than $10K!

DOWNLOAD NOW AT: http://linuxjournal.com/geekguides

DRUPAGEDDON:

SQL INJECTION, DATABASE ABSTRACTION AND HUNDREDS OF THOUSANDS OF WEB SITES

An introduction to SQL injections and their impact on a popular content management system, and recommendations to mitigate SQL injection attacks against Web applications.

SHEA NANGLE

DRUPAL is a very widely used open-source content management system. It initially was released in 2001, and recent statistics show Drupal as the third-most popular content management system, with just less than 800,000 Web sites utilizing Drupal as a content management system.

Drupal is written in PHP, and it is architected to use a database back end to store Web site content and settings, whether a full-fledged database management system (such as MySQL) or an embedded DBMS (such as SQLite). In recent versions, Drupal has provided a database abstraction layer in order to facilitate the use of any of a number of database management systems to support a given Drupal installation. Database abstraction layers provide a consistent programming interface that can be used to communicate with a variety of database systems without development of code specific to a given database management system.

Due to vulnerabilities in the database abstraction layer introduced in version 7 of Drupal, Drupal 7 prior to version 7.32 was vulnerable to an SQL injection attack. This article provides an introduction to SQL injection attacks, an examination of the Drupageddon vulnerability specifically and an explanation of a number of potential defenses against SQL injection attacks.

SQL Injection

SQL injection is an attack methodology in which malicious SQL code is included in user input, leading to the execution of said SQL code as part of SQL statements used by an application. SQL injection attacks can lead to privilege bypass and/or escalation, disclosure of confidential information and corruption of database information, among other effects.

Command injection attacks, such as SQL injection, routinely place at or near the top of the OWASP (Open Web Application Security Project) Top Ten List of Web application security risks. SQL injection attacks are likely the most well-known type of command injection attacks, but injection attacks can occur any time data is supplied to an interpreter by an application. The recent Bash vulnerability known as Shellshock is an example of a command injection attack that is not related to SQL injection.

SQL Injection Example

An example SQL injection attack starts with code utilizing an SQL statement,

such as:

```
$db_statement = "SELECT COUNT(1) FROM `users` WHERE
➥`username` = '$username' AND `password` ='$password'";
```

In an SQL injection attack against code such as this, the attacker supplies input, such as the following, to the application:

```
$username = "badUser";
$password = "' OR '1'='1";
```

Using this example, the SQL statement executed becomes the following:

```
SELECT COUNT (1) FROM `users` WHERE `username`='badUser'
➥AND `password`='' OR '1'='1';
```

In the above example, this results in returning a count of all rows in the "users" table, regardless of the user name or password supplied, since the conditional `'1'='1'` always returns as true. If the query shown in this example is used for authentication purposes, the example SQL injection attack has just bypassed the authentication process for the application in question.

SQL injection attacks, and other command injection attacks in general, represent a significant risk for Web applications. Exploitation of SQL injection vulnerabilities is relatively easy for an attacker to perform, and both the attack itself and searches for vulnerable code are easily automated. Additionally, the impact of SQL injection attacks is quite often very severe, as is seen in the authentication example above, as well as in the specific example of Drupageddon.

Drupageddon

The Drupageddon vulnerability officially was discovered by SektionEins GmbH while the company was performing a security audit for a customer that was utilizing Drupal as a content management system. SektionEins GmbH reported the vulnerability to Drupal developers on September 16, 2014. The vulnerability was disclosed publicly by Drupal on October 15, 2014, using the Drupal advisory identifier DRUPAL-SA-CORE-2014-005 and the CVE identifier CVE-2014-3704. The public disclosure included a description of the vulnerability, as well as recommendations for vulnerability mitigation. The primary recommendation for mitigation of this vulnerability was an immediate upgrade to Drupal version 7.32. For administrators of Web sites that could not be upgraded to Drupal version 7.32 immediately, a patch

was provided to resolve the SQL injection vulnerability.

SektionEins nicknamed this vulnerability Drupageddon due to the potential impact of exploitation of this vulnerability upon Drupal-based Web sites. (Note: in a number of instances in the press, this vulnerability was referred to as "Drupalgeddon", which is inaccurate. The term "Drupalgeddon" refers to a diagnostic tool intended to be used in order to diagnose Drupal instances that may have been compromised due to the Drupageddon vulnerability.)

Successful exploitation of this attack could result in execution of arbitrary PHP commands, privilege escalation, installation of system backdoors and other exploits. Additionally, exploitation of this vulnerability did not require any sort of successful authentication to the target Drupal instance(s) prior to exploitation.

It was estimated that within several hours of the announcement of the Drupageddon vulnerability, active and automated exploits for this vulnerability were being utilized by attackers to compromise Drupal-based Web sites.

On October 29, 2014, the Drupal Security Team released advisory identifier DRUPAL-PSA-2014-003. This advisory informed administrators of Drupal-based Web sites that all Drupal-based Web sites utilizing vulnerable versions of Drupal should be considered compromised if they were not patched/upgraded before 2300 UTC on October 15, 2014 (seven hours following the initial announcement of the vulnerability in SA-CORE-2014-005).

In the case of the Drupageddon vulnerability, the database abstraction layer provided by Drupal included a function called `expandArguments` that was used in order to expand arrays that provide arguments to SQL queries utilized in supporting the Drupal installation. Due to the way this function was written, supplying an array with keys (rather than an array with no keys) as input to the function could be used in order to perform an SQL injection attack.

A potential (non-malicious) use of the `expandArguments` function would be as follows:

```
$query = "SELECT COUNT(1) FROM `users` WHERE `id` IN (:userids)";
$args = [ 'userids' => [ 1, 2, 3, ] ];
$db->expandArguments($query, $args);
```

This would result in the following SQL statement:

```
SELECT COUNT(1) FROM `users` WHERE `id` IN
➥(:userids_0, :userids_1, :userids_2);
```

However, by supplying a carefully crafted argument array, an attacker could perform an SQL injection attack:

```
$query = "SELECT COUNT(1) FROM `users` WHERE `id`
 ➥IN (:userids)";
$args = [ 'userids' => [ '0); DROP TABLE
 ➥importantInformation; --' => 1 ], ];
$db->expandArguments($query, $args);
```

This would result in the following SQL statement:

```
SELECT COUNT (1) FROM `users` WHERE `id` IN (:userids_0);
 ➥DROP TABLE importantInformation; --)
```

The -- marks the remainder of the line as an SQL comment, avoiding the syntax error due to the unmatched right parenthesis. The results of the execution of a malicious query such as this obviously could be catastrophic.

Recommendations

A number of strategies can be used to minimize the risk of SQL command injection attacks. These include input sanitization and whitelisting, use of parameterized queries and defense in depth.

Input Sanitization and Whitelisting: One strategy that can be used for prevention of SQL injection attacks is the sanitization and whitelisting of user input. This strategy is implemented by analyzing both expected or valid input that will be provided by users as well as input that may be provided by attackers attempting to compromise your Web-based application. Following this initial analysis, sanitization code will need to be added in order to remove or escape any harmful/unwanted input by a user prior to use of said input in any database queries. In the case of the SQL injection example given earlier in this article, there are two potential sanitization and whitelisting processes that could be utilized.

In the SQL injection example given earlier, let's assume you previously have told users of the Web application that valid characters for user names are a–z, A–Z, 0–9 and ".". This would represent an excellent opportunity for the use of whitelist-based input validation. In this method of input validation, you would construct a whitelist of allowed characters for the user input and would allow only user input limited to those characters to be passed to the database for processing. In this case, you either would discard any characters provided by users as their user name aside from a–z, A–Z, 0–9 and ".", or you simply would refuse to perform any processing following user input that includes any

characters that are not included in your whitelist of allowed characters. Using this example, an attack in which the following is provided as input for the `username` value:

```
$username = "x'; DROP TABLE importantInformation;
➥SELECT * FROM users WHERE username = 'badUser'";
$password = "Test";
```

either would be refused or would be sanitized (if inappropriate characters are discarded) to the following:

```
$username = "xDROPTABLEimportantInformation
➥SELECTFROMusersWHEREusernamebadUser";
```

If sanitization is used in this example, this would result in the following SQL statement being executed:

```
SELECT COUNT(1) FROM `users` WHERE `username`=
➥'xDROPTABLEimportantInformationSELECTFROMusersWHEREusernamebadUser'
➥AND `password`='Test';
```

Without sanitization, the following SQL statements would be executed:

```
SELECT COUNT(1) FROM users WHERE username='$username =
➥"x'; DROP TABLE importantInformation; SELECT COUNT(1)
➥FROM users WHERE username = 'badUser' AND password = Test';
```

This results in two `SELECT` statements being executed, and also results in the deletion of the `importantInformation` table.

(Note: this SQL example also represents other security problems in that passwords in the database appear to be stored in plain text. However, that is outside the scope of this article, and encryption of passwords would have no impact on vulnerability to SQL injection attacks.)

In the user authentication example, additional processing is needed in order to handle the password provided by the user, however. Assuming that you allow all keyboard characters in an effort to allow for as complex passwords as possible, you cannot simply refuse password input containing potentially dangerous characters. In this case, you will want to sanitize user input by escaping said input prior to query processing. In this example, the following input:

```
$username = "badUser";
$password = "' OR '1'='1";
```

would become escaped to the following:

```
$username = "badUser";
$password = "\' OR \'1\'=\'1";
```

This then would result in the following

At a very high level, the best plan for preventing SQL injection attacks is an overall strategy of defense in depth.

SQL statement being executed:

```
SELECT COUNT (1) FROM users WHERE username='badUser'
➥AND password='\' OR \'1\'=\'1';
```

This version of the SQL statement would result in returning a count of the number of rows in the users table where the contents of the user name field is equal to the string "badUser", and the contents of the password field are equal to the literal string "' OR '1'='1" (that is, the attack has been blocked).

Use of Parameterized Queries

Another strategy for guarding against SQL injection is the use of parameterized queries. With parameterized queries, SQL statements are predefined and stored on the database server with placeholders representing the parameters that will be utilized in the query. When it comes time for the SQL statement(s) in question to be executed, relevant user input is added to the queries prior to execution, with any relevant escaping of user input being handled automatically by the database server.

In the user authentication example shown previously, the parameterized version of the query would resemble something like the following:

```
SELECT * FROM users WHERE username=?un? AND password=?pw?;
```

When it comes time for the SQL statement to be executed, the database management system performs any escaping needed for the parameter(s) in question, and the SQL statement then is executed with the escaped parameter(s) taking the place of the placeholders.

Defense in Depth: At a very high level, the best plan for preventing SQL injection attacks is an overall strategy of defense in depth. This approach relies on deploying a number of different defense mechanisms simultaneously, with the overall strategy being that if an attacker is able to defeat one of the defense mechanisms, the other defense mechanisms still will be in place and still will be able to defend against/detect attempted attacks. In the example of Drupageddon, the following parallel defense strategies in addition to the previously mentioned defense

strategies would have helped to minimize the risk of system compromise due to SQL injection.

- *System and Application Updates:* Keeping systems and applications current with updates is one of the first lines of defense that should be implemented by individuals and organizations in order to prevent system and application compromises. In the case of Drupageddon, either upgrading to Drupal version 7.32 or installing the patch provided by Drupal developers would have mitigated against the Drupageddon vulnerability immediately. Although the Drupageddon vulnerability existed since 2011, there is no evidence of any significant exploitation of the vulnerability until after the public disclosure of Drupageddon by SektionEins.

- *Intrusion Detection Systems:* Most current intrusion detection systems include functionality to detect attempted SQL injection attacks. These systems can provide early warnings of attempted SQL injection attacks, and if paired with intrusion prevention functionality, often can prevent the attacks from occurring.

- *Limitation of Database Privileges:* Privileges of database users used for applications should be limited to as restrictive a set of privileges as possible that will allow for the performance of required database activities in order to support the functionality of the application. For instance, if the only database activities that are required for the application in question are reads from the database, consider limiting the database account used by the application to a read-only account. This will not prevent SQL injection attacks aimed at inappropriate access to information, but it will prevent SQL injection attacks that are intended to cause unauthorized changes to the database.

- *System and Application Monitoring:* Although system and application logging and monitoring will not, in and of themselves, prevent an SQL injection attack from occurring, they will help in the process of detecting attempted attacks. Additionally, use of functionality like file integrity monitoring can help detect the results of system compromises due to SQL injection attacks.

- *Code Audits:* Auditing of source code utilized in an application can help identify security vulnerabilities in the application in question. In fact, the Drupageddon vulnerability was discovered due to a source code audit that was performed against the Drupal code base. Source code audits typically should be performed by a party other than those responsible for the creation of the source code in question, as this provides a fresh perspective on the source code. Unfortunately, source code audits usually are very expensive and time-consuming.■

Shea Nangle is an information security engineer with BlackMesh. His areas of interest include compliance, log analysis and open-source intelligence. In his spare time, you often can find him cycling or homebrewing mead.

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

Resources

Usage of Content Management Systems for Web Sites: **http://w3techs.com/technologies/overview/content_management/all**

Drupal Usage Statistics: **http://trends.builtwith.com/cms/Drupal**

OWASP Top Ten List: **https://www.owasp.org/index.php/Top_10_2013-Top_10**

SA-CORE-2014-005 — Drupal Core — SQL Injection: **https://www.drupal.org/SA-CORE-2014-005**

Drupal Core — Highly Critical — Public Service announcement — PSA-2014-003: **https://www.drupal.org/PSA-2014-003**

Drupalgeddon: **https://www.drupal.org/project/drupalgeddon**

Advisory 01/2014: Drupal — Pre Auth SQL Injection Vulnerability: **https://www.sektioneins.de/en/advisories/advisory-012014-drupal-pre-auth-sql-injection-vulnerability.html**

Drupal 7.31 Pre Auth SQL Injection Vulnerability: **https://www.sektioneins.de/en/blog/14-10-15-drupal-sql-injection-vulnerability.html**

Drupal 7.32 Two Weeks Later — PoC: **https://www.sektioneins.de/en/blog/14-11-03-drupal-sql-injection-vulnerability-PoC.html**

LinuxFest Northwest

April 25th & 26th
Bellingham, WA

Bellingham TECHNICAL COLLEGE

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

Introducing DevAssistant

DevAssistant is a tool that helps you easily kick off development, automate mundane tasks and publish code without hassle.

TOMAS RADEJ

If you ever have started writing a new piece of software or checked out someone else's, you certainly know there's a lot more to be done than just opening the file and starting coding. You need dependencies. You need the proper directory structure. You need the proper things put on your $PATH and countless others considerations. To seasoned developers, these are issues that only somewhat slow them down; for new people, these often are obstacles that make them give up and try something else altogether. A few of my team members and I decided we wanted to fix the problem for both of these kinds of developers. Easier said than done, you think? It depends. About two years ago, we started discussing concrete ways of solving these small, frequent and annoying problems, and we've come up with something that gradually became DevAssistant (usually shortened to DA) today (http://www.devassistant.org).

What we created is, essentially, a simple but powerful tool that sets up your entire environment for developing a particular software project with only a few clicks in a GUI or a single line on the command line, and that you can extend almost without effort. DevAssistant consists basically of three things: the main program, which is packaged under the name devassistant on the Python Package Index and in Fedora; the scripts called Assistants that implement the particular workflows; and the DevAssistant Package Index (DAPI), which is where you install the Assistants from.

What Is It Actually Good For?

This is usually where people start asking questions. Truly, at first glance, DevAssistant may seem redundant. After all, there's support for most of these things in IDEs, so that's sorted out already, right? And dependencies are specified in the package management software for that particular language, aren't they?

Technically, yes. Actually, no. The main problem with IDEs is the very fact that you actually have to use them, and by "them", I mean only the specific one that supports what you need. Every single IDE has a ton of neat features, but what about users of different IDEs? When we designed DevAssistant, we deliberately wanted to avoid this kind of dependence on a particular

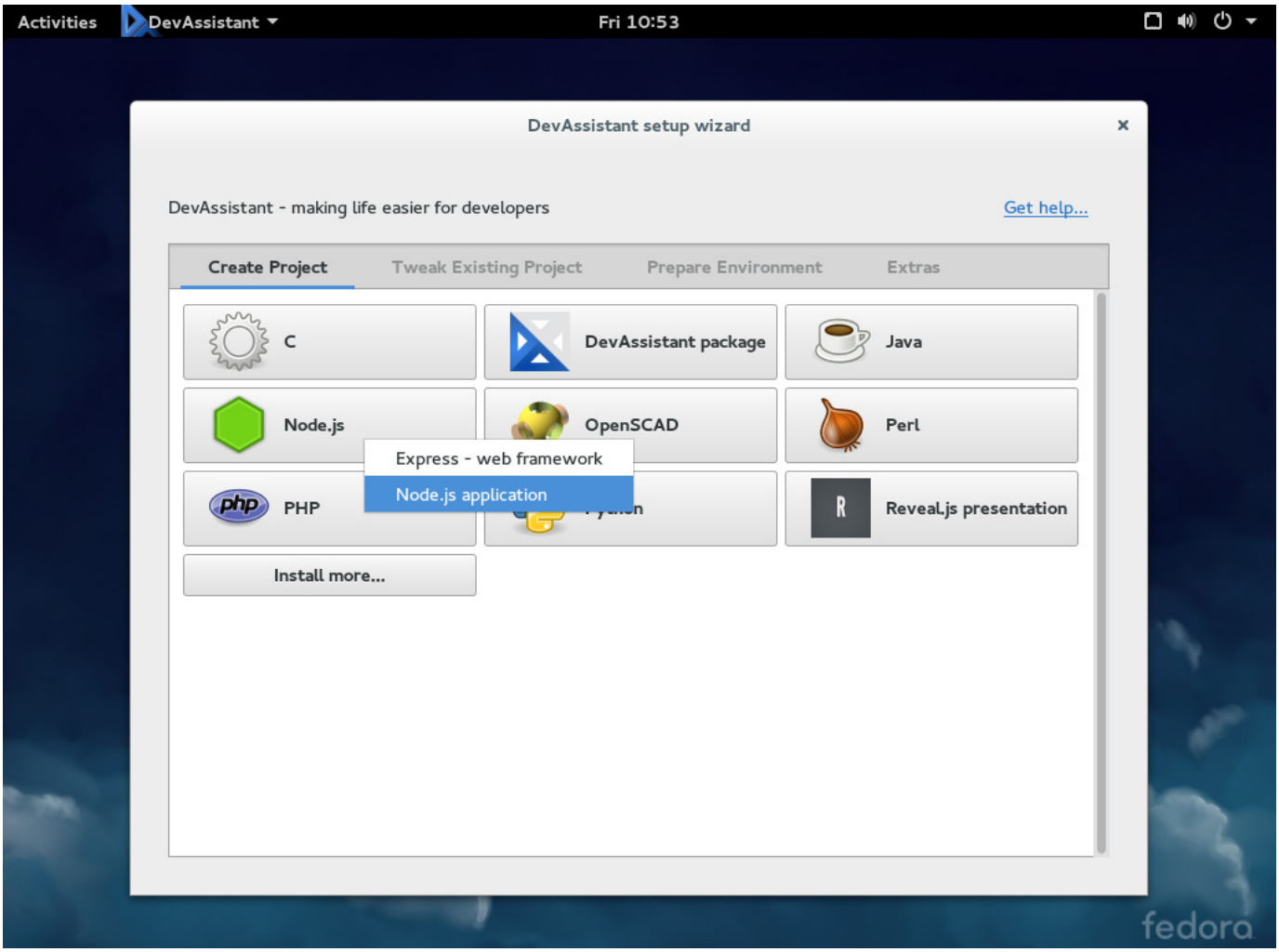


Figure 1. DevAssistant's main window—you can select which Assistant to run from here (Node.js selected, cursor not visible).

development suite, so that we are able to cater to all users, regardless of whether they use a particular IDE, a standalone text editor or write machine code into standard input.

The other major objection we usually get is that there are dependency mechanisms already well established across languages, and there's no reason to have another. That assertion is almost, yet not exactly, true. There certainly are mechanisms to manage dependencies that are necessary to run the program or use the library. On the other hand, there is usually little that can be done if you want to make sure that you have everything needed to develop the code. And even if there is, it rarely takes care of settings other than dependencies. What if you need a working Web server to test your app, open a port in a firewall or set up SELinux? None of the established systems takes care of such trouble for you. Furthermore, nobody solves the dependencies for you if you're not really writing a program—like writing documentation in Markdown and wanting to see how your text will look when converted to HTML or another language.

In addition to all this, there is the question of publishing your code. Since you're using free/open-source software, we bet that you want to publish your code as well. To make this easier for you, DevAssistant is fairly tightly integrated with GitHub, so if you want to create a new repository and push your sources there, you can do so without even opening your browser. The same goes for forking existing repositories. Conventionally, it means this: going into your browser, logging in, typing the repo's full address, clicking "Fork", waiting for the forking to finish, selecting the address for cloning, then going into your console and typing `git clone` and pasting the address. With DevAssistant (and the Assistant called "custom"), you run this:

```
da prepare custom --gh-repo olduser/repo --gh-fork newuser
```

DevAssistant then performs all the steps mentioned above, in a single command. You can do the exact same thing in the GUI with very few clicks as well. More specifically, the GUI and the command-line interface both share the same Assistants that contain instructions and dependencies specifications, and that are analyzed and carried out by the main program.

The previous four paragraphs possibly made you think "Wouldn't it be better if beginners learned by

setting up everything in the first place, instead of having DA do it all for them?" Of course it would. However, ask yourself honestly, when you were starting whatever you are doing now, did you dig deep into each step of the installation instructions and analyze it thoroughly before moving on? Or did you just stumble your way through it, having a vague idea of what the instructions meant, impatient to get to the "Hello world!" example at the end? We are willing to bet that nine out of ten cases were the latter, and most developers learned only much later how the language worked. What our team did was simply build a tool to lower the learning curve for these developers.

Assistants and Workflows

Let's talk about what you can automate through various Assistants. Be aware that at the time of writing of this article, there are two supported versions of DevAssistant: 0.9.3 and 0.10.0. The first one is a bug-fix release, which still includes some Assistants in its core distribution and is available in Fedora 21 by default. To install Assistants for this version, you need to download and unpack the packages manually (more on that later). In version 0.10.0, no Assistants are included, and they can be installed via the command line by running:

```
da pkg install NAME_OF_PACKAGE
```

With Assistants installed, you can run them either from the command-line interface (CLI) or the GUI. In the CLI, an invocation looks like this (the example works with the Assistant called "python", available on the DAPI):

```
da create python flask -n my_flask_app
```

The same can be achieved in the GUI by selecting Create project→Python→Flask and typing "my_flask_app" as the project name. Running this Assistant creates a new Flask Web application named "my_flask_app", installs all necessary dependencies, creates the required directory structure in a directory named like the app itself, and modifies the boilerplate files to conform to your project (for example, by changing paths, names and settings). Once DA's run is finished, you then can just run:

```
./manage.py runserver
```

in the new project's root directory, and you'll have a running Web app only seconds after launching DA.

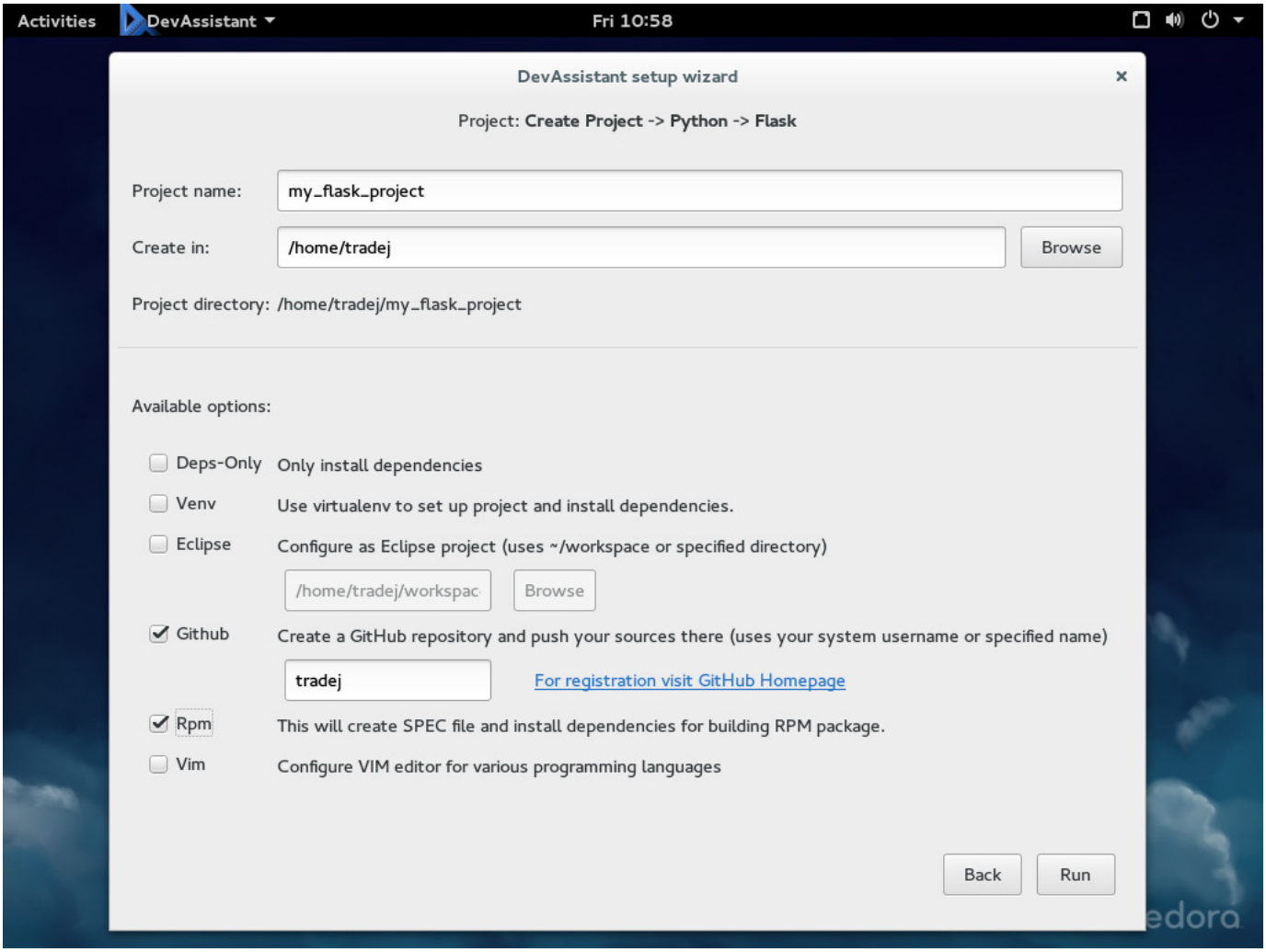


Figure 2. Project creation window—here is where you specify parameters for the project you want to create.

During the run, basic information is logged either to the standard output or the log window (depending on which interface you are using), so you can keep track of what is being done. If you prefer more verbose feedback or need to figure out what the problem is if something went wrong, you can select the debug option and see all the information that either DevAssistant itself or the shell underneath it outputs.

Generally speaking, there are four main kinds of Assistants: create, tweak, prepare and extras. This is the naming used in version 0.10.0 and all future versions; older releases of DevAssistant use "modify" instead of "tweak" and "task" instead of "extras". Each of the Assistant types corresponds to the action you want to perform. Some Assistants help you

create new projects from scratch. Some tweak existing code by adding new headers or generating new files from existing ones. The extras (or "task", in version 0.9.3) Assistants are used for all other workflows that are not covered by the previously mentioned categories.

The prepare type is a bit special, as it is used for preparing the environment for a specific project, not just a generic one. That means that if you create a Web application in Django, add some dependencies, have some specific database settings that work just for your app and don't make sense for other Django projects, you can write a prepare Assistant that will set up users' computers for them. Users then can develop your application without making any further configuration changes. This is especially helpful in cases where the project you created is very complex and you want to make it easy for people to submit patches to you. Consider that if a patch takes technically adept users five minutes to write, and they have to spend an hour setting up their environment just to verify that their patch works, they simply won't write it. With DevAssistant, they spend maybe a minute installing a prepare Assistant for your app and five minutes writing that patch. Everybody wins.

Out of the box, DevAssistant version 0.9.3 supports many popular languages including Python, Java, C(++), PHP, Ruby, Node.js or Perl. For most of these languages, sub-Assistants exist for particular frameworks or apps. So under Python, you can find both the general Python library support as well as Django. If necessary, more Assistants implementing Python workflows easily can be added without affecting the existing ones. Both the command-line arguments and the GUI screens then are generated automatically. As a side note, DA comes with bash completion in Fedora, which also is automatically generated right until the last parameter. All the mentioned Assistants are available on the DAPI for installation into DevAssistant 0.10.0 as well.

Docker

As you may have noticed, all I've mentioned up until now was happening on the user's machine, on bare metal. At this very moment, we are working on having the support for DevAssistant workflows in Docker containers as well, which means that once you set up a development environment in Docker with DA, you easily can share it with

your teammates or just publish the Dockerfile for anyone to rebuild. Note that at the moment, DA is not the tool to deploy that very Docker image in production, but we may add that functionality in the future.

DevAssistant Package Index

As previously mentioned, one of the key parts of the DevAssistant ecosystem is the DevAssistant Package Index, or DAPI. This is a server for developers to upload their Assistants in packages, which are called DAPs (DAP is short for DevAssistant Package). It's worth mentioning that this mechanism is quite new (made for use with DevAssistant version 0.10.0), so more content and features are to come. In the aforementioned version 0.10.0, users can install these packages directly with DA itself by issuing the command:

```
da pkg install FOO BAR
```

Running this particular command would install the packages FOO and BAR with all the DAPs on which these two depend. Similarly, users can un-install and search directly from DevAssistant too. At the moment, this functionality is available only in the command-line interface, but version 1.0 of DevAssistant will feature a re-designed GUI with the same features in a "clickable" form as well.

If you are using version 0.9.3 of DevAssistant and want to install a new Assistant, you need to do so yourself. First, you need to download the DAP file, presumably from the DAPI Web interface found at **https://dapi.devassistant.org**. Once downloaded, you must unpack the file (it is technically a gzipped tarball) into ~/.devassistant while stripping the topmost directory. On the command line, this is done by running:

```
tar xzf DAP_NAME --strip-components=1
```

In a graphical archive tool, you just select the second-level directories (assistants, snippets, icons and so on) and extract those. Furthermore, it is necessary to rename the directories "twk" or "extra" to "mod" or "task", respectively, because of the older layout used by version 0.9.3. The reason for the apparent lack of user-friendliness is that due to significant architectural changes between versions 0.9.3 and 0.10.0, it takes a disproportionate amount of effort to backport the package installation mechanism from 0.10.0 to the previous version, and we'd rather spend it on fixing the new version and implementing new

features for you. For users of Fedora 21 who want to benefit from the DAPI integration, we recommend using the COPR repository located at **https://copr.fedoraproject.org/coprs/tradej/DevAssistant**, where version 0.10.0 is packaged.

Developers who want to upload a package to the DAPI need to log in either with their GitHub account or Fedora's FAS account. Once uploaded, the DAP is tested for validity, and if it passes, saved to the repository. Obviously, this means there soon will be hundreds and hundreds of DAPs in the index, and users will have to make sense of them somehow. We have taken that into consideration, giving users the ability to search by tags (filled in by the authors of DAPs) and by quality, represented by a simple one-to-five-star system (as voted on by logged-in users). There is, of course, a mechanism to report malicious packages as well.

How Assistants Work

So far, I have been talking mainly about the user perspective only. If you don't want to delve deeper in the technical side of things, feel free to skip this section.

On the inside, DevAssistant consists of the core, which is written in Python, and the Assistants accompanied by auxiliary files (which are usually copied to the final project's directory structure). The Assistants are written in a domain-specific language based on YAML. We are expecting quite a few eyebrows will raise right now. We are perfectly aware of the ramifications of this selection, and we didn't make it by rolling dice or doing an eenie-meenie. Instead, we have done our research, and although it wasn't the obvious or even the first choice, we settled with YAML due to a number of reasons.

First and foremost, it isn't associated with any other widely used language. In the past, we repeatedly got a lot of negative feedback when deciding to support only one particular language for writing modules for programs. People using interpreted languages don't like compiled. Users of static typing refuse to use dynamic, and so and so on. For this reason, a markup language was the obvious choice. We didn't select simpler formats like INI due to technical constraints, and we didn't select XML because we think that the scripts should be written in something that is readable by humans. YAML, a mature enough language used widely in systems orchestration, which is fairly close to what DevAssistant does,

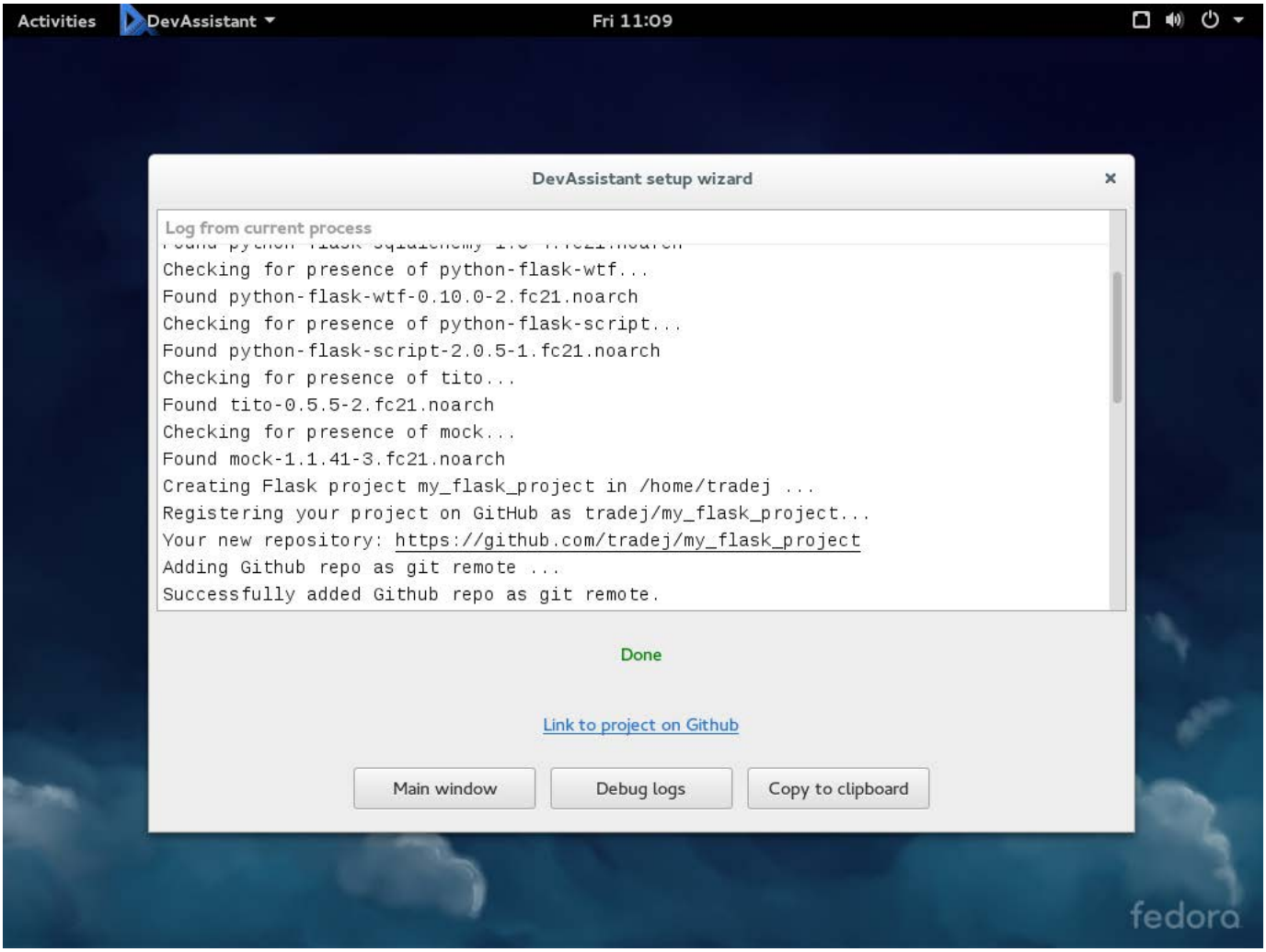


Figure 3. A log window with a finished successful run—when you see this, your project is ready.

emerged as a victor of this challenge.

However, for those who do not want to write their whole script in YAML and prefer to use their language of choice, there is a nice surprise in version 0.10.0. You guessed it, DevAssistant has an API for running scripts in other languages. For each language that you want to use for running an Assistant, you need bindings for the DevAssistant PingPong protocol (DAPP). At the moment, these bindings are available for Python only in the form of a reference implementation. When using the bindings, you need a minimal YAML Assistant file to specify dependencies, arguments and the script, which then does all the heavy lifting itself.

The content of the YAML-only Assistant is fairly simple. There are metadata like name and description,

declarative dependencies, argument and files sections, and an imperative run section that contains the actual commands and shell calls that perform the Assistant's task. Assistants are inheritable and support an inclusion mechanism; therefore, you easily can do things like support Eclipse integration with all your Java projects or add a specific Vim config to only a select few. The invocation of DAPP-enabled scripts is done in the run section too, and it requires only the script to be specified in the "Files" section.

As far as dependencies go, we tried to make the mechanism as bulletproof as possible, which means that you can specify dependencies for a particular packaging format (RPM, pip, pacman and so on) in a descending order. So if DA cannot use a particular package manager, it moves on until it finds one that is available or runs out of package managers altogether. To answer a question we get quite often, right now we do not support versioned dependencies due to the fact that it is difficult to make sure that the underlying package managers report accurate information to DA, but we're in the process of finding alternative solutions.

You can read further descriptions of the Assistants and related concepts at **http://docs.devassistant.org**.

Going Further

At the moment, we have several initiatives going on, which largely will define where DevAssistant goes from here. Apart from the aforementioned Docker container support across Assistants, we aim to improve support of certain project management workflows that take place after the project has been created. However, this support is rather specific because we do want to avoid duplicating the IDE functionality as much as possible.

On the integration level, DevAssistant is a part of a larger ongoing effort to make Fedora Linux more accessible and attractive to developers, so although DA does run on other Linux distros (and, mind you, even runs on Mac OS), integration with Fedora is our main focus. That means, among others things, automatically creating RPM SPEC files for your projects, building on the Fedora infrastructure (especially COPR), possibly deploying on FedoraHosted and the like.

In version 1.0, DA will have a new, more user-friendly graphical interface, which also will support installing, removing and looking up Assistants directly within the application. We still aren't quite sure about some aspects of the GUI, so expect more to come.

Give It a Try

If you want to try DevAssistant, you can get it in several ways. For one, it is present in the Fedora 21 Workstation installation by default (that is true for version 0.9.3). This way, you get seamless installation, bash completion and automatic updates as part of the system update routine. Version 0.10.0 for this version of Fedora is accessible as a COPR repository (for installation instructions, see the Download section of the DevAssistant home page: **https://devassistant.org/download**). Another option is downloading DevAssistant from the Python Package Index. If your system is running pip, you need to run:

```
pip install devassistant
```

Bear in mind, though, that for some dependent Python libraries, you will need certain underlying C header files, so the installation may not be straightforward, and the program may not behave the way it is intended to. It generally should be safer to use the Python 3 version of DA, installed by invoking the Python 3 variant of pip. Still, we can't tell if it will work as expected on your system. The last option is downloading the source code from GitHub, which also is a perfectly valid way to go, especially if you intend to contribute to DevAssistant.

We hope this article makes DevAssistant interesting to you, and that you will check it out (if you pardon the pun). Both DA and DAPI are free and open source (licensed under GPL v2+ and AGPL v3+, respectively). We're eager to get more feedback, because DevAssistant is mainly about helping you, and we can't do that if we don't know what you need as precisely as possible—whether it's a message sent to the mailing list, an IRC question, a bug report or a pull request, we are grateful for all that you tell us. Let DevAssistant help you start developing with ease.■

Tomas Radej is a software engineer at Red Hat, born and raised Czech, alumnus of the FI MUNI in Brno, and fundamentally a 1990s kid. A large part of his work time consists of DevAssistant, Fedora, Python and FOSS in general. He has had an on-off relationship with free software since he was about 15, finally committing halfway through his university studies and joining Red Hat. As for pastimes, he likes literature, politics and all things legal. If you want to meet him, there's a good chance you'll bump into him at some FOSS conference.

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

DOC SEARLS

You're the Boss with UBOS

A new Linux distro for everybody's everythings.

UBOS is a new Linux distro that I like for two reasons. One is that it works toward making it easy for muggles to set up their own fully independent personal home servers with little or no help from wizards. The other is that it comes from my friend Johannes Ernst.

It's in beta at the time of this writing, and it runs on PCs, Raspberry Pi models B and B+, and on Macs in VirtualBox. It's been tested with ownCloud, Known, WordPress, Mediawiki, Selfoss, Shaarli and Jenkins and plugins for a number of those.

Says Johannes:

> My goal is to make the administration of personal servers 10x easier for users, and also to make it much easier for developers to create "personal data" Web applications that don't spy on us and get them deployed. Over time, we will get UBOS to as many pieces of hardware as possible, and pre-install lots of "personal data"-related middleware. We're already working with two Internet of Things projects to get UBOS to be the OS they run as the default on their hardware. Imagine if all the IoT products you bought were "indie" and not tied to some corporate overlord's take-over-the-world strategy? To do that, we need to make administration easier.

Here are some of the ways that's done:

- **Single-command deployment of Web apps**, with automatic database provisioning, Web server configuration and so on—including SSL setup.
- **Full virtual hosting**—for example, you can run two instances of

WordPress with different plugins and one of ownCloud at http://personal.example.com/blog, http://home.example.com/news and http://home.example.net/owncloud on the same host.

- **Single-command undeployment**.
- **Single-command full system upgrade**, which backs up all your data, upgrades all code from the operating system over middleware to applications, runs whatever data migrations might be necessary and redeploys all your apps.
- **Single-command backup and restore** of all or part of the apps installed on the same host.

Phil Windley is the creator of picos (persistent compute objects, or virtual things) and CloudOS (an operating system for picos, the subject of "The First Personal Platform—for Everything" in our October 2013 issue), and my guru on what I'll call the Internet of Personal Things. When I asked him for his take on UBOS, he wrote this back:

> There's a healthy sorting out happening in the "personal cloud" space, around devices, sensors and other "things" that people will use—and around personal control and ownership as well. Picos/CloudOS are about providing specific services in a lightweight way for things that are all under a user's control. Picos allow personal data to remain personal. So the first way picos and systems like UBOS are related is in the heads of people realizing they need to be in control of everything in their life: words, personal data, things or whatever.

More technically, UBOS and picos could interact in the following ways:

1) You could host your picos on UBOS on your own server.

2) You could monitor and control them from such a platform.

3) UBOS would be a great place to run your personal (UMA-based?) authorization server.

To unpack this the rest of the way, I interviewed Johannes.

DS: According to DistroWatch, there are hundreds of Linux distros already. Why one more?

JE: There is a hole in the market, and it's at home. UBOS focuses on making the administration of home servers much simpler, which is not something most distros pay much attention to.

The timing is also right. First, hardware has become cheap and powerful enough to run Web servers at home. Many people now keep their calendars, personal photos and other important private data away from the major cloud providers, because they would rather store it on hardware they have control over. Second, developers of cool IoT projects almost always need a Web server, so they can open their garage door, set their thermostat or water their plants over the Web. While many of those are $35 Raspberry Pis, they still require the same amount of maintenance as any big Linux server used for business purposes.

With UBOS, we set out to make administration of these kinds of personally owned servers at least 10x faster and easier, so more people can run their own servers at home, build more cool IoT projects and keep their data private and at home, without turning home server maintenance into a full-time job.

DS: "10 times" faster system administration is a big claim. How do you back this up?

JE: Think of what you need to do to install a Web app, such as ownCloud, on your server. This usually involves:

1. Finding and downloading a tar file.

2. Full virtual hosting. For example, you can run two instances of WordPress with different plugins and one of ownCloud at http://personal.example.com/blog, http://home.example.com/news

and http://home.example.com/owncloud.

3. Provisioning a MySQL database.

4. Editing application configuration files and fixing permissions.

5. Setting up logging.

6. Restarting servers and so on. Installation of a single app can easily take a few hours and usually requires real Linux admin knowledge. We think it should be easier, so more people can do it, faster, and more reliably.

In UBOS, you say `sudo ubos-admin createsite`. UBOS will ask you a few questions, like the desired virtual hostname or whether you want SSL, and then UBOS does all these steps for you. Total time consumed: maybe a minute.

This already works for a number of Web apps, including ownCloud, WordPress, Mediawiki, Selfoss, Shaarli, Known and even Jenkins where UBOS automatically sets up an Apache reverse proxy to Tomcat.

When it is time to upgrade, server administration is even more time-consuming, and don't we all want to keep our Web sites patched for security reasons! You need to do all of the above, but also back up data before (don't forget any place the app might have written valuable files!), and perhaps manually run data migrations. Some apps like WordPress have made it easy, but most have not. In UBOS, it's simply: `sudo ubos-admin update`. And you can relax while UBOS does all of the above, for all the apps you have installed at all of the virtual hosts on your personal server.

DS: What changes did you need to make to the Web apps you list so they could be administered by UBOS?

JE: So far, we haven't had to fork any of them, so it's been pleasantly simple. It's mostly adding some JSON metadata to the application package and parameterizing configuration files so UBOS can "edit" them correctly. Sometimes some additional scripts are required, but rarely. All the current apps are on GitHub at **https://github.com/indiebox**.

DS: Wasn't one of the existing distros close to being what you want to give the world with UBOS?

JE: UBOS is a derivative of Arch Linux,

which we rely on heavily. You can think of UBOS as a subset of Arch, plus UBOS administration tools and the additional metadata and conventions required to make ubos-admin work.

We inherit many of the great qualities of Arch—for example, it is a rolling release distro, so we never have to impose major upgrades on the user; packages on Arch are never very much out of date, because we do want to run the latest and greatest, and the Arch community and its wiki are simply amazing. UBOS takes many of the Arch packages even without recompiling.

We started out by simply adding our administration packages to Arch, but because we want to make sure that apps always install correctly and upgrade correctly, we need to control when upgraded packages are made available to servers. So it was the needs of the quality assurance process that caused us to create a new distro with separate repositories.

DS: How, besides this interview, are you going to attract hackers to jump in and improve the code?

JE: We just published the very first UBOS beta, for x86_64 and the Raspberry Pi, so we are only at the beginning of this. The next steps are the inevitable bug fixes, easier networking setup, perhaps more hardware platforms if users want that and more apps.

But then, we're looking to work with developers of Web applications who want to make it easier for their users to install their apps, and to developers of Internet-of-Things hardware who want to make it easier for developers to get Web applications running and maintained on their hardware. We all have the same goal here—make home servers easier—and we invite anybody who shares this goal to help make it so.

You might also be a really frustrated home user who's had it with with fixing /etc/i/forget/what Saturday afternoons and would much rather help fix the problem once and for all.

DS: What do you mean by personal servers? I can guess, but I'd rather have you tell me. And how are these different from personal clouds, stores, lockers and vaults, except in the sense of where they live?

JE: All these, in my mind, describe

a different part of mostly the same elephant. A personal server, for me, is a computer that is primarily accessed over the network, but it is one that I have root access to and nobody else does. (Or at least that I can have root access to if I want to, and I can take away root access from anybody else.) Such a personal server can be a physical piece of hardware or a virtualized one in the cloud. It's different from a non-personal server in that I use this server for my own, personal purposes and for those of my family, but not for setting up a Web site for the general public.

A personal cloud, a personal data locker or vault would be one kind of use of that personal server. Another use would be to control one's home appliances. So many kinds of personal applications have their natural home on a personal server, because by its very definition, nobody gets to touch the data on that server other than the owner.

DS: Why just Web apps?

JE: Package managers are great at helping administer most code on a Linux box, and UBOS heavily relies on pacman. But they are not able to administer Web apps because

they do not (and cannot) help with Web server configuration, database configuration and so forth. ubos-admin sits on top of the pacman package manager and adds that additional layer of automation.

DS: What kind of "intelligent Internet of Things devices" are you talking about?

JE: There seem to be four types of IoT "devices" emerging:

1. Sensors, for example, thermometers, often with very low power requirements.

2. Actuators, for example, relays to switch on something.

3. Cloud servers.

4. Higher-powered devices in the home that connect the Internet to the sensors and the actuators. For many hobbyists, that often is a Raspberry Pi.

This last category of devices is often powerful enough to run Web applications. It turns out that for many home IoT applications, there is no need to run Web applications in the cloud at all. So why not turn these devices into devices that run the Web apps directly?

This has the added advantage that our personal IoT data stays private in the home, and can't be abused by a cloud provider.

DS: What's the end state of the "Internet of our own things"? Or at least a state to which *Linux Journal* readers can aspire—and weigh in with some code and product?

JE: As geeks, and as an industry, we have a choice to make here about how we'd like the Internet of Things to look ten years from now.

There's the "NEST way", which is totally closed devices 100% dependent on being tethered to some corporate overlord. Google in this case. Some people consider these beautiful-looking devices to be nothing else than surveillance devices first and thermostats second. I guess they have a point. In any case, we have no control whatsoever over their functioning and terms of service, nor what they do with our data.

And there's the user-owned, free/libre, "indie" way, where every device is—at least in

principle—hackable. Where data stays home, or at least is only shared with anybody because the owner of the device decided that that is what they wanted to do. Where we can run the code we want on whatever device we buy or build. Where interoperability is not subject to whether some big companies love or hate each other this week.

Personally, I'm horrified at the first of these scenarios, and I know many other geeks are too, because all these great IoT projects keep popping up everywhere. The next step is to make this all reliable and maintainable enough that it's not just the geeks who can control their own homes. UBOS is one step in that direction, but many others are required. In fact, we might need a real barn-raising effort to together create an "Internet of our own things". Anybody up for that? I am!

LJ: Us too.■

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

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

Resources

UBOS: **http://ubos.net**

Johannes Ernst: **http://upon2020.com/blog**

Phil Windley: **http://www.windley.com**

Picos: **http://www.windley.com/archives/2014/09/events_picos_and_microservices.shtml**

CloudOS: **http://cloudos.me**

"The First Personal Platform—for Everything" by Doc Searls (in the October 2013 issue of *LJ*): **http://www.linuxjournal.com/content/first-personal-platform—-everything**

UMA: **https://kantarainitiative.org/confluence/display/uma/Home**

DistroWatch: **http://distrowatch.com**