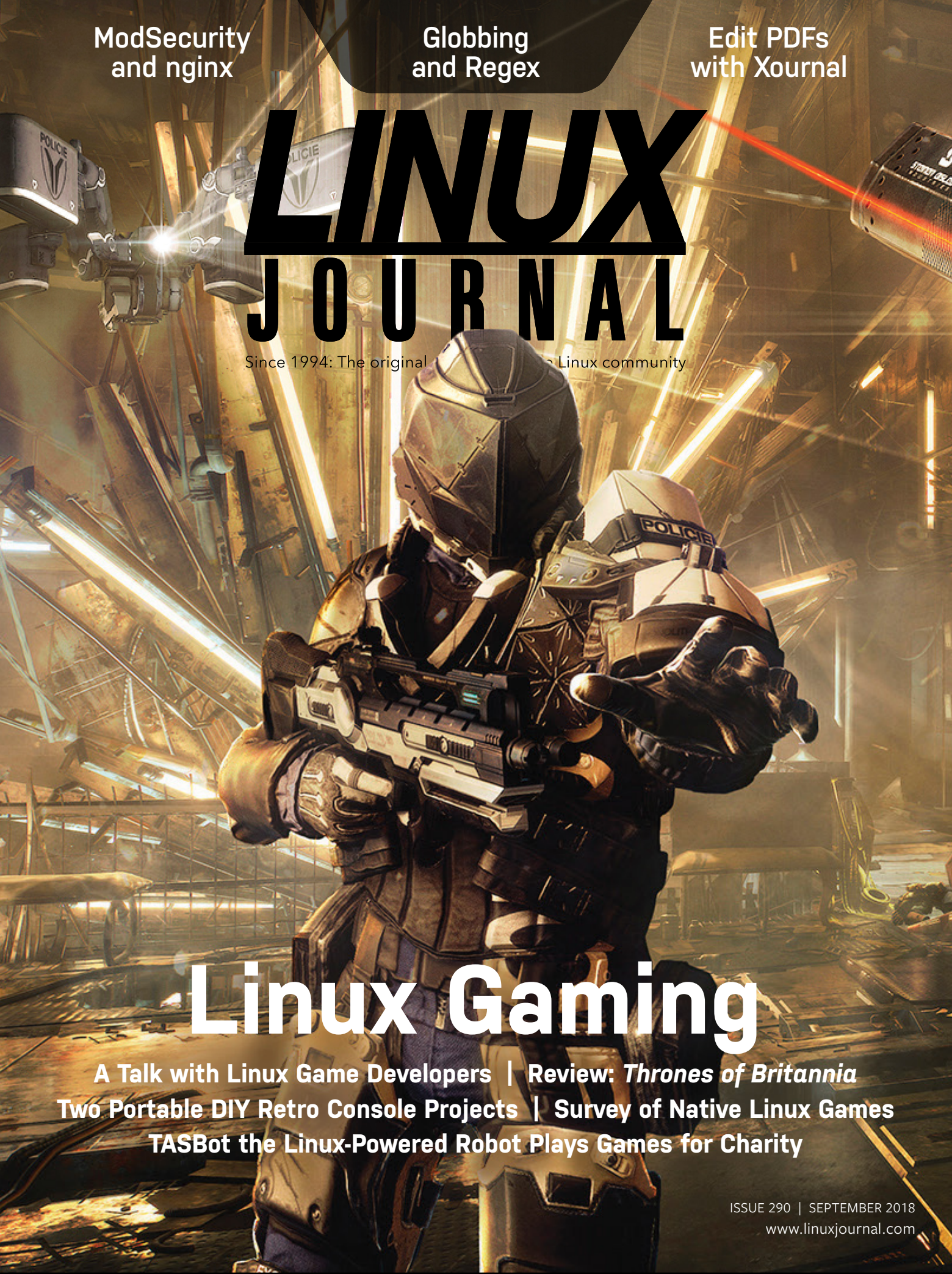
ModSecurity and nginx

Globbing and Regex

Edit PDFs with Xournal



LINUX JOURNAL

Since 1994: The original Linux community

Linux Gaming

A Talk with Linux Game Developers | Review: *Thrones of Britannia*
Two Portable DIY Retro Console Projects | Survey of Native Linux Games
TASBot the Linux-Powered Robot Plays Games for Charity

ISSUE 290 | SEPTEMBER 2018
www.linuxjournal.com

CONTENTS

SEPTEMBER 2018
ISSUE 290

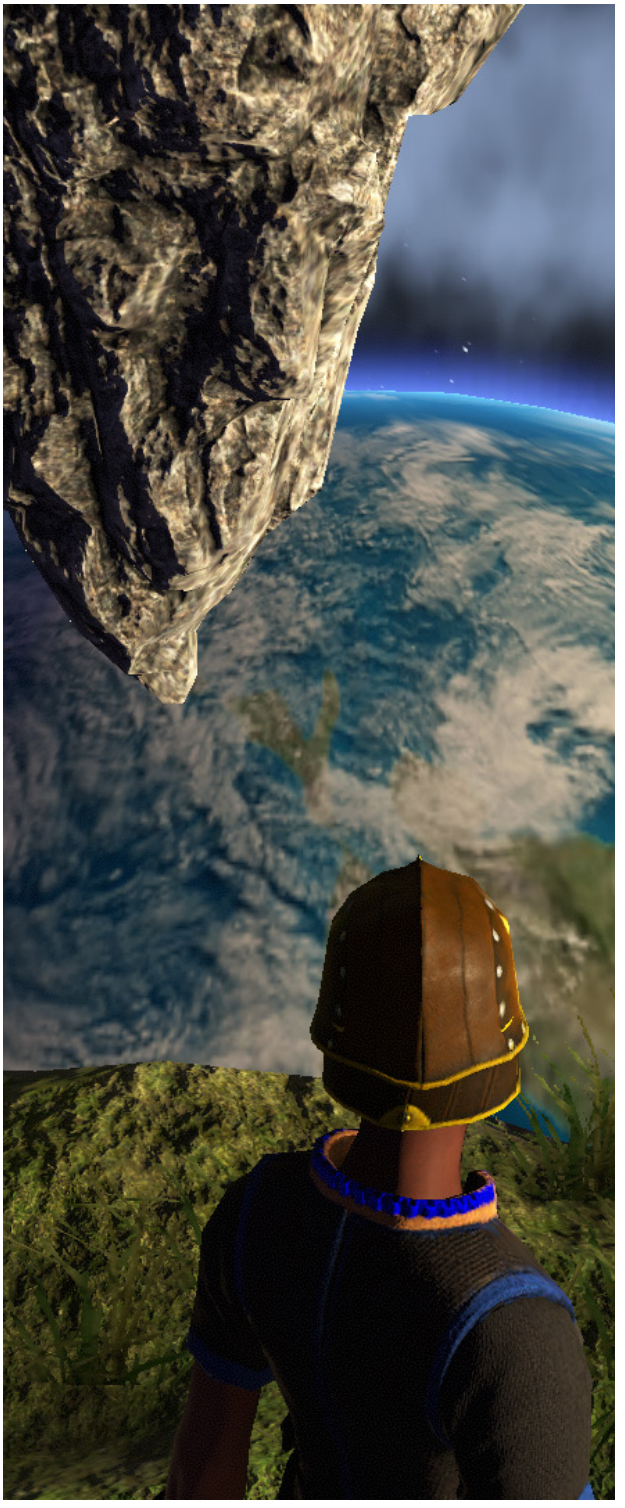


Image from Portalarium's *Shroud of the Avatar*.

Cover image from *Deus Ex: Mankind Divided*, developed by Eidos Montréal and published by Square Enix. Feral Interactive released the game for Linux in November 2016.

CONTENTS

LINUX JOURNAL (ISSN 1075-3583) is published monthly by Linux Journal, LLC. Subscription-related correspondence may be sent to 9597 Jones Rd, #331, Houston, TX 77065 USA. Subscription rate is $34.50/year. Subscriptions start with the next issue.

ARTICLES

AT YOUR SERVICE

SUBSCRIPTIONS: *Linux Journal* is available as a digital magazine, in PDF, EPUB and MOBI formats. Renewing your subscription, changing your email address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly online: http://www.linuxjournal.com/subs. Email us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, 9597 Jones Rd #331, Houston, TX 77065 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 different 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*, 9597 Jones Rd #331, Houston, TX 77065 USA. Letters may be edited for space and clarity.

SPONSORSHIP: We take digital privacy and digital responsibility seriously. We've wiped off all old advertising from *Linux Journal* and are starting with a clean slate. Ads we feature will no longer be of the spying kind you find on most sites, generally called "adtech". The one form of advertising we have brought back is sponsorship. That's where advertisers support *Linux Journal* because they like what we do and want to reach our readers in general. At their best, ads in a publication and on a site like *Linux Journal* provide useful information as well as financial support. There is symbiosis there. For further information, email: sponsorship@linuxjournal.com or call +1-281-944-5188.

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 online: http://www.linuxjournal.com/author.

NEWSLETTERS: 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.

LINUX JOURNAL

EDITOR IN CHIEF: Doc Searls, doc@linuxjournal.com

EXECUTIVE EDITOR: Jill Franklin, jill@linuxjournal.com

TECH EDITOR: Kyle Rankin, lj@greenfly.net

ASSOCIATE EDITOR: Shawn Powers, shawn@linuxjournal.com

EDITOR AT LARGE: Petros Koutoupis, petros@linux.com

CONTRIBUTING EDITOR: Zack Brown, zacharyb@gmail.com

SENIOR COLUMNIST: Reuven Lerner, reuven@lerner.co.il

SENIOR COLUMNIST: Dave Taylor, taylor@linuxjournal.com

PUBLISHER: Carlie Fairchild, publisher@linuxjournal.com

ASSOCIATE PUBLISHER: Mark Irgang, mark@linuxjournal.com

DIRECTOR OF DIGITAL EXPERIENCE:
Katherine Druckman, webmistress@linuxjournal.com

GRAPHIC DESIGNER: Garrick Antikajian, garrick@linuxjournal.com

COVER DESIGN: Carty Sewill

ACCOUNTANT: Candy Beauchamp, acct@linuxjournal.com

Linux Journal is published by, and is a registered trade name of,
Linux Journal, LLC. 4643 S. Ulster St. Ste 1120 Denver, CO 80237

SUBSCRIPTIONS
E-MAIL: subs@inuxjournal.com
URL: www.linuxjournal.com/subscribe
Mail: 9597 Jones Rd, #331, Houston, TX 77065

SPONSORSHIPS
E-MAIL: sponsorship@linuxjournal.com
Contact: Publisher Carlie Fairchild
Phone: +1-281-944-5188

Lazy Fully Scripted—Linux from Scratch

I really enjoyed Petros Koutoupis' article on building your own distro (see "DIY: Build a Custom Minimal Linux Distribution from Source" from the June 2018 issue). I have not had the time to build it yet, but I will. Also I'm very interested in building Fedora SPINS, mostly minimal because with adding PHP, NodeJS, Ruby, etc., it's very easy to eat up HDD space. I found a post from Alagappan Karthikeyan from India on building his own distro in three hours, based on Ubuntu.

But the video that impressed me the most was Tutorial: Linux from Scratch by Anton, and he has four videos on time lapse and successfully builds LFS. I got the book, but I never successfully did an installation.

I remember in one of my RHCE classes the instructor mentioned that everybody can copy/paste, and still it takes a good deal of time, but why not...so I put together a set of shell scripts to accomplish this task: https://github.com/dinooz/lfslfs/blob/master/lfslfs_get_started.sh. Requirements:

1) The scripts will be executed in the same order they are listed.

2) VirtualBox→Xubuntu Live CD and HDD with 10GB HDD.

3) As xubuntu:

```
wget https://raw.githubusercontent.com/dinooz/lfslfs/
↪master/lfslfs_get_started.sh
chmod 755 lfslfs_get_started.sh
./lfslfs_get_started.sh
```

4) All the needed shell scripts will be right there at your fingertips, executing one at a time, and will follow exactly as reading the online LFS.

5) Some scripts will be absolutely necessary to run as root and others as lfs, and the generic as xubuntu for generic OS checks.

I have a video that illustrates the first part of the scripts installing the temp system. Some notes are required for the second part because bash breaks the shell script, that's why I copy … software2.sh to another script to continue the execution.

I'm happy to say that I've learned a lot from this first part. I'm looking forward to another script recipe for X, depending on the desired environment, and who knows, maybe a package manager. I just wanted to share this with you guys. Keep doing a great job.

—Dinooz

Petros Koutoupis replies: Bernardino (Dinooz), thank you very much for the encouraging words. Traditional LFS cookbooks are written to build custom and more fully featured distributions (that is, with more packages) on (and only for) your local machine. The recipes provided on the official website are a wonderful source of information. Unfortunately, not many individuals are able to operate in such an environment, which is why I decided to take the cross-compilation route. The general theme behind the cross-compiled and lightweight Linux distribution is centered around building your minimal distribution for whatever architecture on your local machine and in a sandboxed environment. Then take the final image and deploy it anywhere—physical or virtual machine.

I do appreciate you bringing this GitHub project to my attention. For those who wish to build a more fully featured Linux distribution from source, these scripts definitely can help without the headache of running each command one at a time (copied from the cookbook and pasted into the CLI). In the interim, I have noted your suggestions. Since the publication of that guide, many have requested a second part. I definitely can look into adding a graphical environment and building a minimal X equipped with a basic window manager. Package management may be a tricky one.

Thank you again!

PEP 572

Regarding Reuven M. Lerner's "Python and Its Community Enter a New Phase" in the August 2018 issue: I'm not a fan of Python primarily because of its significant whitespace feature and the 2 vs. 3 incompatibility, although the significant whitespace generally causes me more grief than does the 2 vs. 3 issues. But it's incredibly useful because of the large collection of modules available, so I find it invaluable for rapid prototyping.

From the perspective of a non-sophisticated user of Python, I can't see the reason for all this controversy and animosity for a change that lets code like this:

```
x = 5

while x := x - 1:

    print(f"x is {x}")
```

now work, while still allowing code like this not to work as before:

```
x = 5

while x = x - 1:

    print(f"x is {x}")
```

What am I missing that makes this a "big deal"? It's been a part of C forever.

However, the print function in the example looks strange to me, so I tested it in a Python 3.5.2 interactive session, and I got this:

```
>>> x=3
>>> print(f"x is {x}")
  File "<stdin>", line 1
    print(f"x is {x}")
```

```
                         ^
SyntaxError: invalid syntax
```

So is there a typo in the article? Or is it a Python 3.5.2 vs. newer version issue?

—wally

Reuven M. Lerner replies: Yes, "f-strings", as they're known, were introduced in Python 3.6. The leading "f" before the opening quote allows you to evaluate anything within curly braces in a string. Thus:

```
x = 5
y = 2
print(f"{x} + {y} = {x+y}")
```

Before version 3.6, this would give you an error, as you saw.

Erratum: #geeklife Article in the August 2018 Issue

As a motor-home camper for the last 16 years, I was very interested in Kyle Rankin's "#geeklife: weBoost 4G-X OTR Review". The problems are very familiar to those of us who spend extended time far from civilization. I was very interested in the review of the weBoost, although I assume the captions on the external antenna pictures were interchanged.

Keep up the good work on *LJ*.

—norm scherer

*Yes, unfortunately those photos were accidentally switched during layout of that issue. It has been corrected on the website, so see the article **here** for the corrected photos/captions.—Ed.*

Server Automation

Bravo for Adam McPartlan's "Easy SSH Automation" tip in the August 2018 issue of *LJ*.

This is by far the clearest recipe for setting up passwordless login that I've seen—a task I've done countless times, but each time with trepidation.

I'd love to see more server automation scripts from Adam and others.

—Lloyd

Adam McPartlan replies: Thanks for the feedback. There is an excellent article written by Kyle Rankin and published by *Linux Journal* regarding the use of the ssh-agent, which is worth reading to help make things a little more secure for you while maintaining a key exchange-based authentication: "Secret Agent Man".

Server automation is a big subject, and there are many great tools. Using the bash script method is a good exercise. I have done work for companies that refused to accept the new tools of the trade like Ansible, so it's nice to get a good understanding of how you can achieve similar results. Most of my server automation is derived from the script shared in the article. I have used it to reboot servers, install and configure software, restart services, copy files, query databases and pull reports.

I also recommend reading up on expect scripting. This can help with working on many different network devices deploying new vlans, updating ntp settings and adding radius servers. I'll be sure to share in due course.

From Social Media

Mike Malveaux @m_mlvx: Mike Malveaux Retweeted linuxjournal: A bit of context for Microsoft buying a seat on the Linux Foundation, and being a top-ten contributor to Linux.

Here be dragons.

> **linuxjournal @linuxjournal:** Good Lockdown vs. Bad by @zackrobat: There's an ongoing series of skirmishes between corporations who want to sell products that users don't fully control and the kernel developers who want users to be the highest… https://www.linuxjournal.com/content/good-lockdown-vs-bad

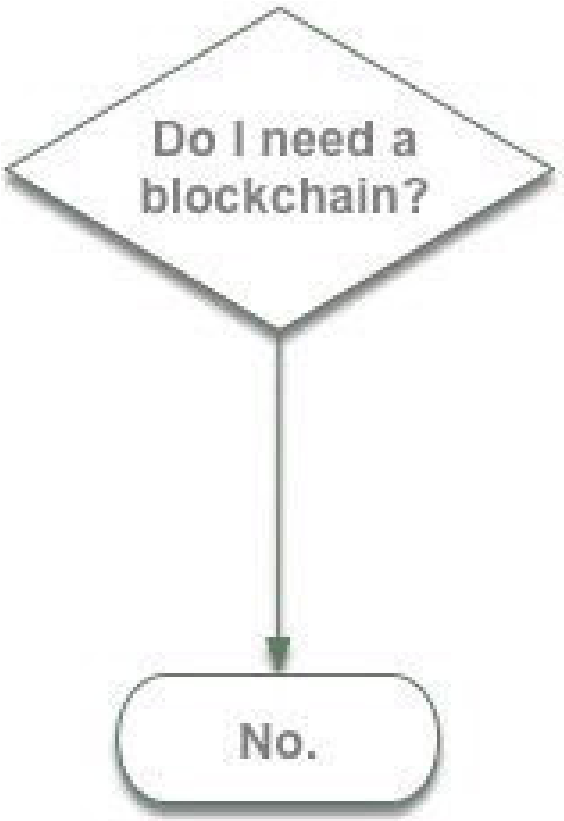
Mr. Penguin @0pensource: Canonical shifted its emphasis to the #cloud a few years back mainly because of the failure of #Linux to establish itself on the desktop.

And, it may turn out to be one of the best things ever to have happened to #opensource.

Jim Hall @jimfhall: An older article, but a good read. I use my @Raspberry_Pi at home as a backup server, print server, and (soon) streaming music server. It's easy! via @linuxjournal Raspberry Pi: the Perfect Home Server https://www.linuxjournal.com/content/raspberry-pi-perfect-home-server

Keith Bennett @keithrbennett: Today I downloaded Linux-Journal-2018-08.pdf and was reminded of this exchange. 4 years ago I asked Linux Journal to give their issue filenames names more logical and helpful. To their credit, they took the suggestion. Kudos, @linuxjournal. Also, sometimes it pays to speak up.

hans marcus @hansoegaboega:



Joachim Nilsson @troglobit: FollowFollow @troglobit. Joachim Nilsson Retweeted linuxjournal: If you're a long-time Linux user like me, it's time to step up. Let's all subscribe to Linux Journal! Quality articles written by great journalists (unlike you and me), with interesting topics for both the n00b, intermediate and advanced user. #linux #opensource

SEND *LJ* A LETTER *We'd love to hear your feedback on the magazine and specific articles. Please write us here or send email to ljeditor@linuxjournal.com.*

PHOTOS *Send your Linux-related photos to ljeditor@linuxjournal.com, and we'll publish the best ones here.*

STOP

Have you subscribed or renewed your subscription to *Linux Journal*?

We can't do this without you!

Follow the below link to get **$5 off your subscription order**, offer valid through September 30, 2018.
http://www.linuxjournal.com/save5

SUBSCRIBE TODAY TO RECEIVE:

12 monthly issues

Free ebook with paid order, *SysAdmin 101*

Free access to our nearly 300 back issue archive

Feedback? Please tell us what you think! https://www.surveymonkey.com/r/ljreader

Clearing Out /boot

The /boot partition sometimes needs a bit of attention. If you enable automatic updates, it will fill up with old kernels that you'll probably never need. It also will stop you from running aptitude to install or remove anything. If you find yourself in this situation, you can use dpkg to get around it. dpkg is the higher-level package manager in Debian-based distributions, and it's very useful when aptitude has broken.

To see the status of your partitions, run: `df -h`:

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.0G   12K  3.0G   1% /dev
tmpfs           597M  528K  597M   1% /run
/dev/dm-0        97G   14G   78G  15% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            5.0M     0  5.0M   0% /run/lock
none            3.0G     0  3.0G   0% /run/shm
none            100M     0  100M   0% /run/user
/dev/sda1       228M  219M     0 100% /boot
```

If you look in the directory /boot, you will see it full of old kernels and images. It is not advisable just to delete them, as you can break your system. Run `uname -r`, which will tell you what kernel you are currently on:

```
3.13.0-137-generic
```

Let's find out which kernels are installed and which can be purged from your system. To do this, run the following:

```
dpkg --list "linux-image*" | grep -v $(uname) | grep ii
```

This will use `dpkg` to list all Linux kernel images (excluding the one you are using) that are installed.

The output still might be quite long, so let's refine it by piping the results in to `awk`. The `awk` command below is an instruction to print the second column from the output:

```
dpkg --list "linux-image*" | grep -v $(uname -r) |
 ↪grep ii | awk '{ print $2 }'
```

This provides a list to work with, and you can stick it in a script or run it from the command line to purge them all.

Caution: make sure the kernel you are using is not in the list. You should have eliminated that when you specified `grep -v $(uname -r)`. The `-v` tells `grep` to exclude anything that contains the output of `uname -r`.

If you are happy and have sudo privileges, go ahead:

```
sudo dpkg --purge $(dpkg --list "linux-image*" | grep -v
 ↪$(uname -r) | grep ii | awk '{ print $2 }')
```

To finish off, run `sudo update-grub2`. This will ensure that grub is updated with the available kernels. Otherwise, you may be heading for trouble. Then fix aptitude by running: `sudo apt-get -f install`, followed by `sudo apt-get auto remove` to clear the images out of aptitude.

Look at your partition, and you will see that it has free space:

```
Filesystem      Size  Used Avail Use% Mounted on
udev            3.0G   12K  3.0G   1% /dev
tmpfs           597M  528K  597M   1% /run
/dev/dm-0        97G   13G   79G  14% /
```

```
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            5.0M     0  5.0M   0% /run/lock
none            3.0G     0  3.0G   0% /run/shm
None            100M     0  100M   0% /run/use
/dev/sda1       228M   98M   118M  46% /boot
```

Adam McPartlan is Father of Twins - Linux lover, Open Source Enthusiast - LFCS, AWS Cloud Practitioner. Follow him on Twitter: @mcparty.

VCs Are Investing Big into a New Cryptocurrency: Introducing Handshake

The entire landscape of how we authenticate domain names likely will see a complete overhaul, all powered by blockchain technologies. Just released, Handshake brings with it the much needed security and reliability on which we rely. Backed by venture capitalists and industry-established blockchain developers, Handshake has raised $10.2 million to replace the current digital entities maintaining our current internet infrastructure.

The project and protocol has been led by Joseph Poon (creator of Bitcoin's Lightning Network), Andrew Lee (CEO of Purse), Andrew Lee (founder of Private Internet Access or PIA) and Christopher Jeffrey (CTO of Purse). The effort also is backed by 67 individuals with funding coming from A16z, Founders Fund, Sequoia Capital, Greylock Partners, Polychain Capital and Draper Associates.

The Handshake project pledges to donate its initial funding of $10.2 million to FOSS projects, university research departments and more. The list of recipients includes projects and foundations such as the Apache Software Foundation,

FreeBSD, Reproducible Builds, GNOME, FSF, SFC, Outreachy, ArchLinux, systemd and many more.

What Is Handshake?

Handshake aims to be a wholly democratic and decentralized certificate authority and naming system. Handshake does not replace the Domain Name System (DNS). It is, however, an alternative to today's certificate authorities—that is, it uses a decentralized trust anchor to prove domain ownership. Although the primary goal of the project is to simplify and secure top-level domain registration while also *making the root zone uncensorable, permissionless and free of gatekeepers*.

A traditional root DNS supports the current infrastructure of the internet and, therefore, facilitates online access. The root servers hosting the internet publish root zone file contents, which are responsible for the internet's DNS functionality. DNS associates information with domain names and maps them to public-facing IP addresses.

The way Handshake differs from this is that it's all peer to peer. Every peer is responsible for validating and managing the root zone (via the use of "light clients"). All existing entries in the root zone file will form the genesis block of the blockchain supporting it. The same root zone will be distributed across the nodes forming the chain. The implementation allows for any participant to help host this distributed root zone and add to it.

How Does It Work?

Handshake makes use of a coin system for name registration (that is, the Handshake coin or HNS). It is the mechanism by which participants are able to transfer, register and update internet domain names. Currently, Handshake has opened a faucet to distribute HNS coins to qualified FOSS contributors. If you are one such contributor and you meet the project's criteria, you can sign up here.

Once HNS coins have been distributed, the Handshake mainnet launches. A "mainnet" forms the central part of a blockchain. In fact, it *is* the blockchain in

that it carries out the functionality of transferring digital currency from senders to recipients. This is the point where coin-holders can start auctioning, registering and transferring top-level domains.

With enough support and Non-Governmental Organization (NGO) cooperation, Handshake eventually will migrate to a more global distribution. Project governance and maintainability are and will continue to be community-driven.

To Learn More

You can find additional information on the official Handshake website: https://handshake.org. In addition, you can access all source code from the project's GitHub page: https://github.com/handshake-org.

—Petros Koutoupis

Edit PDFs with Xournal

Forget all of those magical command-line PDF incantations and edit your PDFs easily with Xournal.

Somehow, despite all the issues with proprietary clients and the history of security issues with Acrobat, PDFs have become the de facto standard for your average print-ready document shared around the office. Sure, people might use some kind of open document format or a cloud editor if they intend to edit a document, but if the goal is to print the document or lock its contents in place, most people these days will export it to a PDF.

Reading PDFs is typically fine on Linux, because Linux has plenty of applications that can open PDFs for viewing, and you easily can print PDFs under Linux as well. Even Adobe supplied a proprietary (and somewhat outdated) port of its Acrobat Reader for Linux. Some distributions also offer the ability to create a special software printer that converts any print job sent to it into a local PDF file.

The problem comes when people want to turn read-only print-ready PDFs into read-write documents you need to modify. As more people work in paperless offices with strictly digital documents and fewer people own fax machines, you are more likely to find official documents like contracts show up in your INBOX in PDF format. These contracts likely were created with a proprietary PDF editor tool, and they usually have blanks for you to fill in and often signature lines so you can add a real signature. Unfortunately, for the longest time, even if you were using Adobe's own Linux port of Acrobat Reader, you couldn't reliably edit these PDFs, and you certainly couldn't easily add a real signature.

A lot of Linux applications claim the ability to edit PDFs from graphical tools like GIMP, or the aforementioned Acrobat Reader or tools like Inkscape. In the past, I've even gone so far as to use command-line tools to convert a PDF into multiple pages of a different

format, edit that format, then use the command-line tools to convert it back to a PDF.

Then I discovered Xournal. Xournal is a graphical tool that's designed for note-taking and sketching either with a keyboard and mouse or even with a tablet and stylus. This program is pretty common, and you should be able to install it in any major Linux distribution, but otherwise, you can download the software from its Sourceforge page.

The particularly nice thing about Xournal is that it can import PDFs and display them like any other document, but because Xournal is designed for note-taking, you can pick its text or pencil tools and type or draw directly on the PDF! This means when you get a contract with a bunch of blanks to fill in, you can select the text tool from the toolbar, select the area where you want to type, and then fill in all those blanks. Then when you get to the signature page, you can zoom in on the signature section, select the pencil tool and then sign with your mouse! If you have a tablet or a laptop with a touchscreen, it's even easier, as you can use a stylus.

The most important thing to note when editing PDFs with Xournal is that to save your changes, you don't just click Save but instead click File→Export to PDF. If you are editing a PDF, I suggest exporting into a new document—I've noticed in the past that sometimes if I save on top of an existing PDF, it will overwrite only a particular layer, so I'll see a blank document apart from my changes. If you export to a new PDF, you can avoid this risk while also preserving your original, unedited PDF.

So next time you get a PDF you need to edit, put away those crazy command-line tools (and that Windows VM, for shame!) and break out Xournal. It's easy, works well, and I wish I'd known about it years ago!

—Kyle Rankin

Resources

Xournal SourceForge Page

Patreon and *Linux Journal*

PATREON

Together with the help of *Linux Journal* supporters and subscribers, we can offer trusted reporting for the world of open-source today, tomorrow and in the future. To our subscribers, old and new, we sincerely thank you for your continued support. In addition to magazine subscriptions, we are now receiving support from readers via Patreon on our website. *LJ* community members who pledge $20 per month or more will be featured each month in the magazine. A very special thank you this month goes to:

- Alex Bradaric
- Appahost.com
- Chris Short
- David Breakey
- Dr. Stuart Makowski
- Josh Simmons
- Mostly_Linux
- NDCHost.com
- Robert J. Hansen

BECOME A PATRON

FOSS Project Spotlight: Nitrux, a Linux Distribution with a Focus on AppImages and Atomic Upgrades

Nitrux is a Linux distribution with a focus on portable, application formats like AppImages. Nitrux uses KDE Plasma 5 and KDE Applications, and it also uses our in-house software suite Nomad Desktop.

What Can You Use Nitrux For?

Well, just about anything! You can surf the internet, word-process, send email, create spreadsheets, listen to music, watch movies, chat, play games, code, do photo editing, create content—whatever you want!

Nitrux's main feature is the Nomad Desktop, which aims to extend Plasma to suit new users without compromising its power and flexibility for experts. Nomad's features:

- The System Tray replaces the traditional Plasma version.
- An expanded notification center allows users to manage notifications in a friendlier manner.
- Easier access to managing networks: quick access to different networks settings without having to search for them.
- Improved media controls: a less confusing way to adjust the application's volume and integrated media controls.
- Calendar and weather: displays the traditional Plasma calendar but also adds the ability to see appointments and the ability to configure location settings to display the weather.
- Custom Plasma 5 artwork: including Look and Feel, Plasma theme, Kvantum theme, icon theme, cursor themes, SDDM themes, Konsole theme and Aurorae window decoration.

Nitrux is a complete operating system that ships the essential apps and services for daily use: office applications, PDF reader, image editor, music and video players and so on. We also include non-KDE or Qt applications like Chromium and LibreOffice that together create a friendly user experience.

Available Out of the Box

Nitrux includes a selection of applications carefully chosen to perform the best when using your computer:

- Dolphin: file manager.
- Kate: advanced text editor.
- Ark: archiving tool.
- Konsole: terminal emulator.
- Chromium: web browser.
- Babe: music player.
- VLC: multimedia player.
- LibreOffice: open-source office suite.
- Showimage: image viewer.

Explore a Universe of Apps in Nitrux

The NX Software Center is a free application that provides Linux users with a modern and easy way to manage the software installed on their open-source operating systems. Its features allow you to search, install and manage AppImages. AppImages are faster to install, easier to create and safer to run. AppImages aim to work on any distribution or device, from IoT devices to servers, desktops and mobile devices.

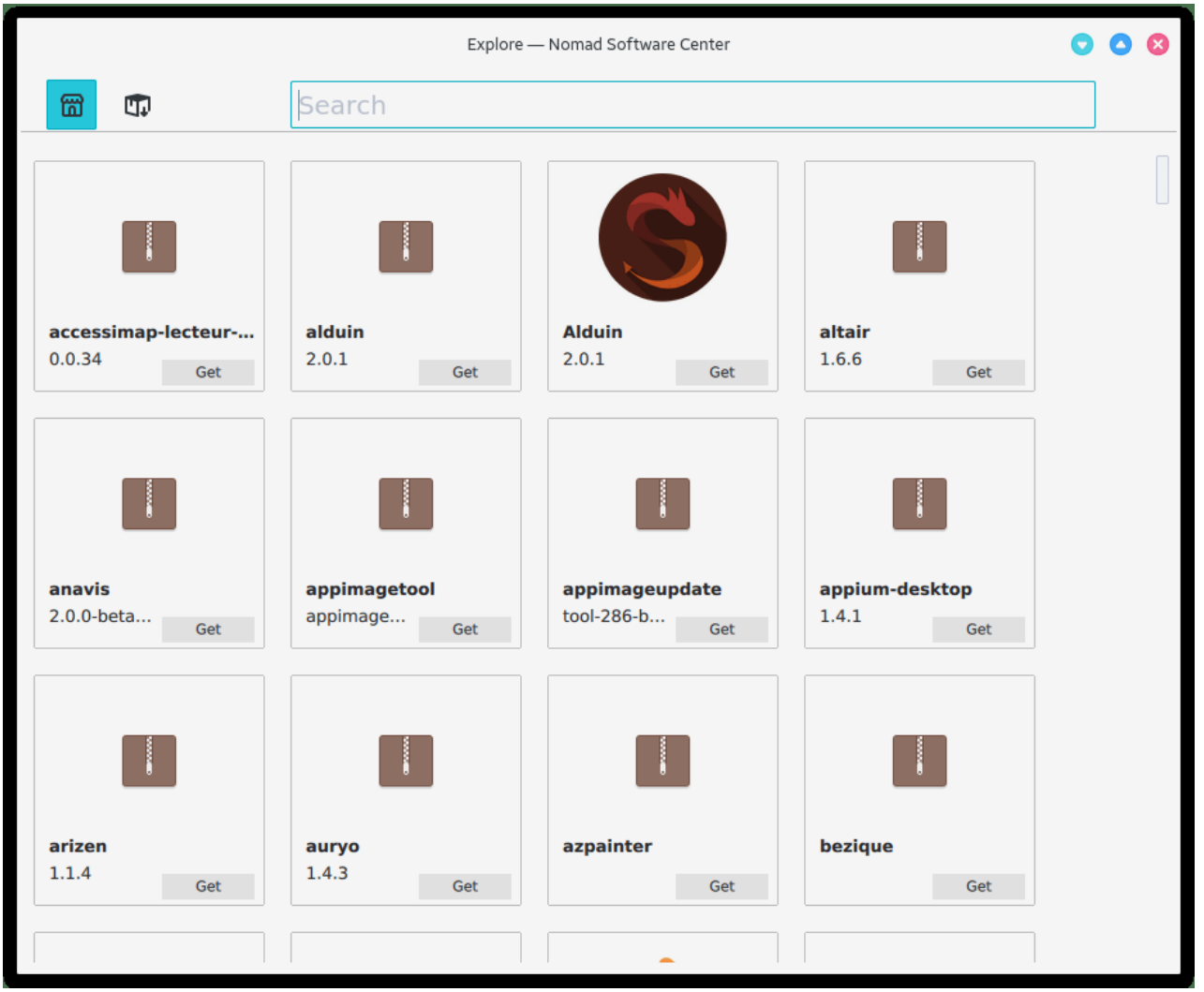


Figure 1. The Nomad Software Center

Securing Your Desktop and Workstation

Nomad Firewall is a firewall tool for Linux that uses the Qt toolkit. You can use the wizard to create a basic firewall and then streamline it further using the dynamic rules. You can open and close ports with a few clicks, or monitor your services giving access only to a select few. Nomad Firewall is an open-source application that provides users with a graphical user

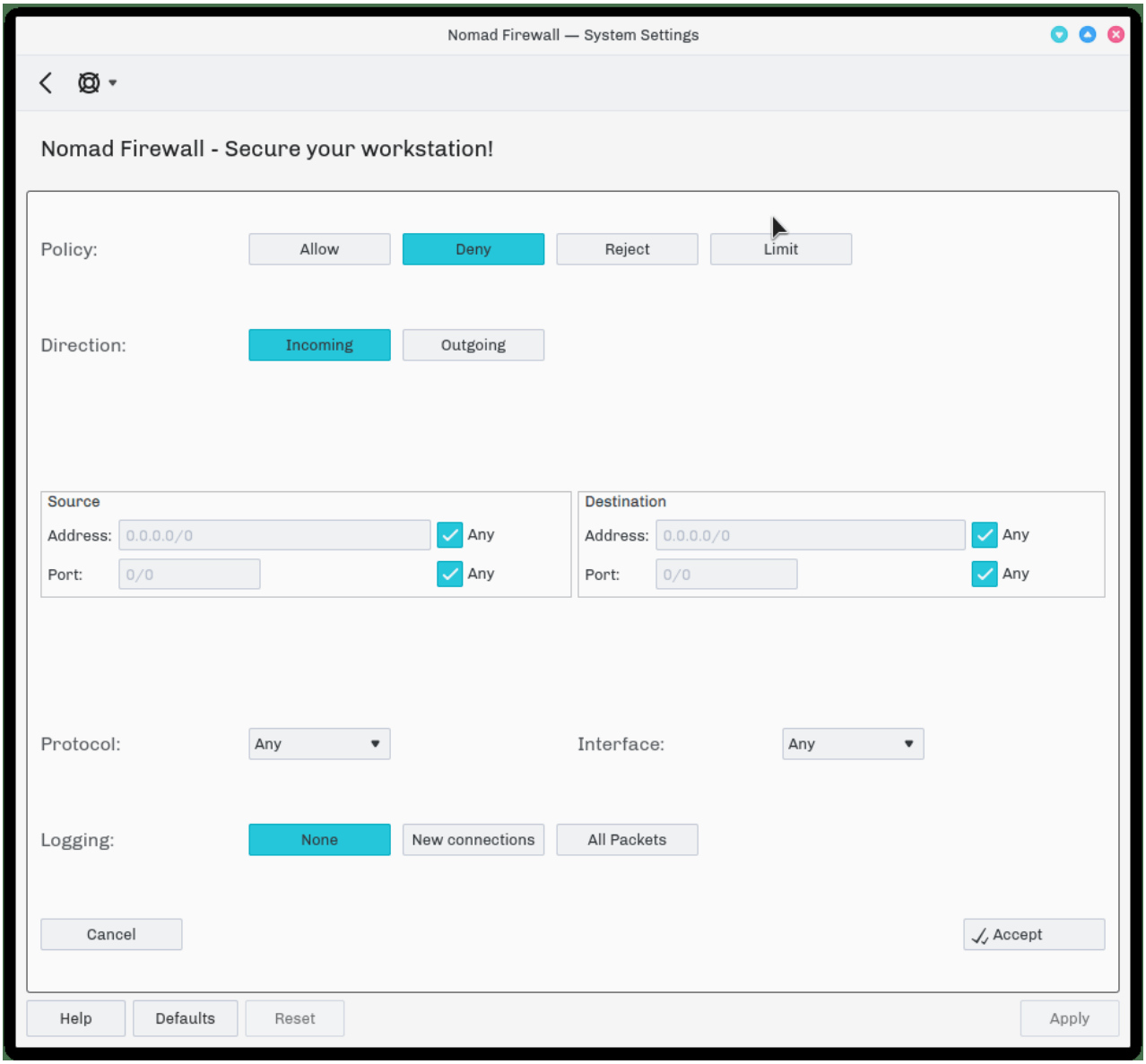


Figure 2. The Nomad Firewall

interface (GUI) for the ufw (Uncomplicated Firewall)/iptables command-line interface, which lets users manage the Linux kernel's packet filtering system.

What We're Working On

znx: https://github.com/Nitrux/znx

znx allows users to perform the following:

- Make parallel deployments of bootable ISO images (Linux-based distributions are expected).
- Upgrade systems in a safe (atomic) way.
- Update images based on differential content.

For more information about how znx works, see the documentation.

Maui Project: https://maui-project.org

The Maui Project is a free and modular front-end framework for developing fast and powerful user experiences:

- Maui Kit: a set of templated controls and tools based off QQC2 and Kirigami, shared among the Maui set of applications. Maui Kit helps to build UIs quickly that follow the Maui HIG and bring ready-to-go tools for different platforms, such as Android and Linux.
- Maui Apps: applications built using the Maui Kit provide a seamless transition between mobile and desktop technology—where the line between desktop and mobile is blurred. Using the same codebase, Maui Apps provide users with one app for multiple form factors and operating systems.

For more information and to download Nitrux, visit https://nxos.org/#welcome.

—Nitrux Latinoamericana S.C.

A Look at KDE's KAlgebra

Many of the programs I've covered in the past have have been desktop-environment-agnostic—all they required was some sort of graphical display running. This article looks at one of the programs available in the KDE desktop environment, KAlgebra.

You can use your distribution's package management system to install it, or you can use Discover, KDE's package manager. After it's installed, you can start it from the command line or the launch menu.

When you first start KAlgebra, you get a blank slate to start doing calculations.

The screen layout is a large main pane where all of the calculations and their results

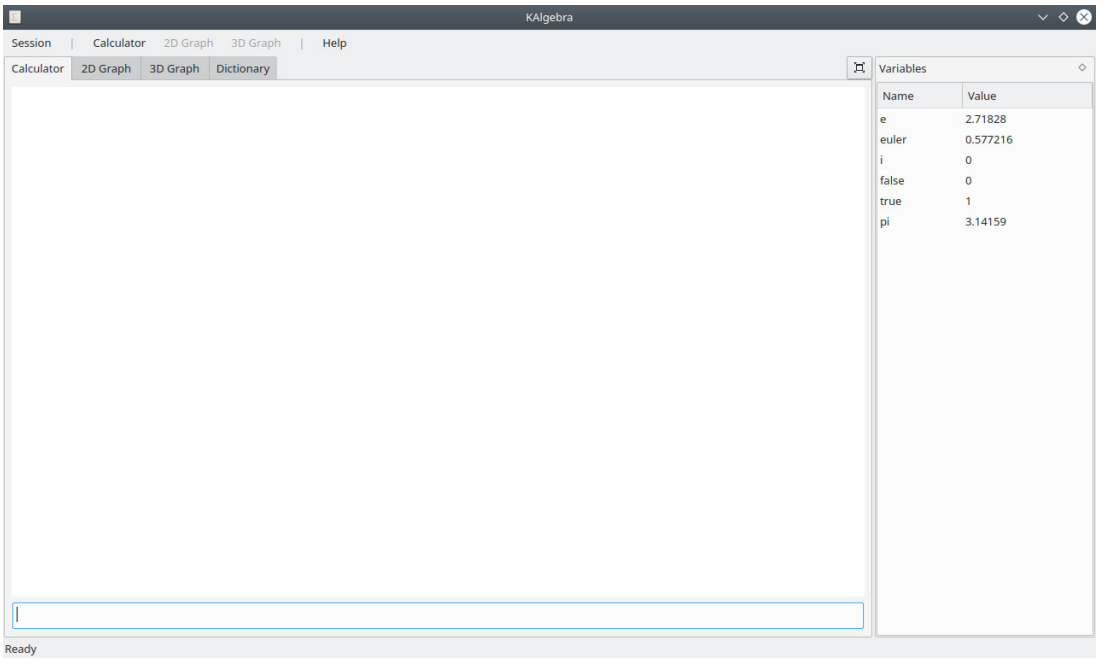


Figure 1. When you start KAlgebra, you get a blank canvas for doing calculations.

are displayed. At the top of this pane are four tabs: Calculator, 2D Graph, 3D Graph and Dictionary. There's also a smaller pane on the right-hand side used for different purposes for each tab.

In the calculator tab, the side pane gives a list of variables, including predefined variables for things like pi or euler, available when you start your new session. You can add new variables with the following syntax:

```
a := 3
```

This creates a new variable named `a` with an initial value of 3. This new variable also will be visible in the list on the right-hand side. Using these variables is as easy as executing them. For example, you can double it with the following:

```
a * 2
```

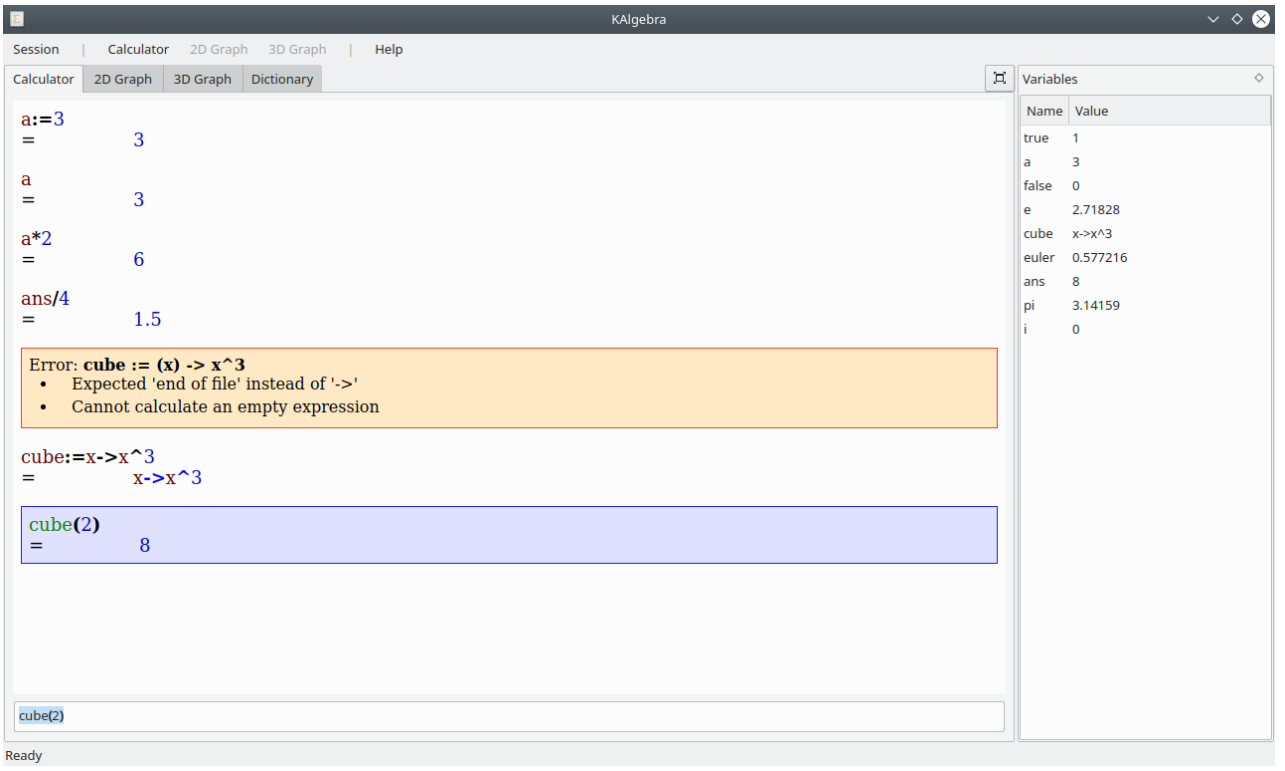


Figure 2. KAlgebra lets you create your own variables and functions for even more complex calculations.

There is a special variable called `ans` that you can use to get the result from your most recent calculation. All of the standard mathematical operators are available for doing calculations.

There's also a complete set of functions for doing more complex calculations, such as trigonometric functions, mathematical functions like absolute value or floor, and even calculus functions like finding the derivative. For instance, the following lets you find the sine of 45 degrees:

```
sin(45)
```

You also can define your own functions using the lambda operator `->`. If you want to create a function that calculates cubes, you could do this:

```
x -> x^3
```

This is pretty hard to use, so you may want to assign it to a variable name:

```
cube := x -> x^3
```

You then can use it just like any other function, and it also shows up in the list of variables on the right-hand side pane.

KAlgebra also has built-in graphing capabilities. Click the 2D Graph tab to bring up an empty graphing pane.

If you don't already have any functions defined, you will be given a pane on the right-hand side where you can add a new one. You can have several functions listed and select which ones you want to plot on the graph. The display includes two red lines that define the x and y coordinates, as well as a blue line that shows the slope of the function at the location of the cursor. If you right-click the graph, you'll see a drop-down menu with options like changing the graph's resolution or whether to display the grid. These options are also available from the 2D Graph menu item.

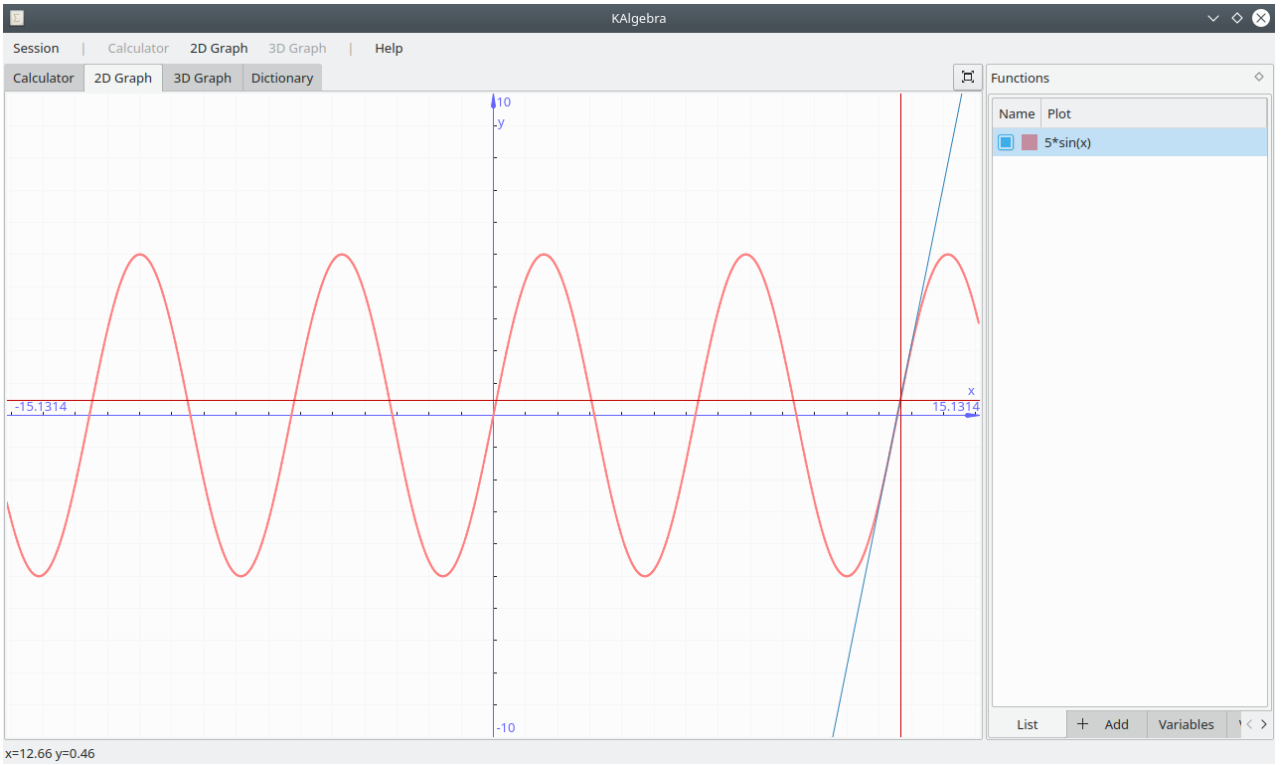


Figure 3. KAlgebra lets you plot multiple functions to make comparisons easier.

Once you generate the plots you want, you can save the results either to an image file or an SVG file.

KAlgebra also has the capability for 3D graphing, which isn't as common. Click the 3D Graph tab to show a new graphing pane, which takes up the entire window.

At the bottom of the pane, there is a text box where you can enter the function you want to plot. For example, the 3D plot you see here was generated with the following formula:

```
sin(x)*cos(y)
```

You can interact with the plot window with your mouse by grabbing and dragging the entire frame to get other views of the surface. You also can zoom in and out with

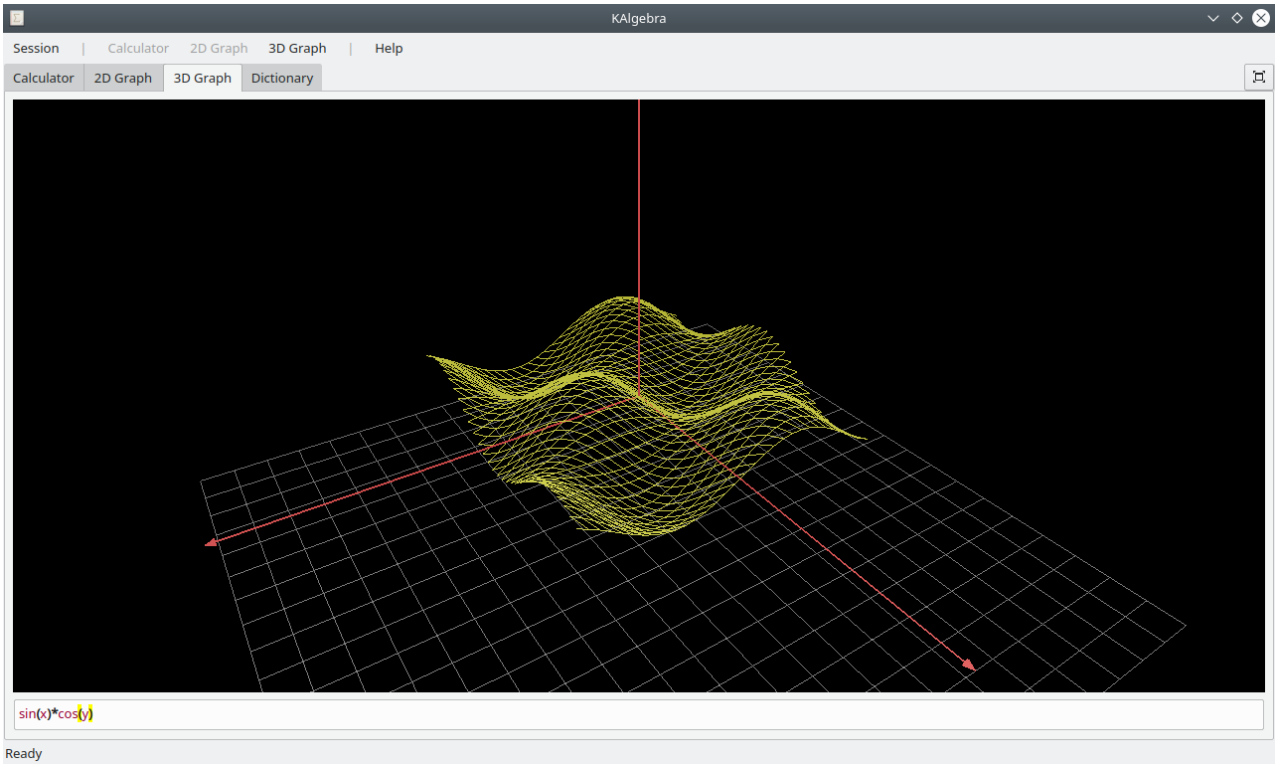


Figure 4. KAlgebra has the ability to do 3D plots of multivariate functions.

your mouse. If you want to change how the function is actually displayed, click the 3D Graph menu item. Here, you can choose either dots, lines or a solid surface to show the results of the function of interest.

The last tab provides a dictionary of all of the functions available in KAlgebra. This lets you explore the provided functions to help you figure out what you might be able to use in your own calculations. There's an information pane that provides a description, a list of parameters and a number of examples. Below it is a plot window that graphs the given function, so you easily can visualize the behavior of the selected function. It's a handy way to select the base functions you want to use to build up more complex functions.

Switching back to the Calculator tab, this activates the Calculator menu item. Under it, you can save and load script files. These files are simple text files, with the .kal file

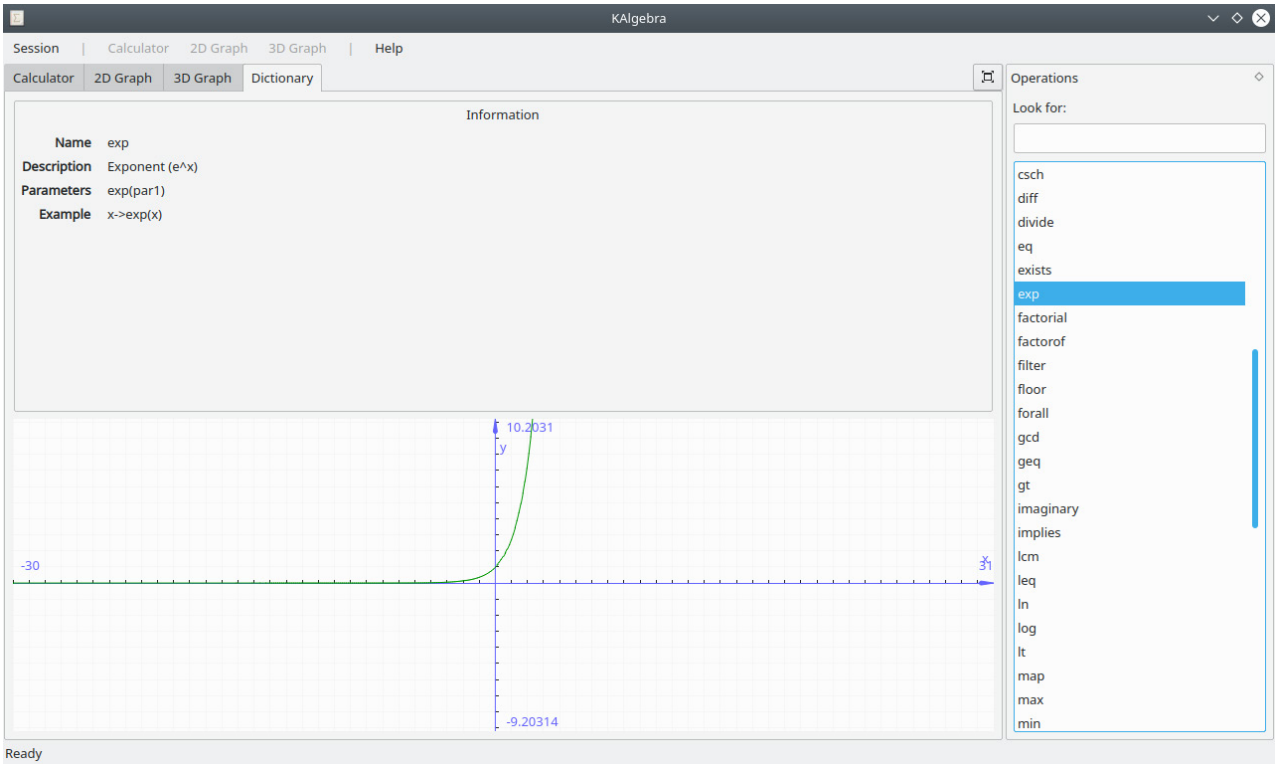


Figure 5. The dictionary tab provides detailed information on the built-in functions available in KAlgebra, along with a plot of the function.

ending, that store the series of KAlgebra statements in your current session. You then can load this script again later to go back to your previous spot, or you can share the file with others.

For ideas on what you can do with KAlgebra, check out the handbook. This handbook also is available from KAlgebra's Help menu. For even more examples, see the KAlgebra section of the KDE UserBase Wiki.

—Joey Bernard

Stop Killing Your Cattle: Server Infrastructure Advice

It's great to treat your infrastructure like cattle—until it comes to troubleshooting.

If you've spent enough time at DevOps conferences, you've heard the phrase "pets versus cattle" used to describe server infrastructure. The idea behind this concept is that traditional infrastructure was built by hand without much automation, and therefore, servers were treated more like special pets—you would do anything you could to keep your pet alive, and you knew it by name because you hand-crafted its configuration. As a result, it would take a lot of effort to create a duplicate server if it ever went down. By contrast, modern DevOps concepts encourage creating "cattle", which means that instead of unique, hand-crafted servers, you use automation tools to build your servers so that no individual server is special—they are all just farm animals—and therefore, if a particular server dies, it's no problem, because you can respawn an exact copy with your automation tools in no time.

If you want your infrastructure and your team to scale, there's a lot of wisdom in treating servers more like cattle than pets. Unfortunately, there's also a downside to this approach. Some administrators, particularly those that are more junior-level, have extended the concept of disposable servers to the point that it has affected their troubleshooting process. Since servers are disposable, and sysadmins can spawn a replacement so easily, at the first hint of trouble with a particular server or service, these administrators destroy and replace it in hopes that the replacement won't show the problem. Essentially, this is the "reboot the Windows machine" approach IT teams used in the 1990s (and Linux admins sneered at) only applied to the cloud.

This approach isn't dangerous because it is ineffective. It's dangerous exactly because it often works. If you have a problem with a machine and reboot it, or if you have a problem with a cloud server and you destroy and respawn it, often the problem does go away. Because the approach appears to work and because it's a lot easier than actually performing troubleshooting steps, that success then reinforces rebooting and respawning as the first resort, not the last resort that it should be.

The problem with respawning or rebooting before troubleshooting is that since the problem often goes away after doing that, you no longer can perform any troubleshooting to track down the root cause. To extend the cattle metaphor, it's like shooting every cow that is a little sluggish or shows signs of a cold, because they might have mad cow disease and not actually testing the cow for the disease. If you aren't careful, you'll find you've let a problem go untreated until it's spread to the rest of your herd. Without knowing the root cause, you can't perform any steps to prevent it in the future, and although the current issue may not have caused a major outage, there's no way to know whether you'll get off so easy the next time it happens. So although you may save time by not troubleshooting, that's time you lose from gaining troubleshooting experience. Eventually, you'll need to flex that troubleshooting muscle, and if you haven't exercised it, you may find yourself with a problem you can't solve.

In short, automation is great, and it's incredibly important in modern infrastructure to be able to respawn any host quickly and easily—just don't turn that infrastructure best practice into a troubleshooting worst practice.

—Kyle Rankin

News Briefs

Visit LinuxJournal.com for daily news briefs.

- The AP reports that Google tracks your location history, even if you turn "Location History" off. On both Android devices and iPhones, Google stores "your location data even if you've used a privacy setting that says it will prevent Google from doing so. Computer-science researchers at Princeton confirmed these findings at the AP's request." This Wired post describes how you actually can disable location tracking.

- The Academy of Motion Picture Arts and Sciences and The Linux Foundation launched the Academy Software Foundation. The ASF's mission is to "increase the quality and quantity of contributions to the content creation industry's open source software base; to provide a neutral forum to coordinate cross-project efforts; to provide a common build and test infrastructure; and to provide individuals and organizations a clear path to participation in advancing our open source ecosystem". Interested developers can sign up to join the mailing list here.

- The Linux 4.18 kernel is out. See this Phoronix post for a list of the best features of this new kernel.

- Ring-KDE 3.0.0, a GNU Ring.cx client, has been released. GNU Ring is a secure, distributed communication platform based on open industry-standard technologies for audio calls, video conferences, chat, screen-sharing and peer-to-peer file transfer. This new version of Ring-KDE is a full rewrite of the app "to use more modern technologies such as touch support, QtQuick2 and KDE Kirigami adaptive widget framework". When you join GNU Ring, "no servers or centralized accounts are needed. Besides an optional blockchain-based way to reserve your username against takeover, nothing leaves your device", and Ring-KDE "provides a simple wizard to help you create credentials or import your personal information from other devices." For more info, also visit here.

- Intel debuts a totally silent ruler-shaped solid state drive, the Intel SSD DC

P4500. This SSD can store 32 terabytes—"equivalent to triple the entire printed collection of the U.S. Library of Congress". In addition, "the no-moving-parts ruler-shaped SSDs can be lined up 32 side-by-side, to hold up to a petabyte in a single server slot. Compared with a traditional SSD, the 'ruler' requires half the airflow to keep cool. And compared with hard disk storage, the new 3D NAND SSD sips one-tenth the power and requires just one-twentieth the space."

- The Mozilla IoT team announced the 0.5 release of the Things Gateway recently, which is "packed full of new features including customisable devices, a more powerful rules engine, an interactive floorplan and an experimental smart assistant you can talk to". If you want to try out this new version of the gateway, you can download it from **here** and use it on your Raspberry Pi. According to the press release, "A powerful new 'capabilities' system means that devices are no longer restricted to a predefined set of Web Thing Types, but can be assembled from an extensible schema-based system of '**capabilities**' through our new **schema** repository. This means that developers have much more flexibility to create weird and wacky devices, and users have more control over how the device is used."

- Docker is moving to a new release and support cycle for its Community Edition (CE) releases, **ServerWatch reports**. New Docker CE versions will come out every six months, and each new CE release will be supported for seven months. The next CE Stable release is due out in September. Docker CE Edge releases will move to a faster cycle—from monthly to nightly builds.

- The city of Rome is **switching to open-source LibreOffice**. The city installed LibreOffice alongside the proprietary alternative on all of its 14,000 PC workstations in April and is gradually making the change. There are 112 staff members called "innovation champions", who are in favour of free and open source, and who are helping with the switch by explaining the reasons for changing to open source and training co-workers (source: **Open Source Observatory**).

- No Starch Press recently released *The Rust Programming Language*, the "undisputed go-to book on Rust", authored by two members of the Rust core

team—Steve Klabnik and Carol Nichols—and featuring contributions from 42 community members. No Starch comments that "this huge undertaking is sure to make some waves and help build the Rust community". The book is published under an open license and is available for free via the Rust site or for purchase from No Starch in either in print or ebook format.

- A new version of KStars—the free, open-source, cross-platform astronomy software—has been released. Version 2.9.7 includes new features, such as improvements to the polar alignment assistant and support for Ekos Live, as well as stability fixes. See the release notes for all the changes.

- Red Hat's *Road to A.I.* film has been chosen as an entry in the 19th Annual Real to Reel International Film Festival. According to the Red Hat blog post, this "documentary film looks at the current state of the emerging autonomous vehicle industry, how it is shaping the future of public transportation, why it is a best use case for advancing artificial intelligence and how open source can fill the gap between the present and the future of autonomy." The *Road to A.I.* is the fourth in Red Hat's Open Source Stories series, and you can view it here.

- Vivaldi Technologies has added a new privacy-focused search engine called Qwant to its Vivaldi web browser. Qwant doesn't store cookies or search history. Softpedia News quotes CEO and co-founder of Vivaldi Jon von Tetzchner: "We believe that the Internet can do better. We do not believe in tracking our users or in data profiling." You need version 1.15 of Vivaldi in order to enable Qwant.

- System76 has moved into its new manufacturing facility in Denver, Colorado. The company will begin making computers in the US, rather than just assembling them. See the System76 blog post for photos of the new digs.

Two Portable DIY Retro Gaming Consoles

A look at Adafruit's PiGRRL Zero vs. Hardkernel's ODROID-GO.

By Kyle Rankin

Kyle Rankin is a Tech Editor and columnist at *Linux Journal* and the Chief Security Officer at Purism. He is the author of *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting*, *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks* and *Ubuntu Hacks*, and also a contributor to a number of other O'Reilly books. Rankin speaks frequently on security and open-source software including at BsidesLV, O'Reilly Security Conference, OSCON, SCALE, CactusCon, Linux World Expo and Penguicon. You can follow him at @kylerankin.

If you enjoy retro gaming, there are so many options, it can be tough to know what to get. The choices range from officially sanctioned systems from Nintendo all the way to homemade RetroPie projects like I've covered in *Linux Journal* in the past. Of course, those systems are designed to be permanently attached to a TV.

But, what if you want to play retro games on the road? Although it's true that you could just connect a gamepad to a laptop and use an emulator, there's something to be said for a console that fits in your pocket like the original Nintendo Game Boy. In this article, I describe two different portable DIY retro gaming projects I've built and compare and contrast their features.

Adafruit PiGRRL Zero

The RetroPie project spawned an incredible number of DIY retro consoles due to how easy and cheap the project made it to build

a console out of the widely available and popular Raspberry Pi. Although most of the projects were aimed at home consoles, Adafruit took things a step further and created the PiGRRL project series that combines Raspberry Pis with LCD screens, buttons, batteries and other electronics into a portable RetroPie system that has a similar form factor to the original Game Boy. You buy the kit, print the case and buttons yourself with a 3D printer, and after some soldering, you have a portable console.

The original PiGRRL was based off the Raspberry Pi and was similar in size and shape to the original Game Boy. In the original kit, you also took apart an SNES gamepad, cut the electronics and used it for gamepad electronics. Although you got the benefit of a real SNES gamepad's button feedback, due to that Game Boy form factor, there were no L and R shoulder buttons, and only A and B buttons on the front, so it was aimed at NES and Game Boy games.

The PiGRRL 2 took the original PiGRRL and offered a number of upgrades. First,

Figure 1. PiGRRL 2

it was based on the faster Raspberry Pi 2, which could emulate newer systems like the SNES. It also incorporated its own custom gamepad electronics, so you could get A, B, X and Y buttons in the front, plus L and R buttons in the back, while still maintaining the similar Game Boy form factor.

The issue with the PiGRRL series up to this point for me was form factor and price. The Raspberry Pi B series was large, and most of its components weren't needed for the console and just took up space. Also, the $60 kit did not include the $30–$40 Raspberry Pi itself, so all told, the project ended up running closer to $100. Then the Raspberry Pi Zero came out—a much smaller and much cheaper no-frills Raspberry Pi that still had enough horsepower to emulate games. Shortly after the Raspberry Pi Zero release, Adafruit launched the next revision of its PiGRRL series called the PiGRRL Zero.

Figure 2. PiGRRL Zero

The PiGRRL Zero, to me, was the perfect compromise. First, the kit itself was cheaper at $55, and it included the Raspberry Pi Zero. Second, the form factor was smaller than the regular PiGRRL series—closer to the size of a large gamepad—and due to the smaller form factor, the shoulder buttons could be placed more conveniently at the top of the case.

After I ordered the kit, I fired up my 3D printer and printed out the case and buttons while I waited for everything to arrive. Overall, the case and buttons aren't difficult to print, and they fit together well if your printer is well calibrated. Note: if you want nice rubbery buttons, you'll need a 3D printer that can handle flexible filament like Ninjaflex.

On the project page, Adafruit advises that this is a somewhat advanced electronics project, and I'm inclined to agree. Part of the point of the project is to learn about electronics, and you certainly do. A lot of soldering is involved, with quite a few small headers to solder on in particular. This means you shouldn't plan on removing the Raspberry Pi Zero from the project and re-using it later unless you are willing to desolder quite a bit.

I had a friend with more experience and better equipment come over to make sure I didn't end up with a $55 brick, and although it took most of an afternoon that ran late into the evening, in the end I had a working portable game console. The console itself seemed to perform well, and the Pi Zero can play most SNES games without serious lag. The major issue I had was due to the regular button switches that came with the project. Although Adafruit offers soft-touch buttons switches as well (and I ordered them), they don't actually line up properly with the gamepad board that comes with the project, so you are forced to bend the feet on each soft-touch button, and the overall fit of everything suffers as a result. After trying to set up one, we just fell back to the normal buttons that were provided. Although I like clicky keyboards, clicky buttons aren't that great. For some games, you might not notice much, but for fighting games or precision side-scrollers, you definitely do. You also need to make sure to clean up the case around the shoulder buttons, as they have a tendency to want to stick.

The end result after all of the work involved is that although I found myself playing the console quite a bit when I took it on a vacation, after the vacation, it mostly stayed in a drawer. The relatively long boot time meant I wasn't as apt to power it on for a quick game, and the poor button switches meant that when I did decide to use it, I found myself skipping a lot of my favorite games just because I knew they would be frustrating to play.

ODROID-GO

At first the friend who came over to help me with my PiGRRL Zero was going to order and assemble his own—I even offered to print out a case for him when he did. After he helped with mine though, I could see that his enthusiasm dropped a bit. Months later, I got a message from him with a picture of an NES game playing on this Game Boy-like device with a clear case. It was a new kit he just bought called an ODROID-GO. Since I already had spent time and money on an alternative (even though I didn't use it much), I wasn't exactly ready to solder together yet another kit, but when I looked into it, I changed my mind.

I'm not completely unfamiliar with Hardkernel's ODROID product line. In fact, I use the ODROID-XU4 board for the home NAS server I mentioned in my "Papa's Got a Brand New NAS" article. In addition to the XU4, Hardkernel has released quite a few other boards, and as part of its 10th anniversary, the company decided to create a console kit called ODROID-GO that I found compelling for a few reasons.

The first thing I found compelling was the cost. The entire kit costs $32, and it has everything you need, including a case. The second thing was the fact that it seemed to have good gamepad buttons like you'd see in a regular gamepad controller instead of those odd clicky switches. The next thing was that because this kit is purpose-built to be a game console, the case is a nice quality hard plastic and thin enough to fit in your pocket comfortably. Finally, when I read through the instructions, it looked like although assembly was required, soldering wasn't. In many ways, the ODROID-GO seems to be competing against the Pocket C.H.I.P. project, and it has guides online for how to use it to write your own games and build your own Arduino projects where you can take advantage of the hardware,

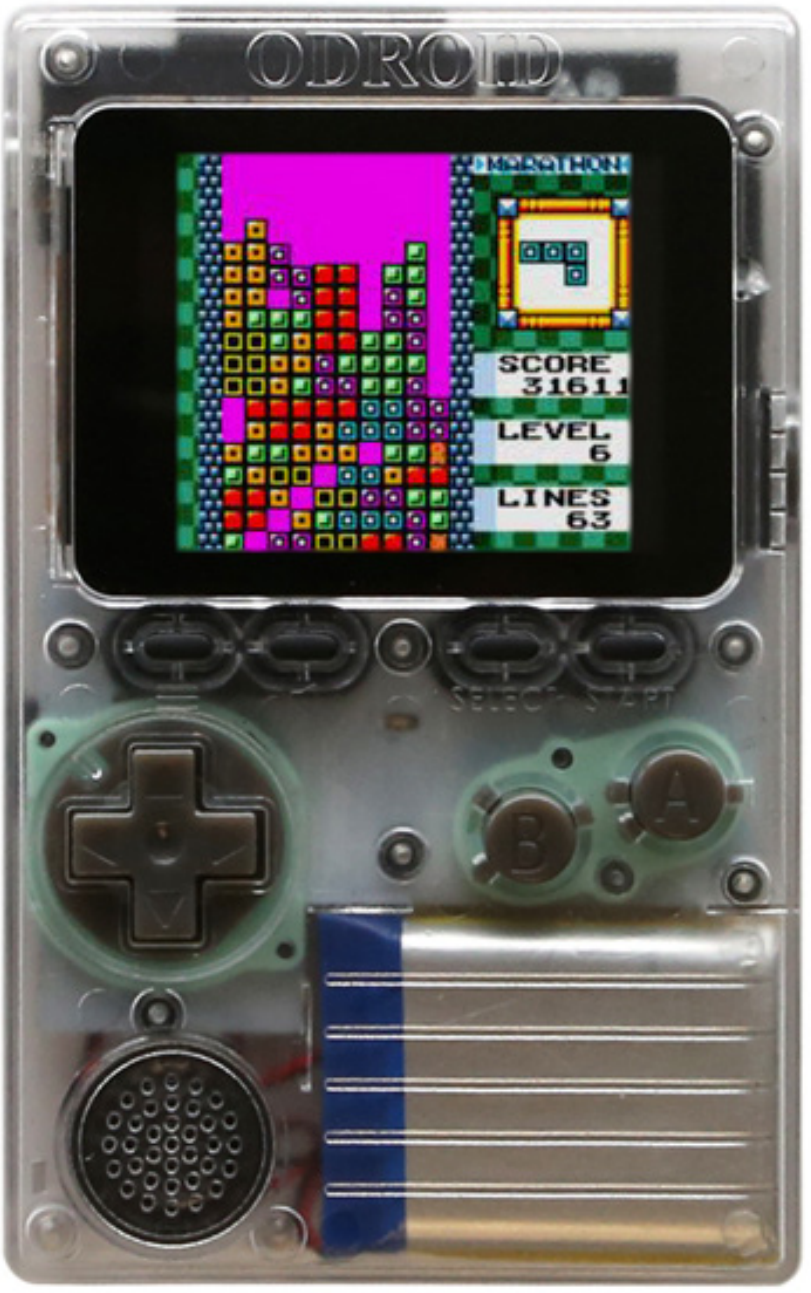


Figure 3. ODROID-GO

like its included WiFi module that otherwise you wouldn't use.

I figured for $32, why not? Although it's true that the ODROID-GO comes to you as a kit and needs assembly, the assembly is very simple and just requires that you do things like place the screen on the board and plug it in, place that in the case, put the

buttons in place, and plug in the battery and speaker. The whole thing took me about 10–15 minutes.

The ODROID-GO uses a custom RetroPie build that boots fast and has a couple additional useful features. The nicest is how well the ODROID-GO integrates save states into the console. Whenever you press the menu button to exit a game back to the main menu, it automatically saves the game state. You then can press the B button to return to the game immediately. When you turn off the console, it still remembers this state, so if you are playing a long game, it's easy to play for a bit, exit to the menu and turn the console off. Then when you are ready to play a bit more, you can turn it back on and press B to pick up where you left off.

Overall, the case is solid, battery life is good, and the gamepad buttons are pretty close to an original Game Boy. Of course, there is one big downside to the ODROID-GO compared to the PiGRRL Zero, and that's the much more sluggish processor. The ODROID-GO can emulate only older consoles, such as the NES, Game Boy, Game Boy Color, Game Boy Advance and Sega Master System. The PiGRRL Zero added SNES, Genesis and even MAME to that list among others, and overall it acted more like a standard RetroPie build.

The other downside to the ODROID-GO has to do with loading ROMs. Like with other console projects, this project uses a microSD card that contains a directory for each console it can emulate. You copy ROMs over to the appropriate directory, and then RetroPie can pick them up. Unfortunately, the board is somewhat picky about the microSD cards you use for it. I had to try a few different cards I had lying around until I found one that it worked well with. Second, I noticed that if you had a large number of files in a particular directory (measured in the hundreds), that it would fail to load (or in reality, it was probably just very slow to read), so if you are used to loading thousands of ROMs for a particular emulator, you will need to prune down your collection for this device.

Conclusion

So, which portable console should you choose? So far, I've found myself using the

ODROID-GO far more than I ever used the PiGRRL Zero, due to the better gamepad, more comfortable and smaller case, and most important, how quickly it boots and resumes a save state. Even though it can play only older consoles compared to the PiGRRL Zero, of the two, I likely will take the ODROID-GO with me when I travel. ■

Resources

- Original PiGRRL Project
- PiGRRL 2 Project
- PiGRRL Zero
- ODROID-GO
- "Super Pi Brothers" by Kyle Rankin, *LJ*, November 2013
- ODROID-XU4 Board
- "Papa's Got a Brand New NAS" by Kyle Rankin, *LJ*, September 2016

Bytes, Characters and Python 2

Reuven M. Lerner teaches Python, data science and Git to companies around the world. His free, weekly "better developers" email list reaches thousands of developers each week; subscribe here. Reuven lives with his wife and children in Modi'in, Israel.

Moving from Python 2 to 3? Here's what you need to know about strings and their role in your upgrade.

By Reuven M. Lerner

An old joke asks "What do you call someone who speaks three languages? Trilingual. Two languages? Bilingual. One language? American."

Now that I've successfully enraged all of my American readers, I can get to the point, which is that because so many computer technologies were developed in English-speaking countries—and particularly in the United States—the needs of other languages often were left out of early computer technologies. The standard established in the 1960s for translating numbers into characters (and back), known as ASCII (the American Standard Code for Information Interchange), took into account all of the letters, numbers and symbols needed to work with English. And that's all that it could handle, given that it was a seven-byte (that is, 128-character) encoding.

If you're willing to ignore accented letters, ASCII can sort of, kind of, work with other languages, as well—but

the moment you want to work with another character set, such as Chinese or Hebrew, you're out of luck. Variations on ASCII, such as ISO-8859-x (with a number of values for "x"), solved the problem to a limited degree, but there were numerous issues with that system.

Unicode gives each character, in every language around the globe, a unique number. This allows you to represent (just about) every character in every language. The problem is how you can represent those numbers using bytes. After all, at the end of the day, bytes are still how data is stored to and read from filesystems, how data is represented in memory and how data is transmitted over a network. In many languages and operating systems, the encoding used is UTF-8. This ingenious system uses different numbers of bytes for different characters. Characters that appear in ASCII continue to use a single byte. Some other character sets (for example, Arabic, Greek, Hebrew and Russian) use two bytes per character. And yet others (such as Chinese and emojis) use three bytes per character.

In a modern programming language, you shouldn't have to worry about this stuff too much. If you get input from the filesystem, the user or the network, it should just come as characters. How many bytes each character needs is an implementation detail that you can (or should be able to) ignore.

Why do I mention this? Because a growing number of my clients have begun to upgrade from Python 2 to Python 3. Yes, Python 3 has been around for a decade already, but a combination of some massive improvements in the most recent versions and the realization that only 18 months remain before Python 2 is deprecated is leading many companies to realize, "Gee, maybe we finally should upgrade."

The major sticking point for many of them? The bytes vs. characters issue.

So in this article, I start looking into what this means and how to deal with it, beginning with an examination of bytes vs. characters in Python 2. In my next article, I plan to look at Python 3 and how the upgrade can be tricky even when you know exactly when you want bytes and when you want characters.

Basic Strings

Traditionally, Python strings are built out of bytes—that is, you can think of a Python string as a sequence of bytes. If those bytes happen to align with characters, as in ASCII, you're in great shape. But if those bytes are from another character set, you need to rethink things a bit. For example:

```
>>> s = 'hello'
>>> len(s)
5
>>> s = 'שלום'  # Hebrew
>>> len(s)
8
>>> s = '你好'  # Chinese
>>> len(s)
6
```

What's going on here? Python 2 allows you to enter whatever characters you want, but it doesn't see the input as characters. Rather, it sees them only as bytes. It's almost as if you were to go to a mechanic and say, "There's a problem with my car", and your mechanic said, "I don't see a car. I see four doors, a windshield, a gas tank, an engine, four wheels and tires", and so forth. Python is paying attention to the individual parts, but not to the character built out of those parts.

Checking the length of a string is one place where you see this issue. Another is when you want to print just part of a string. For example, what's the first character in the Chinese string? It should be the character 你, meaning "you":

```
>>> print(s[0])
�
```

Yuck! That was spectacularly unsuccessful and probably quite useless to any users.

If you want to write a function that reliably prints the first character (not byte) of a

Python 2 string, you could keep track of what language you're using and then look at the appropriate number of bytes. But that's bound to have lots of problems and bugs, and it'll be horribly complex as well.

The appropriate solution is to use Python 2's "Unicode strings". A Unicode string is just like a regular Python string, except it uses characters, rather than bytes. Indeed, Python 2's Unicode strings are just like Python 3's default strings. With Unicode strings, you can do the following:

```
>>> s = u'hello'
>>> len(s)
5
>>> s = u'שלום'  # Hebrew
>>> len(s)
4
>>> s = u'你好'  # Chinese
>>> len(s)
2
>>> print(s[0])
你
```

Terrific! Those are the desired results. You even can make this the default behavior in your Python 2 programs by using one of my favorite modules, `__future__`. The `__future__` module exists so that you can take advantage of features planned for inclusion in later versions, even if you're using an existing version. It allows Python to roll out new functionality slowly and for you to use it whenever you're ready.

One such `__future__` feature is `unicode_literal`. This changes the default type of string in Python to, well, Unicode strings, thus removing the need for a leading "u". For example:

```
>>> from __future__ import unicode_literals
>>> s = 'hello'
```

```
>>> len(s)
5
>>> s = 'שלום'  # Hebrew
>>> len(s)
4
>>> s = '你好'  # Chinese
>>> len(s)
2
>>> print(s[0])
你
```

Now, this doesn't mean that all of you problems are solved. First of all, this `from` statement means that your strings aren't actually strings any more, but rather objects of type `unicode`:

```
>>> type(s)
```

If you have code—and you shouldn't!—that checks to see if s is a string by explicitly checking the type, that code will break following the use of `unicode_literals`. But, other things will break as well.

Reading from Files

For example, let's assume I want to read a binary file—such as a PDF document or a JPEG—into Python. Normally, in Python 2, I can do so using strings, because a string can contain any bytes, in any order. However, Unicode is quite strict about which bytes represent characters, in no small part because the bytes whose eighth (highest) bit is active are part of a larger character and cannot stand on their own.

Here's a short program that I wrote to read and print such a file:

```
>>> filename = 'Files/unicode.txt'
>>> from __future__ import unicode_literals
>>> for one_line in open(filename):
```

```
...  for index, one_char in one_line:
...     print("{0}: {1}".format(index, one_char))
```

When I run it, this program crashes:

```
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xd7
 ↪in position 0: ordinal not in range(128)
```

What's the problem? Well, it's still reading the file using bytes, rather than characters. After reading the current line from the filesystem, Python tries to create a string. But, it can't resolve the conflict between the bytes it received and the Unicode it must create as a string.

In other words, while it has managed to make Python's strings Unicode-compliant, there are a bunch of things in the general Python environment that aren't Unicode-aware or friendly.

You can solve this problem by using the `codecs` module, and the `open` method it provides, telling it which encoding you want to use when reading from the file:

```
>>> import codecs
>>> for one_line in codecs.open(filename, encoding='utf-8'):
...  for index, one_char in enumerate(one_line):
...     print("{0}: {1}".format(index, one_char))
```

To summarize, you can make all of Python's strings Unicode-compliant if you use `unicode_literals`. But the moment you do that, you run into the potential problem of getting data in bytes from the user, network or filesystem, and having an error. Although this seems like a really tempting way to deal with the whole Unicode issue, I suggest that you go the `unicode_literals` route only if you have a really good test suite, and if you're sure that all the libraries you use know what to do

when you change strings in this way. You will quite possibly be surprised to find that although many things work fine, many others don't.

The bytes Type

When talking about strings and Unicode, there's another type that should be mentioned as well: the "bytestring", aka the `bytes` type. In Python 2, `bytes` is just an alias for `str`, as you can see here in this Python shell that has not imported `unicode_literals`:

```
>>> s = 'abcd'
>>> type(s) == bytes
True
>>> str == bytes
True
>>> bytes(1234)
'1234'
>>> type(bytes(1234))
<type 'str'>
>>>
```

In other words, although Python strings generally are thought of as having type `str`, they equally can be seen as having type `bytes`. Why would you care? Because it allows you to separate strings that use bytes from strings that use Unicode already in Python 2, and to continue that explicit difference when you get to Python 3 as well.

I should add that a very large number of developers I've met (and taught) who use Python 2 are unaware that byte strings even exist. It's much more common to talk about them in Python 3, where they serve as a counterpart to the Unicode-aware strings.

Just as Unicode strings have a "u" prefix, bytestrings have a "b" prefix. For example:

```
>>> b'abcd'
'abcd'
```

```
>>> type(b'abcd')
<type 'str'>
```

In Python 2, you don't need to talk about byte strings explicitly, but by using them, you can make it very clear as to whether you're using bytes or characters.

This raises the question of how you can move from one world to the other. Let's say, for example, you have the Unicode string for "Hello" in Chinese, aka ?| 好. How can you get a bytestring containing the (six) bytes? You can use the `str.encode` method, which returns a bytestring (aka a Python 2 string), containing six bytes:

```
s.encode('utf-8')
```

Somewhat confusingly (to me, at least), you "encode" from Unicode to bytes, and you indicate the encoding in which the string is storing things. Regardless, you then get:

```
>>> s.encode('utf-8')
'\xe4\xbd\xa0\xe5\xa5\xbd'
```

Now, why would you want to turn a Unicode string into bytes? One reason is that you want to write the bytes to a file, and you don't want to use `codecs.open` in order to do so. (Don't try to write Unicode strings to a file opened in the usual way, with "open").

What if you want to do the opposite, namely take a bunch of bytes and turn them into Unicode? As you can probably guess, the opposite is performed via the `str.decode` method:

```
>>> b.decode('utf-8')
u'\u4f60\u597d'
```

Once again, you indicate the encoding that should be used, and the result is a

Unicode string, which you see here represented with the special `\u` syntax in Python. This syntax allows you to specify any Unicode character by its unique "code point". If you print it out, you can see how it looks:

```
>>> print(b.decode('utf-8'))
你好
```

Summary

Python 2 is going to be deprecated in 2020, and many companies are starting to look into how to upgrade. A major issue for them will be the strings in their programs. This article looks at strings, Unicode strings and bytestrings in Python 2, paving the way to cover these same issues in Python 3, and how to handle upgrades, in my next article. ■

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

Decentralized Certificate Authority and Naming

Free and open source contributors only:

handshake.org/signup

Globbing and Regex: So Similar, So Different

Shawn Powers is Associate Editor here at *Linux Journal*, and has been around Linux since the beginning. He has a passion for open source, and he loves to teach. He also drinks too much coffee, which often shows in his writing.

Grepping is awesome, as long as you don't glob it up! This article covers some grep and regex basics.

By Shawn Powers

There are generally two types of coffee drinkers. The first type buys a can of pre-ground beans and uses the included scoop to make their automatic drip coffee in the morning. The second type picks single-origin beans from various parts of the world, accepts only beans that have been roasted within the past week and grinds those beans with a conical burr grinder moments before brewing in any number of complicated methods. Text searching is a bit like that.

For most things on the command line, people think of *.* or *.txt and are happy to use file globbing to select the files they want. When it comes to grepping a log file, however, you need to get a little fancier. The confusing part is when the syntax of globbing and regex overlap. Thankfully, it's not hard to figure out when to use which construct.

Globbing

The command shell uses globbing for filename completion. If you type something like `ls *.txt`, you'll get a list of all the files that end in .txt in the current directory. If you do `ls R*.txt`, you'll get all the files that start with capital R and have the .txt extension. The asterisk is a wild card that lets you quickly filter which files you mean.

You also can use a question mark in globbing if you want to specify a single character. So, typing `ls read??.txt` will list readme.txt, but not read.txt. That's different from `ls read*.txt`, which will match both readme.txt and read.txt, because the asterisk means "zero or more characters" in the file glob.

Here's the easy way to remember if you're using globbing (which is very simple) vs regular expressions: globbing is done to filenames by the shell, and regex is used for searching text. The only frustrating exception to this is that sometimes the shell is too smart and conveniently does globbing when you don't want it to—for example:

```
grep file* README.TXT
```

In most cases, this will search the file README.TXT looking for the regular expression `file*`, which is what you normally want. But if there happens to be a file in the current folder that matches the `file*` glob (let's say filename.txt), the shell will assume you meant to pass that to `grep`, and so grep actually will see:

```
grep filename.txt README.TXT
```

Gee, thank you so much Mr. Shell, but that's not what I wanted to do. For that reason, I recommend always using quotation marks when using `grep`. 99% of the time you won't get an accidental glob match, but that 1% can be infuriating. So when using `grep`, this is much safer:

```
grep "file*" README.TXT
```

Because even if there is a filename.txt, the shell won't substitute it automatically.

So, globs are for filenames, and regex is for searching text. That's the first thing to understand. The next thing is to realize that similar syntax means different things.

Glob and Regex Conflicts

I don't want this article to become a super in-depth piece on regex; rather, I want you to understand simple regex, especially as it conflicts with globbing. Table 1 shows a few of the most commonly confused symbols and what they mean in each case.

Table 1. Commonly Used Symbols

Special Character	Meaning in Globs	Meaning in Regex
*	zero or more characters	zero or more of the character it follows
?	single occurrence of any character	zero or one of the character it follows but not more than 1
.	literal "." character	any single character

To add insult to injury, you might be thinking about globs when you use `grep`, but just because you get the expected results doesn't mean you got the results for the correct reason. Let me try to explain. Here is a text file called filename.doc:

```
The fast dog is fast.
The faster dogs are faster.
A sick dog should see a dogdoc.
This file is filename.doc
```

If you type:

```
grep "fast*" filename.doc
```

The first two lines will match. Whether you're thinking globs or regex, that makes

sense. But if you type:

```
grep "dogs*" filename.doc
```

The first three lines will match, but if you're thinking in globs, that doesn't make sense. Since `grep` uses regular expressions (regex) when searching files, the asterisk means "zero or more occurrences of the previous character", so in the second example, it matches dog and dogs, because having zero "s" characters matches the regex.

And let's say you typed this:

```
grep "*.doc" filename.doc
```

This will match the last two lines. The asterisk doesn't actually do anything in this command, because it's not following any character. The dot in regex means "any character", so it will match the ".doc", but it also will match "gdoc" in "dogdoc", so both lines match.

The moral of the story is that `grep` never uses globbing. The only exception is when the shell does globbing before passing the command on to `grep`, which is why it's always a good idea to use quotation marks around the regular expression you are trying to `grep` for.

Use `fgrep` to Avoid Regex

If you don't want the power of regex, it can be very frustrating. This is especially true if you're actually looking for some of the special characters in a bunch of text. You can use the `fgrep` command (or `grep -F`, which is the same thing) in order to skip any regex substitutions. Using `fgrep`, you'll search for exactly what you type, even if they are special characters. Here is a text file called file.txt:

```
I really hate regex.
All those stupid $, {}, and \ stuff ticks me off.
Why can't text be text?
```

If you try to use regular `grep` like this:

```
grep "$," file.txt
```

You'll get no results. That's because the "$" is a special character (more on that in a bit). If you'd like to `grep` for special characters without escaping them, or knowing the regex code to get what you want, this will work fine:

```
grep -F "$," file.txt
```

And, `grep` will return the second line of the text file because it matches the literal characters. It's possible to build a regex query to search for special characters, but it can become complicated quickly. Plus, `fgrep` is much, much faster on a large text file.

Some Simple, Useful Regex

Okay, now that you know when to use globbing and when to use regular expressions, let's look at a bit of regex that can make grepping much more useful. I find myself using the caret and dollar sign symbols in regex fairly often. Caret means "at the beginning of the line", and dollar sign means "at the end of the line". I used to mix them up, so my silly method to remember is that a farmer has to plant carrots at the beginning of the season in order to sell them for dollars at the end of the season. It's silly, but it works for me!

Here's a sample text file named file.txt:

```
chickens eat corn
corn rarely eats chickens
people eat chickens and corn
chickens rarely eat people
```

If you were to type:

```
grep "chickens" file.txt
```

you will get all four lines returned, because “chickens” is in each line. But if you add some regex to the mix:

```
grep "^chickens" file.txt
```

you’ll get both the first and fourth line returned, because the word “chickens” is at the beginning of those lines. If you type:

```
grep "corn$" file.txt
```

you will see the first and third lines, because they both end with “corn”. However, if you type:

```
grep "^chickens.*corn$" file.txt
```

you will get only the first line, because it is the only one that begins with chickens and ends with corn. This example might look confusing, but there are three regular expressions that build the search. Let’s look at each of them.

First, `^chickens` means the line must start with chickens.

Second, `.*` means zero or more of any character, because remember, the dot means any character, and the asterisk means zero or more of the previous character.

Third, `corn$` means the line must end with corn.

When you’re building regular expressions, you just mush them all together like that in a long string. It can become confusing, but if you break down each piece, it makes sense. In order for the entire regular expression to match, all of the pieces must match. That’s why only the first line matches the example regex statement.

A handful of other common regex characters are useful when grepping text files. Remember just to mush them together to form the entire regular expression:

- `\` — the backslash negates the "special-ness" of special characters, which means you actually can search for them with regex. For example, `\$` will search for the $ character, instead of looking for the end of a line.

- `\s` — this construct means "whitespace", which can be a space or spaces, tabs or newline characters. To find the word pickle surrounded by whitespace, you could search for `\spickle\s`, and that will find "pickle" but not "pickles".

- `.*` — this is really just a specific use of the asterisk, but it's a very common combination, so I mention it here. It basically means "zero or more of any characters", which is what was used in the corn/chicken example above.

- `|` — this means "or" in regex. So `hi|hello` will match either "hi" or "hello". It's often used in parentheses to separate it from other parts of the regular expression. For example, `(F|f)rankfurter` will search for the word frankfurter, whether or not it's capitalized.

- `[]` — brackets are another way to specify "or" options, but they support ranges. So the regex `[Ff]rankfurter` is the same as the above example. Brackets support ranges though, so `^[A-Z]` will match any line that starts with a capital letter. It also supports numbers, so `[0-9]$` will match any line that ends in a digit.

Your Mission

You can do far more complicated things with regular expressions. These basic building blocks are usually enough to get the sort of text you need out of a log file. If you want to learn more, by all means, either do some googling on regex, or get a book explaining all the nitty-gritty goodness. If you want me to write more about it, send a note to ljeditor@linuxjournal.com and let me know.

I really, really encourage you to practice using regex. The best way to learn is to do, so make a few text files and see if the regex statements you create give you the results you expect. Thankfully, `grep` highlights the "match" it finds in the line it returns. That means if you're getting more results than you expect, you'll see why the regex

matched more than you expected, because `grep` will show you.

The most important thing to remember is that `grep` doesn't do globbing—that wildcard stuff is for filenames on the shell only. Even if globbing with `grep` seems to work, it's probably just coincidence (look back at the dog/dogs example here if you don't know what I'm talking about). Have fun grepping! ■

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

Creating the Concentration Game PAIRS with Bash, Part II

Dave Taylor has been hacking shell scripts on Unix and Linux systems for a really long time. He's the author of *Learning Unix for Mac OS X* and *Wicked Cool Shell Scripts*. You can find him on Twitter as @DaveTaylor, and you can reach him through his tech Q&A site Ask Dave Taylor.

Dave finishes up the PAIRS concentration game, only to realize it's too hard to solve!

By Dave Taylor

In my last article, I tossed away my PC card and talked about how I was a fan of the British colonial-era writer Rudyard Kipling. With that in mind, I do appreciate that you're still reading my column.

I was discussing the memory game that the British spy plays with the orphan boy Kim in the book of the same name. The game in question involves Kim being shown a tray of stones of various shapes, sizes and colors. Then it's hidden, and he has to recite as many patterns as he can recall.

The card game Concentration is clearly inspired by the same pattern memorization game, and it's considerably easier to

set up: shuffle a deck of cards, place them face down in a grid, then flip pairs to find matches. In the beginning, it's just guessing, of course, but as the game proceeds, it becomes more about spatial memory than luck. Someone with an eidetic memory always will win.

Using letters makes things easy, so I suggested a row, column, notational convention like this:

```
    1   2   3   4   5   6   7   8   9   10  11  12  13
1: [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-]
2: [-] [-] [-] [A] [-] [-] [-] [-] [-] [-] [-] [-] [-]
3: [-] [-] [-] [-] [-] [-] [-] [-] [E] [-] [-] [-] [-]
4: [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [-] [Z]
```

You can represent uppercase letters as a shell array like this:

```
declare -a letters=(A B C D E F G H I J K L M N O P Q R
                    S T U V W X Y Z)
```

Unfortunately, Bash doesn't support multidimensional arrays, so you're going to have to represent the grid as a one-dimensional array. It's not too hard though, because the grid is straightforward. Here's an index formula if `firstvalue` is the first digit and `rest` is the remainder of the index value:

```
index=$(( ( ( $firstvalue - 1 ) * 13 ) + $rest ))
```

The letter "E" in the above grid, at 3,9, would show up in the array as ((3-1)*13)+9 or slot 35.

Shuffle Those Values

The script from my last article already initializes everything in sequential order and defaults to 2 * 13 slots (for simplicity in debugging). The work of the script is really in the shuffle, but it turns out that there's a pretty elegant little shuffle algorithm (shown

in a kind of sloppy C for illustrative purposes) floating around the internet that can be tapped for this task:

```
shuffle {
   for (i = n-1; i > 0; i-) {
     int j = rand() % (i+1);
     swap( array[i], array[j]);
   }
}
```

Translating this into a shell script and using better variable names, here's what I created:

```
shuffle ()
{
   # shuffle board with $1 * 13 values

   totalvalues=$(( $1 * 13 ))

   index=$totalvalues

   while [ $index -gt 1 ] ; do

     randval=$(( ( $RANDOM % $index ) + 1 ))

     # swapping value pair

     temp=${board[$index]}
     board[$index]=${board[$randval]}
     board[$randval]=$temp

     index=$(( $index - 1 ))
   done
}
```

Instead of having a separate function for the value swap, I just went ahead and dropped that into the function itself. It's faster and also lets you sidestep the dereference hassle neatly.

Here's what happens when you initialize the grid, shuffle it, then display it on screen (and yes, I changed the "[]" to "<>" to make it more visually interesting):

```
    1   2   3   4   5   6   7   8   9   10  11  12  13
1: <V> <X> <M> <R> <C> <F> <K> <O> <U> <H> <T> <Q> <L>
2: <A> <G> <B> <N> <J> <Y> <P> <W> <Z> <E> <D> <I> <S>
```

Of course, 26 grid spots equals exactly the number of letters in the alphabet, so there are exactly zero pairs. That's not much fun as games go, but what if you request a four-line grid?

```
    1   2   3   4   5   6   7   8   9   10  11  12  13
1: <G> <J> <A> <K> <P> <L> <B> <O> <I> <X> <Y> <N> <F>
2: <Y> <C> <Z> <O> <G> <D> <T> <N> <V> <D> <H> <E> <U>
3: <W> <C> <R> <Q> <M> <B> <E> <K> <F> <I> <T> <Q> <R>
4: <U> <Z> <P> <H> <S> <W> <L> <J> <M> <X> <V> <S> <A>
```

A few pairs jump out, like 2,13 and 4,1 for the "U" values, but remember, the game is going to hide all of this, and it's your job to guess these pairs.

Ah, it's suddenly not so easy, eh?

Tracking What's Been Guessed

Now that the grid is being created and shuffled correctly, the next step is to differentiate between spots that have been guessed correctly and those that are still unknown. You could do that with a parallel array, but why go through the hassle? Instead, initialize every value to have a dash as its first character and then remove it once guessed.

The display function now can test a value to see if it's a "negative" letter. If so, it'll display a "-" instead of the actual value. Now the initial grid looks like this:

```
    1   2   3   4   5   6   7   8   9   10  11  12  13
1: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
2: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
3: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
4: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
```

What about entering your guess for the location of a given pair? I'm going to make things harder by not showing the values in a grid but rather just displaying them directly.

Enter a grid value as row, col, then split it into those values and multiply it out to a unique array index. It's complicated, but if you read `$slot1` and `$slot2` as the input values from the user, the analysis loop is this:

```
row1=$( echo $slot1 | cut -c1 )
col1=$( echo $slot1 | cut -d, -f2 )
row2=$( echo $slot2 | cut -c1 )
col2=$( echo $slot2 | cut -d, -f2 )

index1=$(( ( $row1 - 1) * 13 + $col1 ))
index2=$(( ( $row2 - 1) * 13 + $col2 ))

val1=${board[$index1]}
val2=${board[$index2]}
```

There's a woeful lack of error-checking here, but that's something I like to add afterward, once I get the core algorithm functional.

Armed with `$val1` and `$val2` above, testing to see if you have a match is easy:

```
if [ $val1 = $val2 ] ; then
```

```
  echo "You've got a match. Nicely done!"
  board[$index1]=${val1:1:1}
  board[$index1]=${val2:1:1}
  unsolved=$(( $unsolved - 2 ))
else
  echo "No match. $row1,$col1 = ${val1:1:1} and \
       $row2,$col2 = ${val2:1:1}."
fi
```

Did you notice `$unsolved` in the matching conditional code? That's how you can keep track of whether the grid has been solved.

So Let's Give It a Try!

With all this code in place, let's give it a whirl:

```
Welcome to PAIRS. Your mission is to identify matching letters
in the grid. Good luck. If you give up at any point, just use
q to quit.


Enter a pair of values in row,col format : 1,1 4,1
No match, but 1,1 = P and 4,1 = A.

    1   2   3   4   5   6   7   8   9   10  11  12  13
1: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
2: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
3: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
4: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->

Enter a pair of values in row,col format : 2,1 3,1
No match, but 2,1 = Z and 3,1 = B.
```

```
    1   2   3   4   5   6   7   8   9   10  11  12  13
1: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
2: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
3: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
4: <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <-> <->
```

I'm basically done with the program at this point, and I'm realizing something. This is really hard to solve as a game.

Hacks and Mods Here's an exercise for you, dear reader: this is generating 26 possible values, the letters A–Z, which requires a minimum grid of 52 slots. That's a lot! Modify it to work with single digits, and then adjust the grid dimensions appropriately. For example, 20 slots can be portrayed in a 4 x 5 grid. For sure, 19 possibilities for the match of a revealed value is a lot easier than 51 possibilities.

Have fun with this, and grab the full script below or from here. Let me know how you modify this to make it more entertaining, interesting or just make it easier!

The Full Script

```
#!/bin/sh

# PAIR - a simple matching game, implmemented as a linear array

# Usage: PAIR rowcount
#    note: rowcount must be even and specifies how many 13-slot
#    rows are created
#    the default value is 2 rows of 13 values

declare -a board
declare -a letters=(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
rows=4                  # default # of 13 slot rows
```

```
initialize ()
{
   # given number of rows, initialize the board with all blank values

   count=1   maxcount=$1

   while [ $count -le $maxcount ]
   do
     addon=$(( 13 * ( $count - 1 ) ))

     for slot in {1..13}
     do
       index=$(( $addon + $slot ))
       letter=$(( $index % 26 ))        # unshuffled value

       board[ $index ]="-${letters[$letter]}" # unguessed start with '-'
     done
     count=$(( $count + 1 ))
   done
}

shuffle ()
{
   # given the board[] array with $1 * 13 entries, shuffle the contents

   totalvalues=$(( $1 * 13 ))

   index=$totalvalues

   while [ $index -gt 1 ] ; do

     randval=$(( ( $RANDOM % $index ) + 1 ))
```

```
    # swapping value pair

    temp=${board[$index]}
    board[$index]=${board[$randval]}
    board[$randval]=$temp

    index=$(( $index - 1 ))
  done
}

showgrid ()
{
  # show our grid. This means we need to display $1 x 13 slots with
  # labels
  #     rows is grabbed from the global var 'rows'

  count=1

  echo "    1   2   3   4   5   6   7   8   9   10  11  12  13"

  while [ $count -le $rows ]
  do
    /bin/echo -n "$count: "
    addon=$(( 13 * ( $count -1 ) ))

    for slot in {1..13}
    do
      index=$(( $slot + $addon ))
      value=${board[$index]}
      if [ ${value:0:1} != '-' ] ; then   # matched!
        /bin/echo -n "<${board[$index]}> "
      else
        /bin/echo -n "<-> "     # still unknown
```

```
      fi
    done
    echo ""
    count=$(( $count + 1 ))
  done

}

###################################

if [ $# -gt 0 ] ; then
  rows=$1
fi

if [ $(( $rows % 4 )) -ne 0 ] ; then
  echo "Ooops! Please only specify a multiple of 4 as the number
of rows (4, 8, 12, etc)"
  exit 1
fi

slot1=slot2="X"                 # start with a non-empty value
unsolved=$(( $rows * 13 ))
maxvalues=$unsolved             # parameter testing

echo "Welcome to PAIRS. Your mission is to identify matching letters
in the grid."
echo "Good luck. If you give up at any point, just use q to quit."
echo ""

initialize $rows

shuffle $rows
```

```
showgrid

while [ $unsolved -gt 0 ] ; do
  echo ""
  /bin/echo -n "Enter a pair of values in row,col format : "
  read slot1 slot2

  if [ "$slot1" = "" -o "$slot2" = "" ] ; then
    echo "bye."
    exit 1
  fi

  row1=$( echo $slot1 | cut -c1 )
  col1=$( echo $slot1 | cut -d, -f2 )
  row2=$( echo $slot2 | cut -c1 )
  col2=$( echo $slot2 | cut -d, -f2 )

  index1=$(( ( $row1 - 1) * 13 + $col1 ))
  index2=$(( ( $row2 - 1) * 13 + $col2 ))

  if [ $index1 -lt 0 -o $index1 -gt $maxvalues -o $index2 -lt
 ↪0 -o $index2 -gt $maxvalues ] ; then
    echo "bad input, not a valid value"
    exit 1
  fi

  val1=${board[$index1]}
  val2=${board[$index2]}

  if [ $val1 = $val2 ] ; then
    echo "You've got a match. Nicely done!"
    board[$index1]=${val1:1:1}
    board[$index1]=${val2:1:1}
```

```
    unsolved=$(( $unsolved - 2 ))
  else
    echo "No match, but $row1,$col1 = ${val1:1:1} and $row2,$col2 =
↪${val2:1:1}."
  fi

  echo ""
  showgrid

done

exit 0
```

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

What's New in Kernel Development

By Zack Brown

Zack Brown is a tech journalist at *Linux Journal* and *Linux Magazine*, and is a former author of the "Kernel Traffic" weekly newsletter and the "Learn Plover" stenographic typing tutorials. He first installed Slackware Linux in 1993 on his 386 with 8 megs of RAM and had his mind permanently blown by the Open Source community. He is the inventor of the *Crumble* pure strategy board game, which you can make yourself with a few pieces of cardboard. He also enjoys writing fiction, attempting animation, reforming Labanotation, designing and sewing his own clothes, learning French and spending time with friends'n'family.

Moving Compiler Dependency Checks to Kconfig

The Linux kernel config system, **Kconfig**, uses a macro language very similar to the make build tool's macro language. There are a few differences, however. And of course, make is designed as a general-purpose build tool while Kconfig is Linux-kernel-specific. But, why would the kernel developers create a whole new macro language so closely resembling that of an existing general-purpose tool?

One reason became clear recently when **Linus Torvalds** asked developers to add an entirely new system of dependency checks to the Kconfig language, specifically testing the capabilities of the **GCC compiler**.

It's actually an important issue. The Linux kernel wants to support as many versions of GCC as possible—so long as doing so would not require too much insanity in the kernel code itself—but different versions of GCC support different features. The GCC developers always are tweaking and adjusting, and GCC releases also sometimes have bugs that need to be worked around. Some Linux kernel features can only be built using one

version of the compiler or another. And, some features build better or faster if they can take advantage of various GCC features that exist only in certain versions.

Up until this year, the kernel build system has had to check all those compiler features by hand, using many hacky methods. The art of probing a tool to find out if it supports a given feature dates back decades and is filled with insanity. Imagine giving a command that you know will fail, but giving it anyway because the specific manner of failure will tell you what you need to know for a future command to work. Now imagine hundreds of hacks like that in the Linux kernel build system.

Part of the problem with having those hacky checks in the build system is that you find out about them only during the build—not during configuration. But since some kernel features require certain GCC versions, the proper place to learn about the GCC version is at config time. If the user's compiler doesn't support a given feature, there's no reason to show that feature in the config system. It should just silently not exist.

Linus requested that developers migrate those checks into the Kconfig system and regularize them into the macro language itself. This way, kernel features with particular GCC dependencies could identify those dependencies and then show up or not show up at config time, according to whether those dependencies had been met.

That's the reason simply using `make` wouldn't work. The config language had to represent the results of all those ugly hacks in a friendly way that developers could make use of.

The code to do this has been added to the kernel tree, and **Masahiro Yamada** recently posted some documentation to explain how to use it. The docs are essentially fine, although the code will gradually grow and grow as new versions of GCC require new hacky probes.

It's actually not so easy to know what should and should not go into the config system. If we're probing for GCC versions, why not probe for hardware peripherals as well? Why leave this for the kernel to do at runtime? It's not necessarily clear. In

fact, it's an open debate that ultimately could swing either way. Dumping all this GCC-detection code into Kconfig may make Kconfig better able to handle other such dumps that previously would have seemed like too much. The only way we'll really know is to watch how the kernel developers probe Linus to see what he'll accept and what would be going too far.

Bug Hunting Inlined Code

The Linux kernel has various debugging tools. One is the kernel **function tracer**, which traces function calls, looking for bad memory allocations and other problems.

Changbin Du from **Intel** recently posted some code to increase the range of the function tracer by increasing the number of function calls that were actually compiled into the kernel. Not all function calls are ever actually compiled—some are "inlined", a C feature that allows the function code to be copied to the location that calls it, thus letting it run faster. The downside is that the compiled binary grows by the number of copies of that function it has to store.

But, not all inlined functions are specifically intended by the developers. The GNU C Compiler (GCC) also will use its own algorithms to decide to inline a wide array of functions. Whenever it does this in the Linux kernel, the function tracer has nothing to trace.

Changbin's code still would allow functions to be inlined, but only if they explicitly used the `inline` keyword of the C language. All other inlining done by GCC itself would be prevented. This would produce less efficient code, so Changbin's code never would be used in production kernel builds. But on the other hand, it would produce code that could be far more thoroughly examined by the function tracer, so Changbin's code would be quite useful for kernel developers.

As soon as he posted the patches, bug reports popped up all over the kernel in functions that GCC had been silently inlining. As a result, absolutely nobody had any objections to this particular patch.

There were, however, some odd false positives produced by the function tracer, claiming that it had found bugs that didn't actually exist. This gave a few kernel developers a slight pause, and they briefly debated how to eliminate those false positives, until they realized it didn't really matter. They reasoned that the false positives probably indicated a problem with GCC, so the GCC people would want to be able to see those false positives rather than have them hidden away behind workarounds.

That particular question—what is a kernel issue versus a GCC issue—is potentially explosive. It didn't lead anywhere this time, but in the past, it has led to bitter warfare between the kernel people and the GCC people. One such war was over GCC's failure to support **Pentium** processors and led to a group of developers forking GCC development into a competing project, called **egcs**. The fork was very successful, and it began to be used in mainstream Linux distributions instead of GCC. Ultimately, the conflict between the two branches was resolved only after the egcs code was merged into the GCC main branch, and future GCC development was handed over to the egcs team of developers in 1999.

Support for a LoRaWAN Subsystem

Sometimes kernel developers find themselves competing with each other to get their version of a particular feature into the kernel. But sometimes developers discover they've been working along very similar lines, and the only reason they hadn't been working together was that they just didn't know each other existed.

Recently, **Jian-Hong Pan** asked if there was any interest in a **LoRaWAN** subsystem he'd been working on. LoRaWAN is a commercial networking protocol implementing a low-power wide-area network (LPWAN) allowing relatively slow communications between things, generally phone sensors and other internet of things devices. Jian-Hong posted a link to the work he'd done so far: https://github.com/starnight/LoRa/tree/lorawan-ndo/LoRaWAN.

He specifically wanted to know "should we add the definitions into corresponding kernel header files now, if LoRaWAN will be accepted as a subsystem in Linux?" The

reason he was asking was that each definition had its own number. Adding them into the kernel would mean the numbers associated with any future LoRaWAN subsystem would stay the same during development.

However, **Marcel Holtmann** explained the process:

> When you submit your LoRaWAN subsystem to netdev for review, include a patch that adds these new address family definitions. Just pick the next one available. There will be no pre-allocation of numbers until your work has been accepted upstream. Meaning, that the number might change if other address families get merged before yours. So you have to keep updating. glibc will eventually follow the number assigned by the kernel.

Meanwhile, **Andreas Färber** said he'd been working on supporting the same protocol himself and gave a link to his own proof-of-concept repository: https://github.com/afaerber/lora-modules.

On learning about Andreas' work, Jian-Hong's response was, "Wow! Great! I get new friends :)"

That's where the public conversation ended. The two of them undoubtedly have pooled their energies and will produce a new patch, better than either of them might have done separately.

It's interesting to me the way some projects are more amenable to merging together than others. It seems to have less to do with developer personalities, and more to do with how much is at stake in a given area of the kernel. A new load-balancing algorithm may improve the user experience for some users and worsen it for others, depending on their particular habits. How can two developers resolve their own questions about which approach is better, given that it's not feasible to have lots of different load balancers all in the kernel together? Wars have gone on for years over such issues. On the other hand, supporting a particular protocol or a particular peripheral device is much easier. For one thing, having several competing drivers

in the kernel is generally not a problem, at least in the short term, as long as they don't dig too deeply into core kernel behaviors. Developers can test their ideas on a live audience and see what really works better and what doesn't. When that sort of freedom disappears, the closer you get to real speed issues and real security issues.

Support for a GNSS and GPS Subsystem

Recently, there was a disagreement over whether a subsystem really addressed its core purpose or not. That's an unusual debate to have. Generally developers know if they're writing support for one feature or another.

In this particular case, **Johan Hovold** posted patches to add a **GNSS** subsystem (Global Navigation Satellite System), used by **GPS** devices. His idea was that commercial GPS devices might use any input/output ports and protocols—serial, USB and whatnot—forcing user code to perform difficult probes in order to determine which hardware it was dealing with. Johan's code would unify the user interface under a /dev/gnss0 file that would hide the various hardware differences.

But, **Pavel Machek** didn't like this at all. He said that there wasn't any actual GNSS-specific code in Johan's GNSS subsystem. There were a number of GPS devices that wouldn't work with Johan's code. And, Pavel felt that at best Johan's patch was a general power management system for serial devices. He felt it should not use names (like "GNSS") that then would be unavailable for a "real" GNSS subsystem that might be written in the future.

However, in kernel development, "good enough" tends to trump "good but not implemented". Johan acknowledged that his code didn't support all GPS devices, but he said that many were proprietary devices using proprietary interfaces, and those companies could submit their own patches. Also, Johan had included two GPS drivers in his patch, indicating that even though his subsystem might not contain GNSS-specific code, it was still useful for its intended purpose—regularizing the GPS device interface.

The debate went back and forth for a while. Pavel seemed to have the ultimate truth

on his side—that Johan's code was at best misnamed, and at worst, incomplete and badly structured. Although Johan had real-world usefulness on his side, where something like his patch had been requested by other developers for a long time and solved actual problems confronted by people today.

Finally **Greg Kroah-Hartman** put a stop to all debate—at least for the moment—by simply accepting the patch and feeding it up to Linus Torvalds for inclusion in the main kernel source tree. He essentially said that there was no competing patch being offered by anyone, so Johan's patch would do until anything better came along.

Pavel didn't want to give up so quickly, and he tried at least to negotiate a name change away from "GNSS", so that a "real" GNSS subsystem might still come along without a conflict. But with his new-found official support, Johan said, "This is the real gnss subsystem. Get over it."

It's an odd situation. On the other hand, the Linux kernel generally avoids trying to stake out territory for infrastructure that doesn't yet exist. It may be that Johan's non-GNSS GNSS subsystem will be all that's ever needed for GPS device support. In which case, why assume it will ever be more complicated than this? Famous last words.

Note: if you're mentioned in this article and want to send a response, please send a message with your response text to ljeditor@linuxjournal.com, and we'll run it in the next Letters section and post it on the website as an addendum to the original article. ■

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

DEEP DIVE

GAMING

Crossing Platforms: a Talk with the Developers Building Games for Linux

Games for Linux are booming like never before. The revolution comes courtesy of cross-platform dev tools, passionate programmers and community support.

By K.G. Orphanides

In the last five years, the number of mainstream games released for Linux has increased dramatically, with thousands of titles now available. These range from major AAA releases, such as *Civilization VI* and *Deus Ex: Mankind Divided*, to breakout indie hits like *Night in the Woods*.

For this article, I spoke to different developers and publishers to discover the shape of the Linux games market and find out what's driving its prodigious growth.

Figure 1. Multi-award-winning comedy adventure game *Night in the Woods* is one of many games simultaneously released on Linux, macOS and Windows, thanks to development tools that can build for all three platforms.

Why Develop Games for Linux?

Support for Linux has boomed with the introduction of cross-platform development tools that make it comparatively easy to release titles on multiple operating systems. Perhaps more important, almost all the developers I spoke to personally support the Open Source movement, even if their games are proprietary.

For Zack Johnson, creative director of asymmetric's stick-figure comedy RPG, *West of Loathing*, the game's origins as a spin-off from popular browser game *Kingdom of Loathing* played a significant role.

"There was a vocal contingent of original *Kingdom of Loathing* players who urged us to [release a Linux version]", he said. "We knew we'd be able to get information and support from them during development, so it seemed like a worthwhile thing to do."

Figure 2. New Britannia, the online world created for Portalarium's *Shroud of the Avatar*, has been accessible to Linux users since its early pre-Alpha releases.

His experiences making an online game also helped form that decision: "My first game project was built on the LAMP stack, so I wouldn't have a career without open-source software. And just in general, it's hard to understate its importance to the fundamental underpinnings of the internet."

At Portalarium, the company behind Richard "Lord British" Garriott's latest fantasy RPG epic, *Shroud of the Avatar*, tech director Chris Spears says the drive to support Linux came from within:

> Like most game development studios, there are a lot of Linux advocates in the office. When we ended up choosing an engine that had support for Linux, it seemed like a no-brainer to support Linux.
>
> Also, while the gaming audience for Linux is a fairly small market, they are passionate,

typically more hard-core gamers, which matches up directly with what we want in our audience! We're a bit of a thinking person's game, and that meshes exceptionally well and appeals more to the Linux crowd than most other games.

Finji's 2017 indie hit *Night in the Woods*, a coming-of-age comedy set in a run-down industrial town and starring an animated cat called Mae Borowski, was built in the Unity game engine. The heart of its complex narrative system and branching dialog choices is The Secret Lab's Yarn Spinner, released under an MIT license.

Programmer Jon Manning says that "open source is what made Yarn Spinner, the narrative engine inside *NITW*, as featureful as it needed to be to ship the game. We're

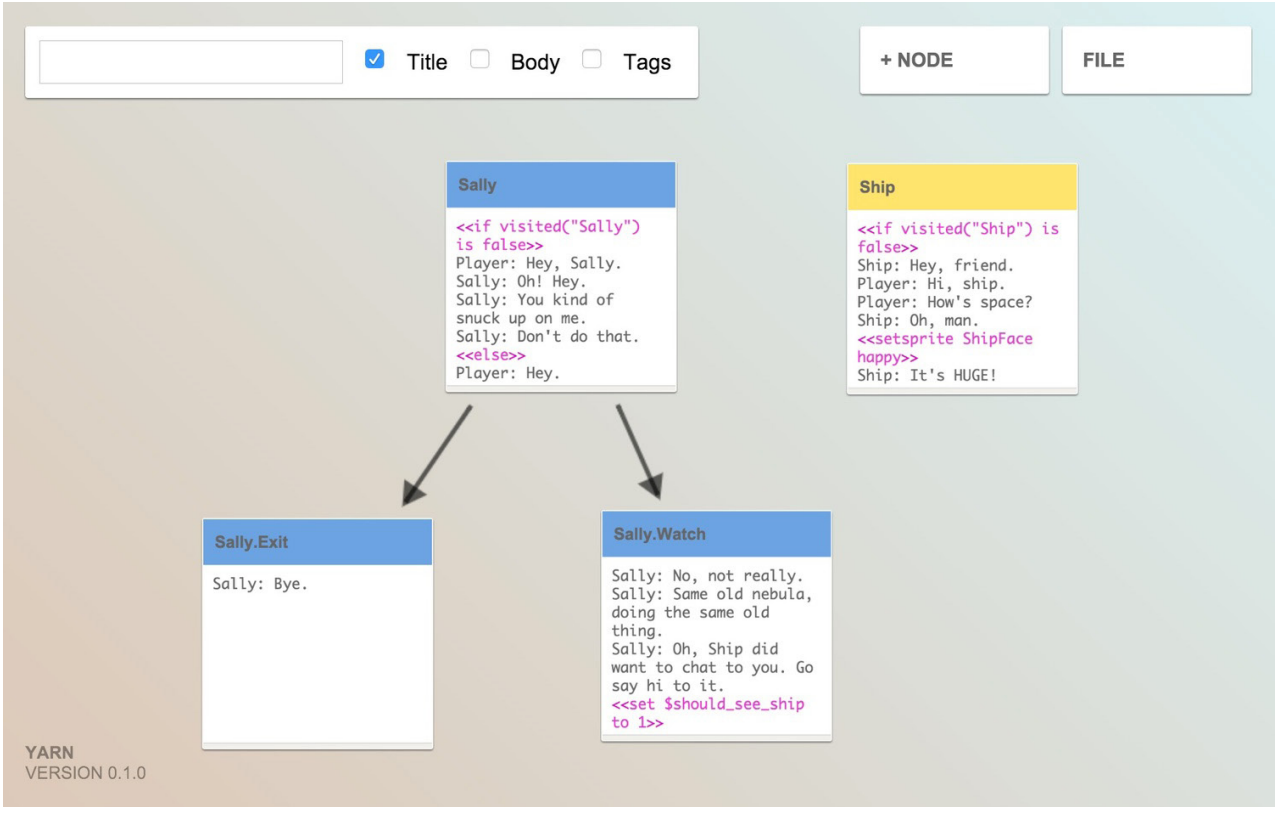


Figure 3. Yarn Spinner, available as a Unity plugin, allows developers to create and track complex narrative choices.

huge fans, and we'd love to see more people contributing to both games, game engines and tools to help game developers build their visions."

Lottie Bevan of Weather Factory, the studio behind Lovecraftian mystery card game *Cultist Simulator*, speaks highly of Linux fans:

> We were impressed with the Linux community back when we were at [the team's previous employer] Failbetter—they were our smallest platform demographic, but they were by far the most helpful and the most technically aware. Linux users seem to understand that as they're so few it's not a necessary biz decision to support Linux, and therefore are quite appreciative and helpful if you do!

What Makes It Possible?

Many of the developers I spoke to emphasized the importance of cross-platform development tools in general and of the Unity game engine in particular. Here's Chris Spears of Portalarium speaking of their decision to use Unity:

> We chose Unity 3D as our engine, and it has native support for Linux. While it isn't quite as simple as just checking a box that says "Support Linux", it is orders of magnitude easier than having to build our own engine to support multiple platforms. As a game company, most of our tech staff was already Linux-savvy, and more than a third of our engineers run Linux as their primary development OS.

Weather Factory's Lottie Bevan agrees: "Unity takes care of around 95% of the problems we'd have otherwise had to deal with, which is invaluable for a two-person microstudio!"

The *West of Loathing* team members at asymmetric also are fans of the game engine, which has supported Linux as a target platform since 2011 and has had a Linux version of its development environment available since 2015.

"We used the Unity engine to build *West of Loathing*, so right from the start, we had support for all the essential technology", says programmer Victor Thompson.

"I do not have any particular knowledge about Linux! I used a couple different distributions in the distant past, but that was strictly for maintaining personal mail and web servers."

Although many core tools and libraries are cross-platform, a game's development strategy ideally should be designed from the ground up with Linux support in mind.

When Lizard Cube was developing its lush hand-drawn remake of Westone Bit Entertainment's classic platform game *Wonder Boy: The Dragon's Trap*, director Omar Cornut says:

> We made all the data formats portable. Even if it meant, for example, using free software, such as a portable video player, which is very poorly optimized from a

Figure 4. When they created a remake of *Wonder Boy: The Dragon's Trap*, Lizard Cube's developers worked with cross-platform systems from the very start.

run-time point of view. But the simplicity of not maintaining different exports and proprietary players was really worth it. And we made all the tooling portable.

I'm heavily using my free software library, Dear ImGui, which allowed us to make a portable tooling UI. In fact, the game rendering itself is piped through Dear ImGui. This tooling even works on consoles—which aren't POSIX systems and are not supported by traditional UI toolkits.

Kitsune Games' roguelike *MidBoss*, in which a lowly imp possesses other monsters to gain their powers, similarly supported Linux from early on. "We got a lot of requests for a Linux version during development", says studio founder Emma 'Eniko' Maassen-Yarrow. She adds, "The game was also built on Ethan Lee's FNA library, and after talking to him at some events we got the impression that the Linux gaming community

Figure 5. Kitsune Games' roguelike *MidBoss* relied on an open-source reimplementation of Microsoft XNA Game Studio libraries to bring it to Linux.

is very good to developers who port to Linux, so we wanted to reward that."

Like many developers, she bought in expert assistance when it was needed: "The FNA library [an open-source reimplementation of the Microsoft XNA Game Studio 4.0 Refresh libraries] made the whole process pretty painless. We also hired Ethan to help us with the ports, and if you're working on an XNA/FNA game, I highly recommend that. In fact, we liked it so much that we've been porting our new custom game-making framework to use FNA!"

Porting specialist Feral Interactive highlights developments in graphics technologies as being particularly important. Feral PR manager Mary Musgrove says:

> In recent years, Vulkan has made a huge difference, allowing us to make performance improvements that weren't available to us in OpenGL.
>
> In terms of process, we build a lot on our own work. Internally, we have a set of unified libraries that we use across games (and sometimes platforms) that mean that no one port is entirely standalone. We have built up a strong core of knowledge over the years, meaning that our teams benefit from the learning gained from earlier projects.
>
> Externally, our contributions to open-source projects and thus the wider Linux ecosystem have had a long-term benefit. For example, Feral developers have to date made 50+ commits to Mesa drivers, and we also fix issues in Khronos' Vulkan layers.

What Are the Challenges?

Even with tools that can publish to Linux as easily as to Windows, there still are overheads associated with supporting an entire additional operating system.

asymmetric's Victor Thompson says:

> The main challenge was supporting (as in actually testing) a wide enough variety of Linux distributions. As the person most directly responsible for this aspect of testing,

Figure 6. Quality assurance testing proved to be a key issue for ensuring proper Linux support in asymmetric's *West of Loathing*.

I had to get creative with a variety of USB install and test distributions.

We did have a couple of snags, both of which appeared as bugs found by users. In those cases, the users reporting the bug were helpful in finding out what was happening. One case I remember was caused by not setting an appropriate locale for the game (important because we read text from files, and numbers that look like "1.000" mean different things depending on locale settings).

I also (re)discovered the joy of alternate window managers, some of which alter the way our game will interact with the graphics driver. We had to scramble to add a graphics option for users who had this issue, and I still don't know what a Unity game can do to get around them automatically.

Quality assurance is a recurring issue. At Portalarium, the *Shroud of the Avatar*

dev team has automated its builds for Windows, macOS and Linux, but "there are occasionally a few hiccups due to differences in the graphics library and shaders, but not too many that we have to address specifically for Linux platforms. Probably the biggest workload for supporting Linux has fallen on our tiny QA team since they have to test all new versions of the game on each platform."

Wube Software, the makers of manufacturing, logistics and resource management game *Factorio*, encountered some specific difficulties with cross-platform implementation due to the deterministic nature of the game's automated factories and processes. CTO Michal 'Kovarex' Kovarik says the team found that across different platforms:

> Some compilers implement certain functions in different ways, causing calculations to vary over time.
>
> It turned out that, in C++, basic trigonometric functions (sin, cos, asin, atan, and so on) are not guaranteed to give the same results across different platforms. We are not talking about precision here, it is about getting the same (though possibly imprecise) outcomes.

This meant that cross-platform multiplayer wouldn't work. In the end, Kovarex wrote custom implementations of these functions to produce consistent gameplay for all users.

Porting

An alternative to in-house parallel development is to bring in a porting specialist. Two of the most notable porting studios for Linux games are Feral Interactive and Aspyr Media, both of which started life as Mac porters before introducing Linux titles as their client developers began to take an interest in the platform.

Communications director Jonathan Miller says:

> At Aspyr, we are Linux believers. In the past year, we have released *Observer* and

InnerSpace on Linux, with *Next Up Hero* slated for release later this year. In addition, we released six DLC packs and the *Rise and Fall* expansion for *Civilization VI*. With the proliferation of SteamOS as well as smart TVs, we see a bright future for the platform. In short, we love our Linux supporters and will always endeavor to bring each of our titles to Linux, if at all possible.

Feral Interactive started putting out Linux conversions in 2014, with *XCOM: Enemy Unknown*. Since then, the company's most popular Linux ports have included *XCOM 2*, *Rise of the Tomb Raider*, *Life is Strange* and *Deus Ex: Mankind Divided*.

As a result of working with a number of different original developers, Feral handles a wider range of game engines and uses a greater variety of tools than most of the studios we've spoken to, porting titles originally developed in Glacier Engine, Crystal Dynamics' Foundation Engine and Unreal Engine.

Figure 7. A small, but growing number of AAA games are ported to Linux by conversion specialists like Feral Interactive.

Feral's PR manager Mary Musgrove says: "We saw Linux as an underserved audience, and because we had experience in Unix-based porting, we thought it would be an obvious extension to what we already do."

Linux gamers sometimes are frustrated that even games that have been ported to macOS often don't make it to Linux. Although modern development tools and game engines can make the job easier, there's still a lot of work involved.

The Feral team took us through the key factors involved in porting to Linux and explained where those critical differences come in:

- Compiler: "We compile Linux games with GCC, the GNU Compiler Collection, which behaves differently to the Clang compiler that we use on macOS."

- Drivers: "Linux uses a completely different set of drivers from macOS. In the case of open-source drivers, Feral developers often contribute to fixes that our games require."

- Graphics APIs: "In the last couple years, the advent of Metal and Vulkan means that our macOS and Linux games now use different graphics APIs. Prior to this, they both used OpenGL, although each platform required separate optimization processes."

- Hardware and software testing: "When we release a game, we thoroughly test it on a wide range of hardware and software configurations to determine what that game can officially support. Not only is there little overlap between macOS and Linux hardware and software, but there's naturally a lot more variation in terms of what hardware and software Linux users have."

Feral also has developed and released open-source tools to make life easier not only for themselves but also for Linux game developers in general. A recent example is GameMode (available now on GitHub), "an open-source tool that allows games to request that a set of performance-improving optimizations be temporarily applied

to the host OS. Previously, some of our games required users to manually swap the CPU governor using sudo-privileged commands, but GameMode automates this, and ensures the CPU is restored to a more efficient state when they've finished playing."

The State of the Market

One of the biggest shifts toward mainstream Linux gaming came in late 2012, when Valve launched its Steam beta client for Ubuntu, along with its hugely successful zombie shooter, *Left 4 Dead 2*.

Since then, the company has been a strong supporter of Linux hardware and software development. Although Valve's SteamOS gaming distro received a somewhat muted response, the company continues to support it, along with other projects including SteamVR for Linux, a Steam Controller kernel driver and a Linux version of the Source game engine.

A little more than 20% of all games on Steam—around 5,000 titles in total—run on Linux, and it continues to be the biggest release platform for major game publishers.

Meanwhile, retro-loving, DRM-free GOG.com—a subsidiary of CD Projekt, developer of the Witcher games—operates a more curated digital storefront, with a total of just less than 2,500 games available. Since launching Linux support with 50 titles on August 19, 2014, 851 of the games on GOG.com—more than 35%—now have native Linux installers.

Indie gaming darling itch.io has supported Linux games since it first launched on March 3, 2013, with the first downloadable Linux title, *Sophie Houlden's Rose and Time*, coming to the platform later the same month.

itch.io creator Leaf Corcoran says that Linux games and open-source software are personally important:

> I switched to Linux many years ago. I use it as my daily driver and to develop itch.io. I'm happy to see the reasons for going back to Windows for gaming are shrinking

every day. Linux as a platform was supported the first day itch.io launched. We continue to prioritize it when launching new products, like our desktop app and our upload/patching tools.

As a distribution platform, itch.io is home to many games that can be played in a web browser. There are more than 38,000 of them right now—around 35% of itch.io's total—all of which work on Linux browsers. There are also 15,000 downloadable Linux titles, adding up to another 13.8% of all itch.io games. This makes it the biggest Linux distribution platform in terms of sheer numbers, and the games on it range from artistic experiments and game jam entries to better-known games, such as *Doki Doki Literature Club*, *Night in the Woods* and *Everything*.

How Many People Play Games on Linux Anyway?

Lizard Cube's Omar Cornut says the number of Linux players of *The Dragon's Trap* is "quite low, unfortunately. Last time I looked, Linux was about 1.5% of our Windows

Figure 8. *Cultist Simulator* tells stories through the medium of a virtual card game.

sales. I think if we add up Linux and OS X sale, we just recouped external cost there, so it's not too bad!"

However, he feels that a simultaneous release may have bumped up dedicated Linux sales figures. "We made a mistake where due to timing issues, we didn't launch all platforms simultaneously, so that didn't help the Linux and Mac sales I suppose."

Figures are similar across the board. Just 2% of *Cultist Simulator*'s current player base uses Linux. For *West of Loathing*—on Steam, at least—it's a little more than 1%. A little more than 2% of *MidBoss*'s players are on Linux, while Wube Software reports that between 1% and 2% of *Factorio* players are using the open-source OS.

For *Night in the Woods*, Finji director Adam Saltsman says that Linux makes up just 0.4% of the game's total users across all platforms, which includes consoles as well as computers.

Although Portalarium's *Shroud of the Avatar* has both online and offline modes, it's most widely played as an online multiplayer game, and Chris Spears says its Linux

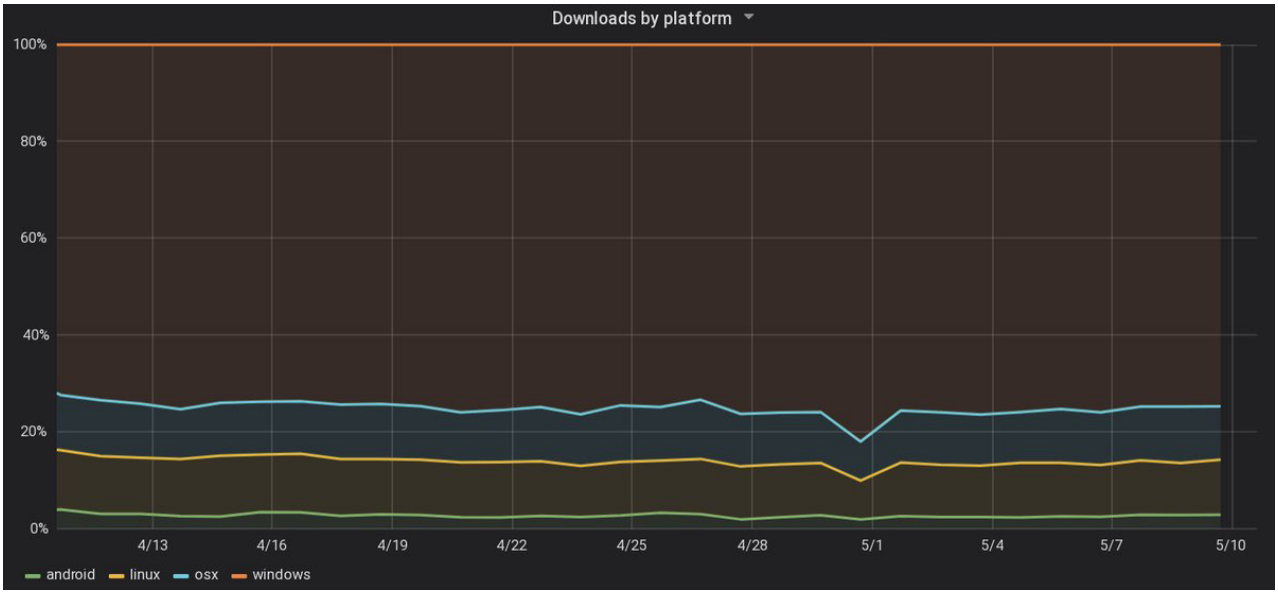


Figure 9. On indie-friendly itch.io, Linux downloads make up around 12% of the total.

players are particularly committed: “Right now only about 2% of our player base plays on Linux, but they average more hours per player. We have seen this growing slightly over time, and we have a number of players who play on multiple platforms.”

Linux users make up only a tiny percentage of these games’ total players, but most of the developers’ figures are still a marked improvement on the 0.55% Linux market share reported in Steam’s April 2018 hardware and software survey. Numbers vary, but when a company goes to the trouble of releasing a mainstream game for the OS, they can expect some 2% of their customers to play on Linux.

The figures become even more optimistic on itch.io’s indie-oriented distribution platform, where Linux downloads account for roughly 12% of the total, tied with macOS. itch.io founder Leaf Corcoran says that Linux users generate more downloads per game on average than their Mac-using counterparts too.

What’s Next?

If you’re keen to start developing games for Linux, there’s never been a better time, with tools ranging from beginner-friendly text-game-maker Twine to fully-fledged integrated development environments, such as Unity and Unreal Engine. Both support a wealth of third-party assets and plugins to help you craft anything from a point-and-click adventure game to a modern first-person shooter.

Although margins are tight, many developers genuinely appreciate the enthusiasm of their Linux-using players and are willing to support the OS if it’s at all financially viable, even if it doesn’t turn a huge profit.

If Linux users want more games to play, it’s important to add them to your wishlists—this affects the visibility of pre-released games on Steam—and to buy them. Specifically, buy them from your Linux system, as many distribution platforms register what OS you’re using at the time of purchase.

Help developers out with bug reports and reasonable feature requests. And if you enjoy a Linux title, leave a positive review for it. Reviews can have a real impact on

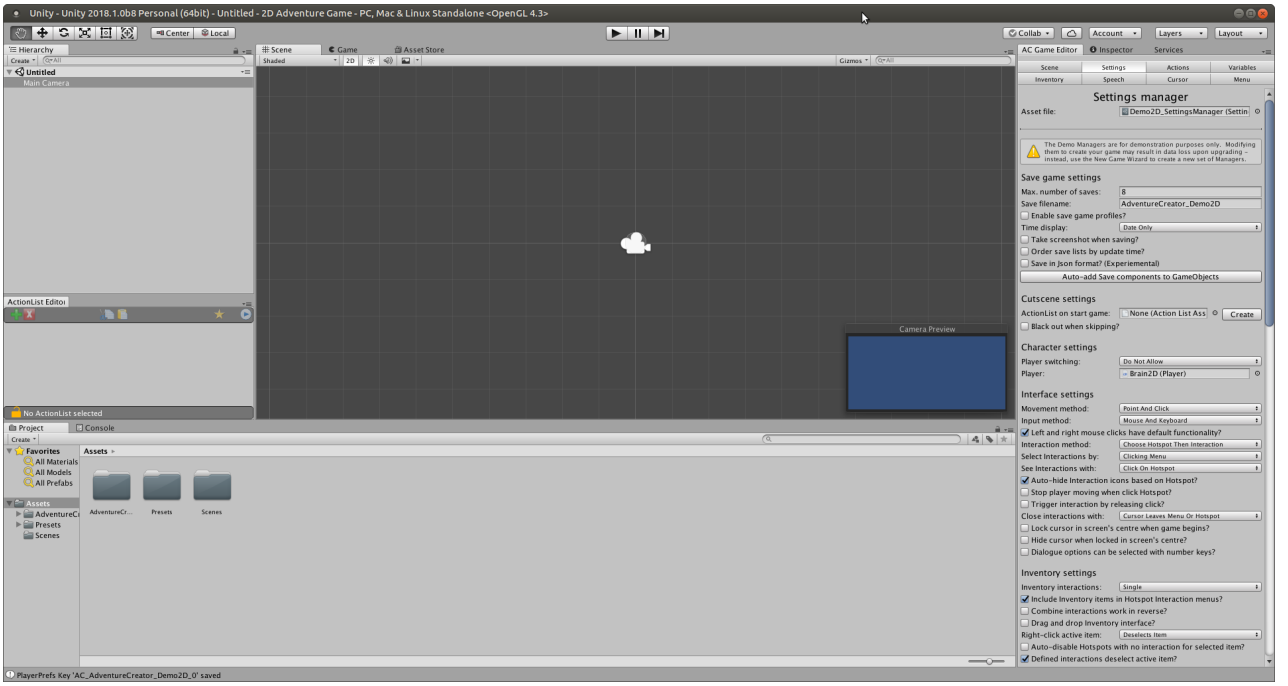


Figure 10. The Unity game engine is the development tool of choice for most of the Linux game programmers I spoke with.

Steam's Best-Loved Linux Games

Steam250 aggregates the number of positive reviews games receive on Steam to reveal which titles are best regarded by their players (source: https://steam250.com/linux250).

1. *Portal 2*
2. *Terraria*
3. *Counter-Strike*
4. *Left 4 Dead 2*
5. *Factorio*
6. *Euro Truck Simulator 2*
7. *Stardew Valley*
8. *The Binding of Isaac: Rebirth*
9. *Portal*
10. *Mount & Blade: Warband*

a game's visibility and rating on online stores, as well as giving a warm glow to a developer who's gone to the effort of supporting what's still a minority gaming OS. ■

K.G. Orphanides is a writer, journalist and occasional game developer who's been muttering darkly at the bash prompt since 1999. K.G. can be found on Twitter @KGOrphanides and on itch.io at https://mightyowlbear.itch.io.

Resources

- Unreal Engine for Linux Installation
- Unity Linux Build
- Yarn Spinner
- Twine
- Dear ImGui
- FNA
- GameMode
- SteamOS

Would You Like to Play a Linux Game?

A look at several games native to Linux.

By Marcel Gagné

Don't worry, I'm not trying to get you to play *Global Thermonuclear War*, since as we all know, the only way to win is not to play. And we want to play. Okay, enough with the mandatory classic movie references.

There are, of course, tons of games for the Linux platform if you're willing to install Steam. For the sake of this article, however, I want to show you some games that are available native to Linux—none of this firing up Java so you can play something on your Ubuntu, or Fedora, or Debian or whatever your personal flavor of Linux happens to be.

I also want to stay away from the classics. Sure, I know, *Frozen Bubble* is awesome and always has been awesome. Ditto for *SuperTux* or *SuperTuxKart*. They're all fantastic, and if you don't know those games, be sure to check them out after you finish reading this. In fact, if you fire up your distribution's software center and search the repository, you're going to find a ton of games. I'm writing this on a computer running Kubuntu 18.04, and Discover, the default software center, has a big section dedicated to games (Figure 1).

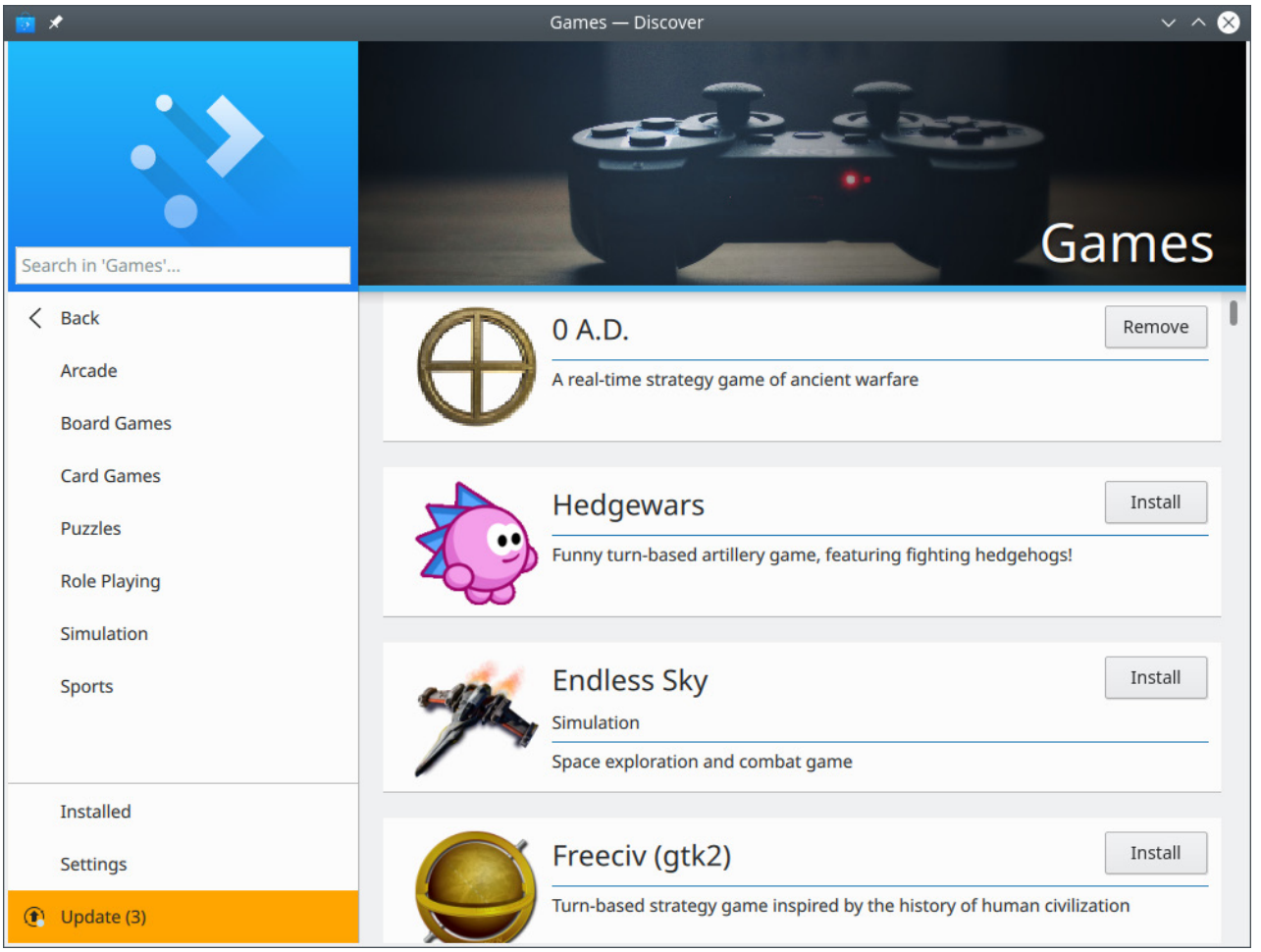


Figure 1. Discover is the Kubuntu and Plasma default software center.

That said, I want to take you out of the classic Linux comfort zone for games and show you something you may not have considered. In short, if you've got the itch to try something new, I may very well have a way for you to scratch that itch. While looking around for great game ideas, I started off scouring the web for anything I might have missed in recent days or weeks. So much of what I found was either oldies and goldies (hello again, *Frozen Bubble*) or Steam. And then, I tripped over itch.io, a place I had somehow totally missed before.

Here's a short introduction. itch.io is a website for indie game designers to share their passion with the public. You can download a game, pay what you like (or can afford)

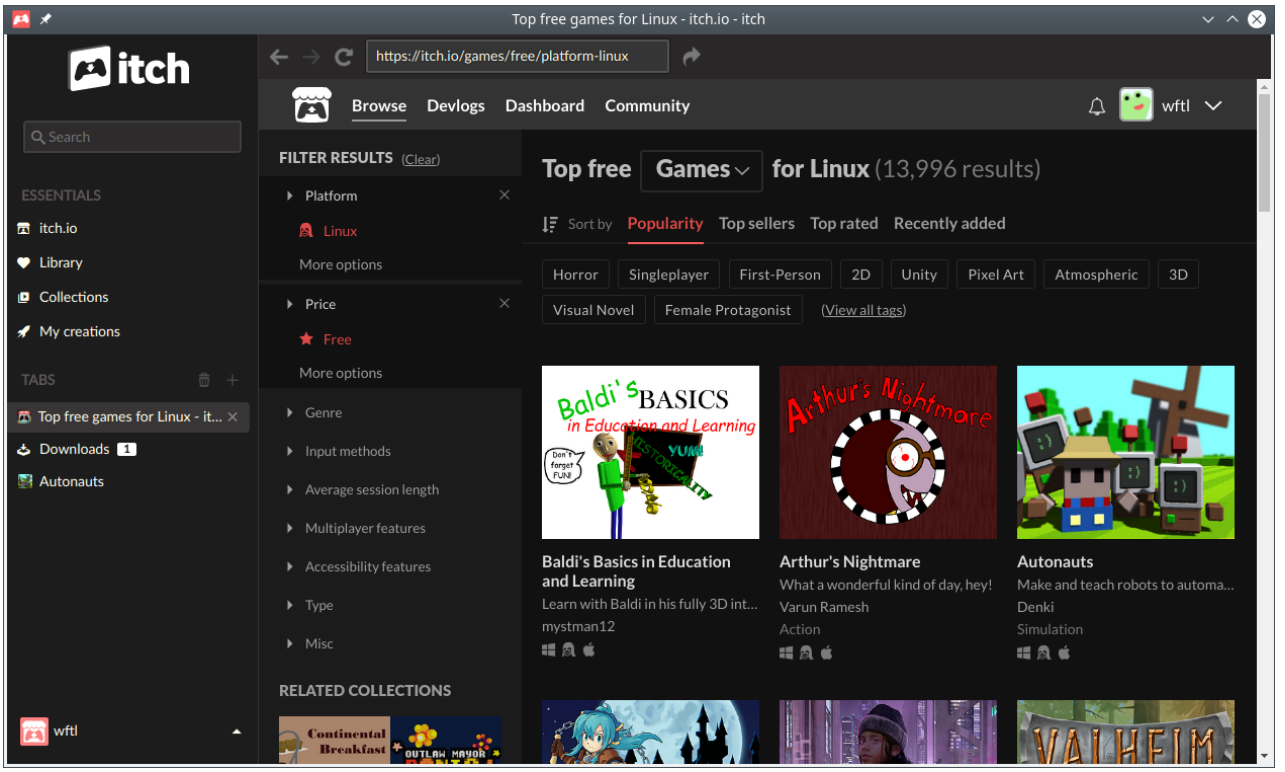


Figure 2. The itch.io Storefront and App

and communicate with the developer as the game matures. The nature of the site is going to seem very familiar to anyone who has been in the open-source universe for any length of time. There's a lot of releasing early and some releasing often. Because players can interact directly with developers, you can influence the design of games you love.

Here's another cool thing. The site, like Steam, has an app you can download and install on your Linux system that acts as a storefront for all the games, as well as a place to track, install and run said games. They maintain DEB packages for Ubuntu and Debian derivatives, RPMs for Red Hat/Fedora and openSUSE, AUR packages for Arch Linux distros, and there's even something for the Gentoo crowd. Figure 2 shows a look at the storefront opened to Linux games. It's a veritable gold mine of new gaming opportunities for the Linux lover in you.

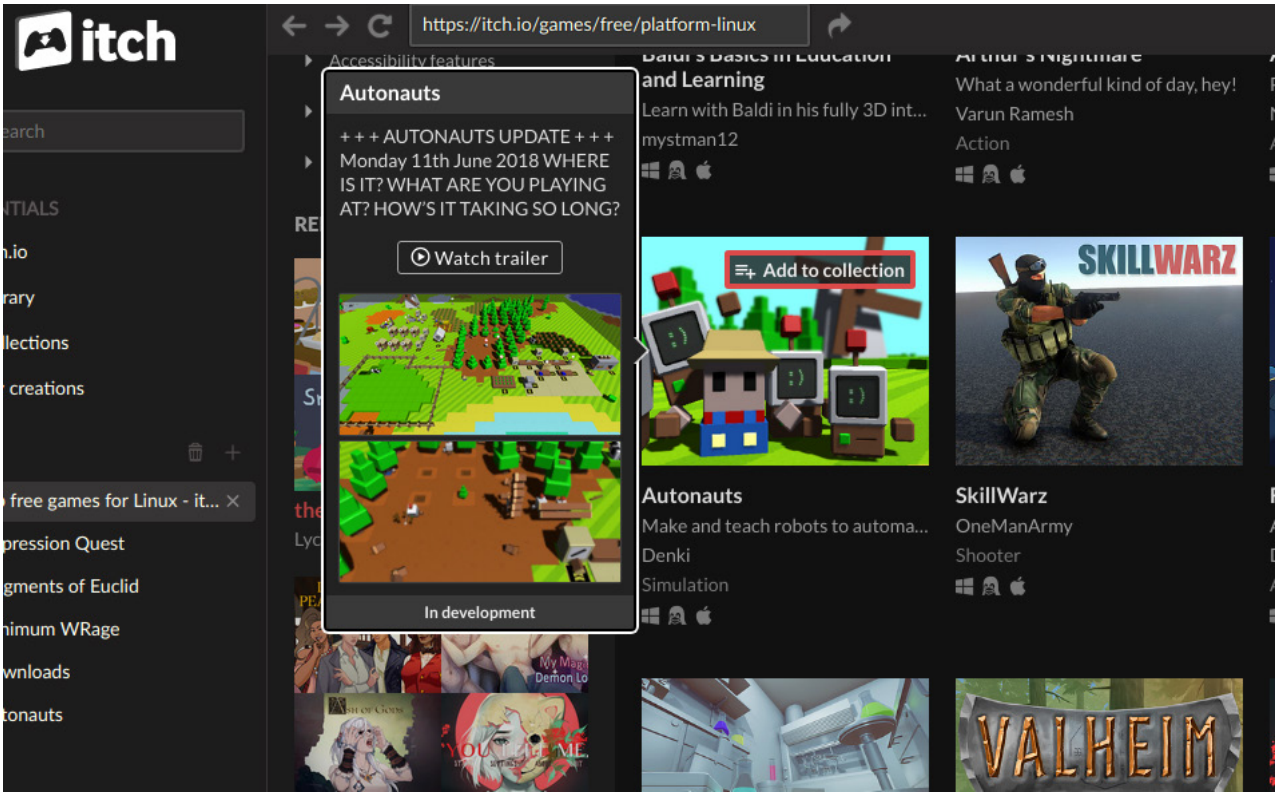


Figure 3. Hover over a title to show a small descriptive pop-up.

I'll take two seconds and mention that you also can search for and install Windows and macOS games as well, for people who still run those. The app also lets you search by whether the games are completely free (or pay what you like), popular, new, highly rated or of a particular genre (such as FPS or RPG). To the right of the app, you can choose to organize your game library as you see fit and launch the games from there.

The games are listed in a grid on the right. Scroll up or down to find something you like, then click a title to learn more about it. If you simply hover over a particular game, a small windowed description panel will pop up; from there, on some of the games, you can choose to play a game trailer (Figure 3).

When you click a title for the full description, look to the far right below the game thumbnail for an Install button. Directly below that, depending on the game, you may

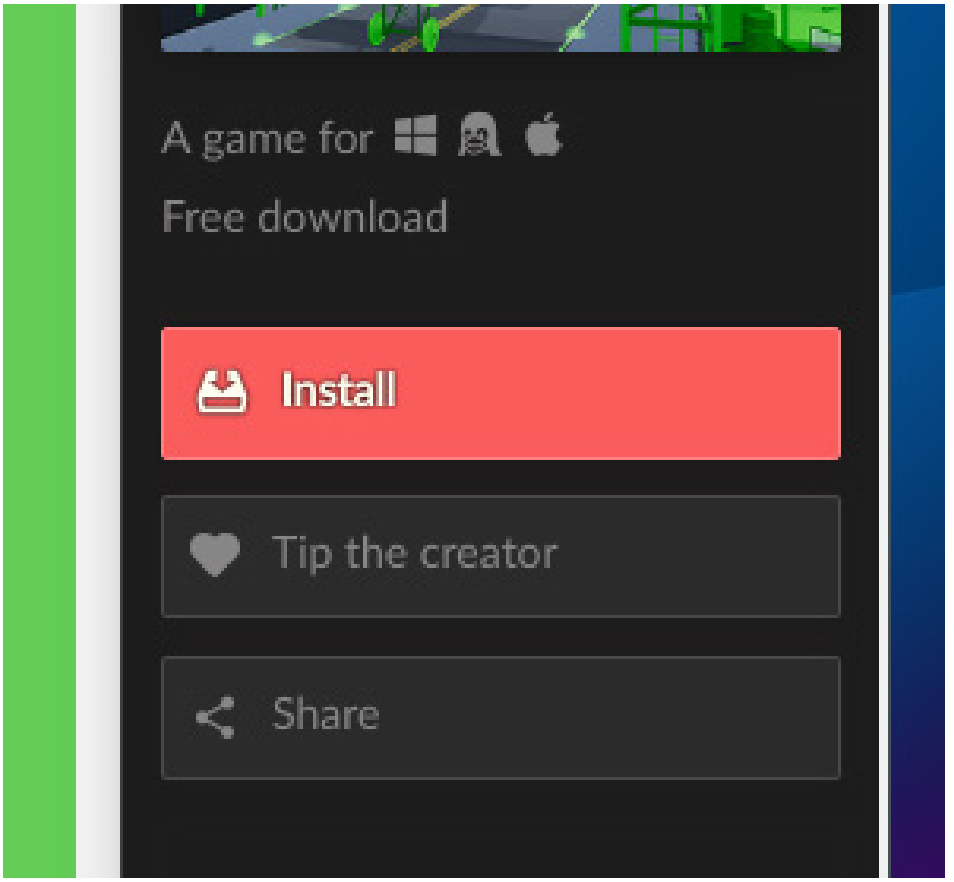


Figure 4. Install a game, and maybe give a little something to the creator.

see an option to tip the creator for his or her work on creating the game (Figure 4). When you are ready, click Install, and everything happens in the background. Assuming you haven't wandered away from the description, the Install button will turn into a Launch button from which you can, you guessed it, launch the game.

Eventually, you'll install several games, and your games all will appear under the Library button to the left of the itch.io app. Directly below that, there's a collections button, as each game you download can be added to your personal collection where you organize things as you see fit. Whether you look at the entire library or a defined collection of game types, now it's just a matter of clicking the Launch button.

When you launch a game, a configuration dialog appears offering you a variety of screen resolutions on the left from which you can select what works best for your particular

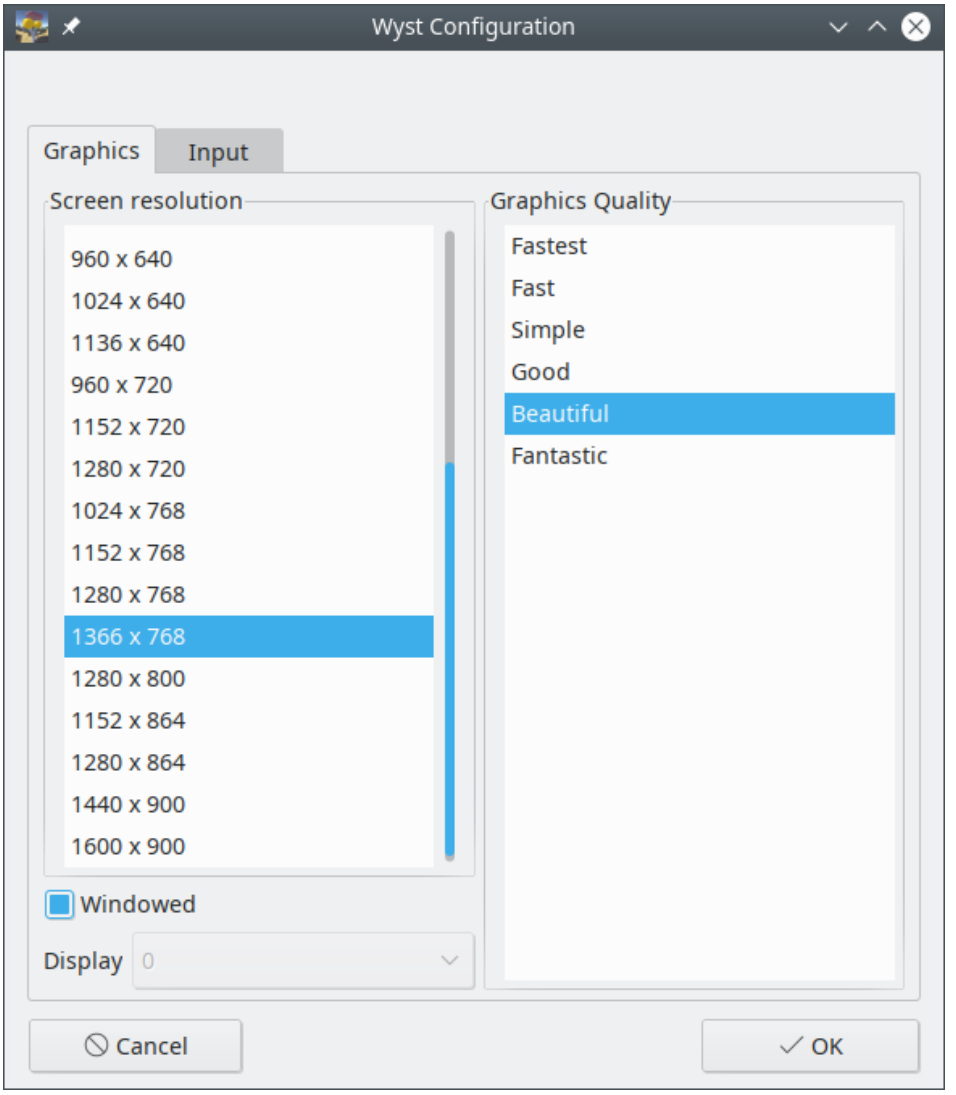


Figure 5. Each game lets you select both screen resolution and quality.

monitor (Figure 5). Directly below the resolution list is a check box that allows you to choose either full-screen (the default) or windowed play. On the right is the graphics quality listed from Fastest through to Fantastic. Needless to say, fantastic graphics may deliver slower performance on lesser systems. Use what works best for you.

Click OK and let the fun begin!

Let me show you a few games that I found and enjoyed, starting with ***Autonauts***. At first glance, *Autonauts* looks like a *Minecraft*-ish sandbox game, and certainly there

Figure 6. Programming a new robot to find and cut down trees.

are some elements of that—you collect resources, craft things, build and so on. Autonauts, short for Automationatics, are artificially intelligent robots who travel the universe looking for worlds whose development they can assist (Figure 6).

They take advantage of local resources to build a basic infrastructure from which they can build other robots. When new robots come online, the more advanced Autonauts can train them to perform tedious tasks like collecting resources and building stuff. You do this by having new workers watch you and record what you do, so that they can then be left to operate on their own. As the society evolves, increasingly complex workerbots can perform increasingly complex tasks. The idea is to continue churning out robots, automating everything that can be automated so as eventually to create a modern, efficient society.

I took a break from playing to chat with Aaron from Denki Games, the creator of *Autonauts*. The game has a Discord link where you can chat with the creator and

Figure 7. Taking to the skies in *Attack on Toys*.

with other players, and that's where I found Aaron. I asked him what he thought of the itch.io concept, and he answered "We've known about Itch for a while. I'm more interested in experimental than mainstream games in general, so I'd come across it long before we started *Autonauts*."

When I told him that it reminded me of the Steam store and front-end app, he said, "I think that's a fair comparison. For us, it allowed us to experiment with our ideas without the harsh, critical spotlight of Steam Early Access. People are a lot more forgiving on Itch. :-)"

Realism is great, but I don't need my wargames to have the smell of soldier sweat

Figure 8. The Mysterious Island World of *Naufrage*

mixed with blood and mud. That's why I loved ***Attack On Toys***, a wargame third-person shooter where you fight with plastic toy soldiers. You command the green army against the tan army. There is strategy involved as you place your defences before each battle, so it's not just shoot and hope for the best. There are different vehicles you can ride in, including an airplane, and different weapons. The enemy also has a big mechanical soldier/robot that you have to defeat. The action takes place in a large room, across tables and chairs and bookcases (Figure 7). It's really kind of awesome if, like me, you're still six years old on the inside.

Then, there's ***Naufrage***, a bizarre adventure without rules. The game opens with you dropping a coin into an arcade machine, and suddenly you are following the coin and dropped onto a mysterious island. Ahead of you is a silent amusement park, shrouded in darkness, with a silent ferris wheel beckoning you (Figure 8). As you explore this place, you soon discover you're not alone, although you wander in solitude. A

wonderfully creepy soundtrack accompanies you as you decipher puzzles, read cryptic messages and follow signs that are often anything but helpful. It's frankly hard to explain *Naufrage*—a French word meaning "shipwreck"—but it's definitely worth the peaceful yet foreboding journey across and around this island.

What I love about *Naufrage* is the nearly complete absence of rules. Using either cursor keys, or the classic WASD to move and spacebar to jump, the entire point of the game is to figure it out as you go along.

Although I enjoyed the experience, it did make me wonder about the commercial possibilities for some of these games. Like many open-source projects past and present (but mostly past), games on itch.io seemed to be designed to scratch a developer's itch. Never mind whether the game was going to make somebody rich—much of what I encountered seemed to come from a place where the desire to make something spring from that inner-creative place was powered by the thought that somewhere out there, other kindred spirits might enjoy it.

You see, the independent and experimental nature of the site means you're going to find games for your Linux system that you aren't likely to run across anywhere else. Are you ready to make $7.50 an hour, actually fighting other employees for a promotion only to face off against a boss who literally happens to be your boss? Try ***Minimum WRage***. How about an interactive story about gay girls playing baseball and learning about love (***Butterfly Soup***)? Or a horror version of Arthur the cartoon aardvark (***Arthur's Nightmare***)? There's also a bizarre game that drops you into a strange world of M.C. Escher architecture to discover...who knows what (***Fragments of Euclid***). Should you just want to burn off a little tension, choose one of the many first-person shooters, like ***Rexuiz***, and try your best to stay alive for more than a few seconds.

The selection is huge, and I feel like I've just touched the tip of the proverbial iceberg. Some are silly and just plain fun, and others will have your heart pounding. There also are some pretty disturbing and occasionally terrifying titles in there, so make sure you read the descriptions before installing anything—unless you're one

Figure 9. *The Blind Griffin*, an Otome Game

of the brave ones, without fear. Another surprise, for me at least, was the large number of Otome games like *The Blind Griffin* (Figure 9). For the uninitiated, as I was, these are anime-style and story-based games aimed at women (otome is Japanese for "maiden").

Don't be afraid to venture outside the classics available via popular Linux distribution repositories or Steam. People have grown used to thinking of Steam as the place to go for great games on Linux, and it makes sense given the quality of those offerings. Steam even has its own Linux distribution, so it's definitely playing in our court—most of the time. But beyond those repos and Steam's catalog of games, I'm happy to report that there's another world where independent developers create games and chat with their fans on Discord channels.

You should really check it out. ■

Marcel Gagné is Writer and Free Thinker at Large. The Cooking With Linux guy. Ruggedly handsome! Science, Linux and technology geek. Occasionally opinionated. Always confused. Loves wine, food, music and the occasional single malt Scotch.

Resources

- Discover, the KDE Software Center
- itch.io
- itch.io Storefront App Installation Instructions for Linux

Classic Linux Games:

- *Frozen Bubble*
- *SuperTux*
- *SuperTuxKart*

itch.io Games Mentioned in This Article:

- *Autonauts*
- *Attack on Toys*
- *Naufrage*
- *Minimum WRage*
- *Butterfly Soup*
- *Arthur's Nightmare*
- *Fragments of Euclid*
- *Rexuiz*
- *The Blind Griffin*

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

Meet TASBot, a Linux-Powered Robot Playing Video Games for Charity

Can a Linux-powered robot play video games faster than you? Only if he takes a hint from piano rolls...and doesn't desync.

By Allan Cecil

Let me begin with a brief history of tool-assisted speedruns. It was 2003. Less than half the developed world had internet access of any kind, and YouTube hadn't been created yet. Smartphones were rare and nascent. Pentium III processors still were commonplace, and memory was measured in megabytes. It was out of this primordial ooze that an interesting video file circulated around the web—an 18MB .wmv labeled only as a "super mario bross3 time attack video" [sic]. What followed was an absolutely insane 11-minute completion of the game by someone named Morimoto replete with close calls, no deaths and Mario destroying Bowser after apparently effortlessly obtaining 99 lives. The only other context was a link to a page written in Japanese, and the rough encoding that Windows Media Video format was known for in that era made it difficult for casual viewers to observe that it was an emulator recording rather than the output of a real Nintendo Entertainment System (NES) console.

The video encode had in fact been made with the Famtasia NES emulator using Tool-Assisted Speedrun (TAS) re-recording tools consisting of a "movie file" of the

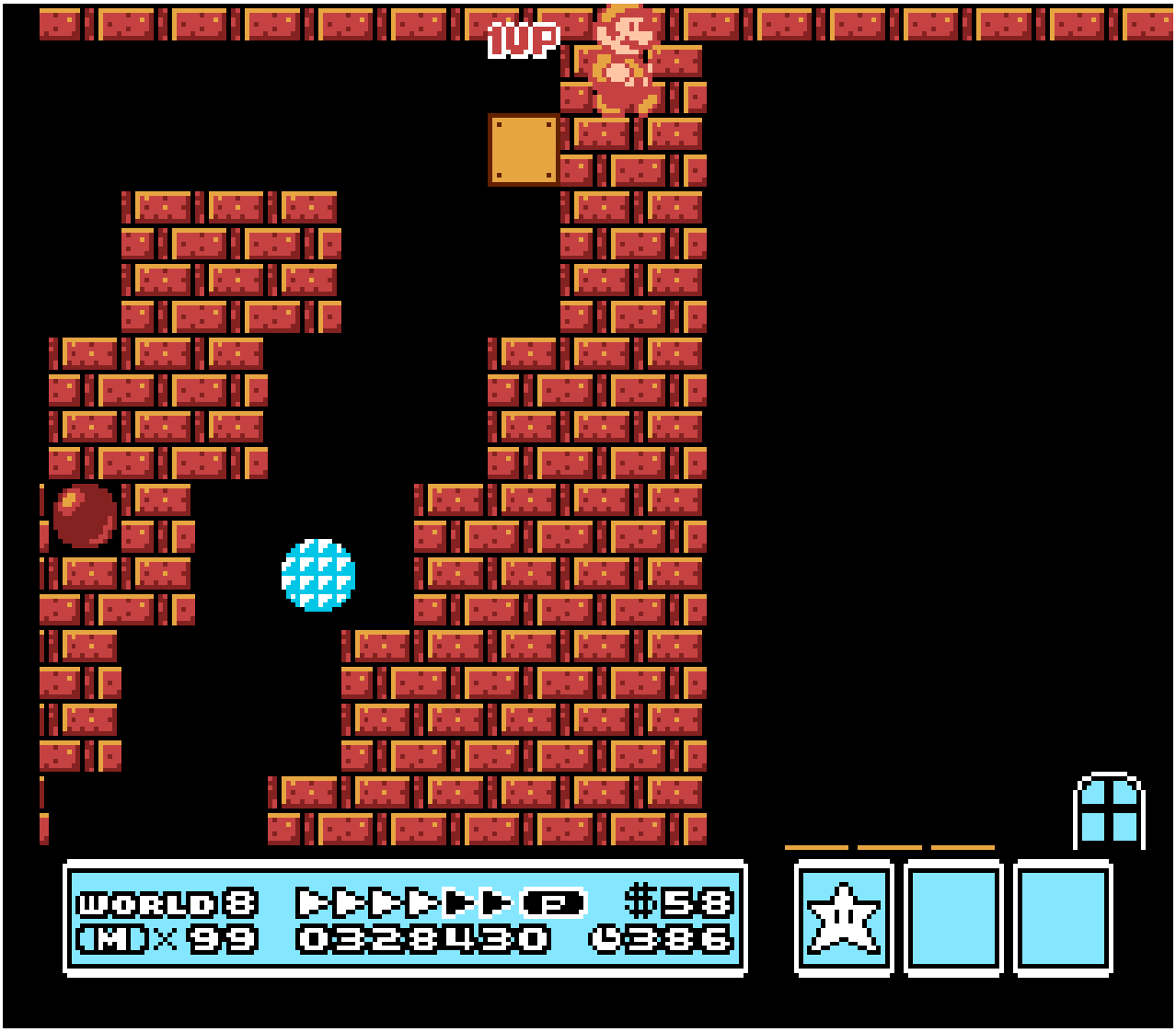


Figure 1. Morimoto's 2003 *Super Mario Bros. 3* (SMB3) Time Attack Video

sequence of all buttons pressed along with the use of savestates, or CPU and memory snapshots allowing returning to a previous state. Morimoto had in essence augmented his own human skill by using tools that allowed him to return to a previous save point any time he was dissatisfied with the quality of his play. By iteratively backing up and keeping only the best results, he had created what he considered at the time to be a perfect play-through of the game. I didn't know anything about how it was made the first time I saw the run, but it blew my mind and had me asking questions to which I

couldn't find answers.

The human speedrunning community members were naturally highly offended by what they saw as an unlabeled abomination akin to a doped athlete being allowed to compete in the Olympics. Their view was that anything that augmented raw human ability in any way (even as rudimentary as keyboard macros in PC games) was considered cheating, and Morimoto's run was nothing more than a fraud best left ignored. There was fascination, intrigue and division. It was, in retrospect, the perfect recipe for a new website.

An industrious viewer going by the name Bisqwit was especially impressed by Morimoto's run of SMB3 and in late 2003 founded what ultimately would become TASVideos.org. He worked to ensure the human speedrunning community was as appeased as possible by providing disclaimers on all Tool-Assisted encodes of gameplay to help ensure that later viewers knew it was not purely human skill on display. He provided context for Morimoto's run through a history revival project.

Others quickly joined him, and a thriving community rallied around the creation of superplays and speedruns as well as around the creation of various tools like frame advance and embedded Lua emulator scripting to further allow "perfect" input with no mistakes. Older TAS runs inspired by Doom Done Quick that took segmented *Doom* runs to previously unheard of extremes were added to the site, bringing light to TAS content that substantially predated Morimoto's SMB3 run.

I was one of many who saw Morimoto's SMB3 run early on, and then I promptly put it out of my mind until TASVideos had grown in size to the point that it caught my attention circa 2006. By 2008, I had joined the site myself under the name dwangoAC in order to submit my first TAS attempt completing the NES pinball sim *High Speed* using the Linux-compatible FCEUX NES emulator. I was along for the ride, as TASes became a distinct art form anchored at TASVideos.org with a highly organized community of judges, publishers (video encoders) and emulator coders. TASVideos has since matured as the de facto repository of TAS content, and it now hosts a vast number of TAS movies across many game platforms.

Self-Playing Instruments: a Prior Art Interlude

I want to pull back for a moment and talk about a bit of history. Multiple centuries ago, incredible inventors started creating the first self-playing musical instruments using large drums with pegs like oversized music boxes and more complex devices using air forced through holes punched in cardboard, paper or even metal sheets to drive organs. Regardless of the mechanism, the end

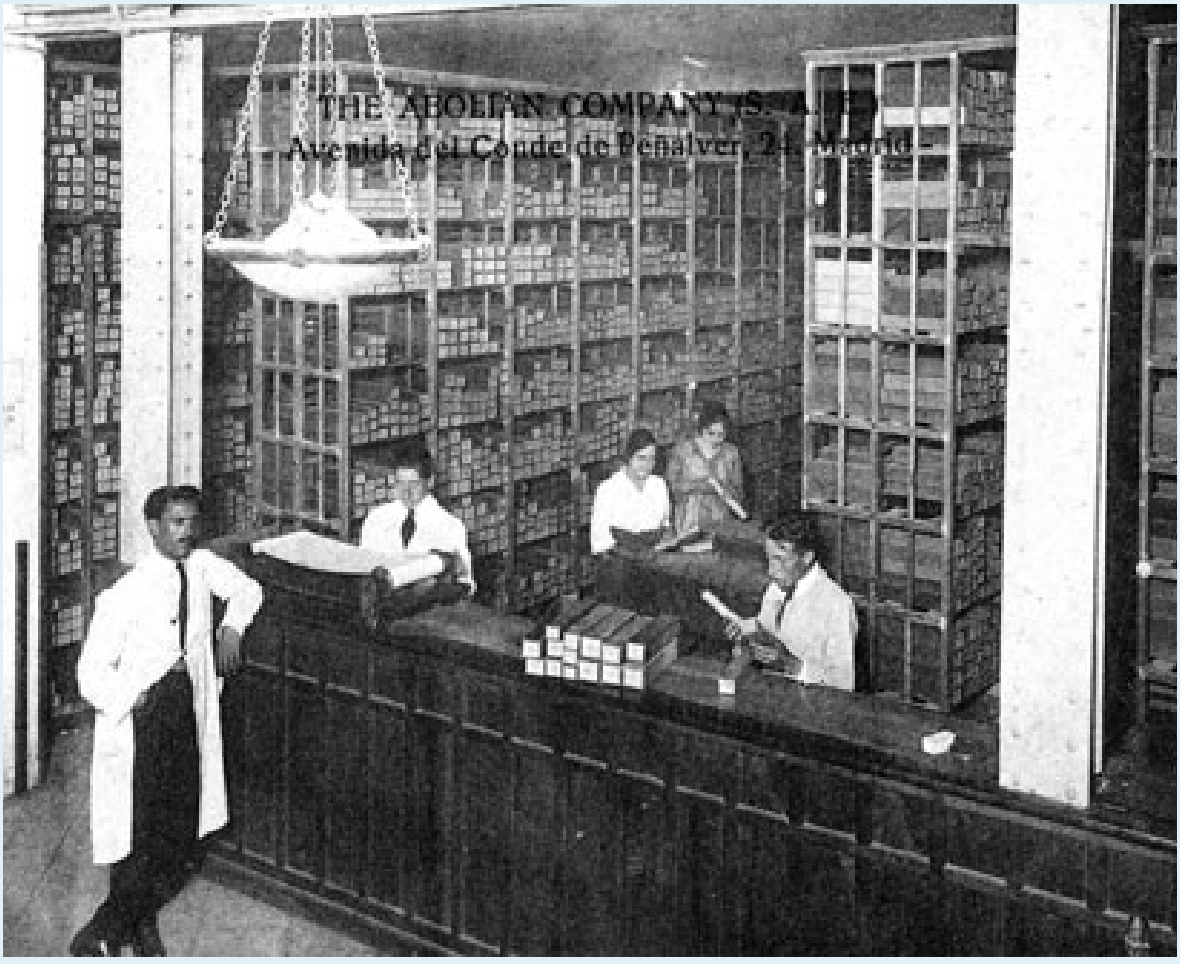


Figure 2. An Aeolian Company Roll Library, Madrid, Spain, c. 1918; Image Credit: pianola.org

result was a pre-arranged sequence of notes that could be deterministically played back with the art form gaining widespread adoption by the 1920s. An entire industry of piano roll manufacturers with more or less agreed upon standardized piano roll formats for at-home player pianos existed.

Some piano rolls were made by having a famous pianist sit at a reproducing piano that would faithfully record every note (good or bad) often by punching perforations or holes in a paper roll in real time as the pianist played. Eventually after a few attempts, the pianist's result would be satisfactory, and the piano roll would be copied and sold to customers to play on their own player pianos at home. The modern gaming equivalent would be human speedrunners recording their single-segment attempts and posting their best results online.

Other compositions of arranged music were made tediously by hand by reading the original arrangement and punching holes one row at a time while compensating for factors such as changing tempo and the fact that a piano roll shrinks in diameter causing the paper movement speed to change as the piece plays. I would equate this to making a normal TAS. At some point, someone making one of these pre-arranged compositions pieced together that there are 88 keys on a keyboard but only ten fingers on humans and reasoned that because the piano roll wasn't designed for a human anyway, a few extra simultaneous keys to add some punch to the composition couldn't hurt. The logical end result many decades later is now known as a Black MIDI composition, but that's an article for another day. The point is that if you take human limitations out of the equation, you can do some rather interesting things in many art forms.

The Dawn of Replay Devices

As PC specifications improved through the years, it became possible to use the increased resources to improve emulation accuracy without making the emulator unbearably slow. This allowed an even more ambitious concept to form: the idea of taking a TAS movie file and coercing an original unmodified console to behave the same way. The feat relies on extremely accurate emulation of each component of a console including any cartridge-based memory mappers and expansion chips. The theory is that if an emulator can step through the execution and interrelated dependencies of an entire system accurately enough, it should logically be possible to provide that same sequence of button presses to a real console and achieve the same result. This deterministic behavior is viable even on games that appear to contain randomness, because many consoles contain no external source of entropy and rely on player input as the only seed for a pseudo-random number generator—meaning a given sequence of button presses sent in order from power-on results in the same game state every time.

The idea was first raised in 2006, but it wasn't until 2009 that a TASVideos user going by the name true started working on a device to send prerecorded input to an NES via the controller ports. That same year, a hacker named Jaku independently created a device that was able to play back the entire first level of SMB1, unbeknownst to the TASVideos community. Finally in 2011, micro500 successfully completed an entire play-through verification of SMB1, thanks to accuracy improvements in the FCEUX emulator, and he created an Instructables guide on how he built his NESBot. DarkKobold followed micro500's guide to make his own device and played back *SMB2* and *Wizards and Warriors 3* publicly at SGDQ 2011 (more on that in a moment). By the end of the year, SoulCal had created a replay device named Droid64 that was capable of playing back the full 120-star completion of the N64 game *Super Mario 64*.

Interest in playing back TAS movie files on original hardware using replay devices increased from 2011 on and became known as console verification, but only a small number of creators had the requisite replay device hardware. TASVideos was updated with a special category to denote runs that had been console-verified with proof provided using camera shots of real consoles playing back runs. GhostSonic adapted

his replay device to work on a Sega Genesis, and later endrift created a Game Boy Advance device using the Gamecube's Game Boy Player. Despite the progress on all those fronts, the concept that input could be sent to a console was generally not well known outside TASVideos.

Gaming for Charity

Over time, SpeedDemosArchive.com, or SDA, became the most used site for recording human speedrun attempts of video games. Several SDA users decided to hold a charity fundraiser in which speedrunners would complete one game after another without stopping in a 2010 24/7 marathon event they called Classic Games Done Quick (in homage to the earlier Doom Done Quick naming). The event was a success, raising more than $10,000 for CARE. The organizers went on to form a series of increasingly successful week-long marathon events held twice a year at GamesDoneQuick.com primarily benefiting Doctors Without Borders and the Prevent Cancer Foundation, in time arguably overshadowing SDA in visibility.

Back when DarkKobold used an NESBot at Summer Games Done Quick (SGDQ) 2011, the events still were being held in the basement of one of the organizers, but by the time the winter Awesome Games Done Quick (AGDQ) 2014 marathon was being planned, the events had increased in size substantially, with hundreds of attendees watching runs live in increasingly larger hotel ballrooms. It was around this time that I became interested in attending an event myself, but I wasn't content with just showing up to watch. I wanted to participate somehow. A lot of other people had the same idea, and it was fairly clear that only the most interesting or well executed runs had a chance to get in. For some reason, the idea of doing TAS-related content wasn't initially on my mind (and embarrassingly, I briefly entertained the idea of doing a speedrun of the game *Scribblenauts Unlimited* until I quickly discovered I lacked the requisite skills without a lot more practice, but I digress).

In time I came to my senses, remembered my TAS'ing roots, and volunteered to present TAS replays at the event. Although there were some dissenting voices from established SDA members who still were upset about the Morimoto conflict, the overall reaction was positive. In the end, the TAS submission was accepted and

allotted 30 minutes of the marathon schedule.

Talking to a Console (and Hoping It Talks Back)

When I made the AGDQ 2014 TAS replay submission, I wasn't initially sure whether I could pull off console verification of the runs. At the time, I had no hardware and limited experience, and it was a daunting learning curve becoming familiar with everything from oscilloscopes to shift registers. I borrowed an Arduino from a colleague and set about attempting to build my own NESBot based on micro500's Instructables guide. Although I technically succeeded, in the sense that I was able to play back a run of *Tetris*, it quickly became clear to me that my amateur attempt using

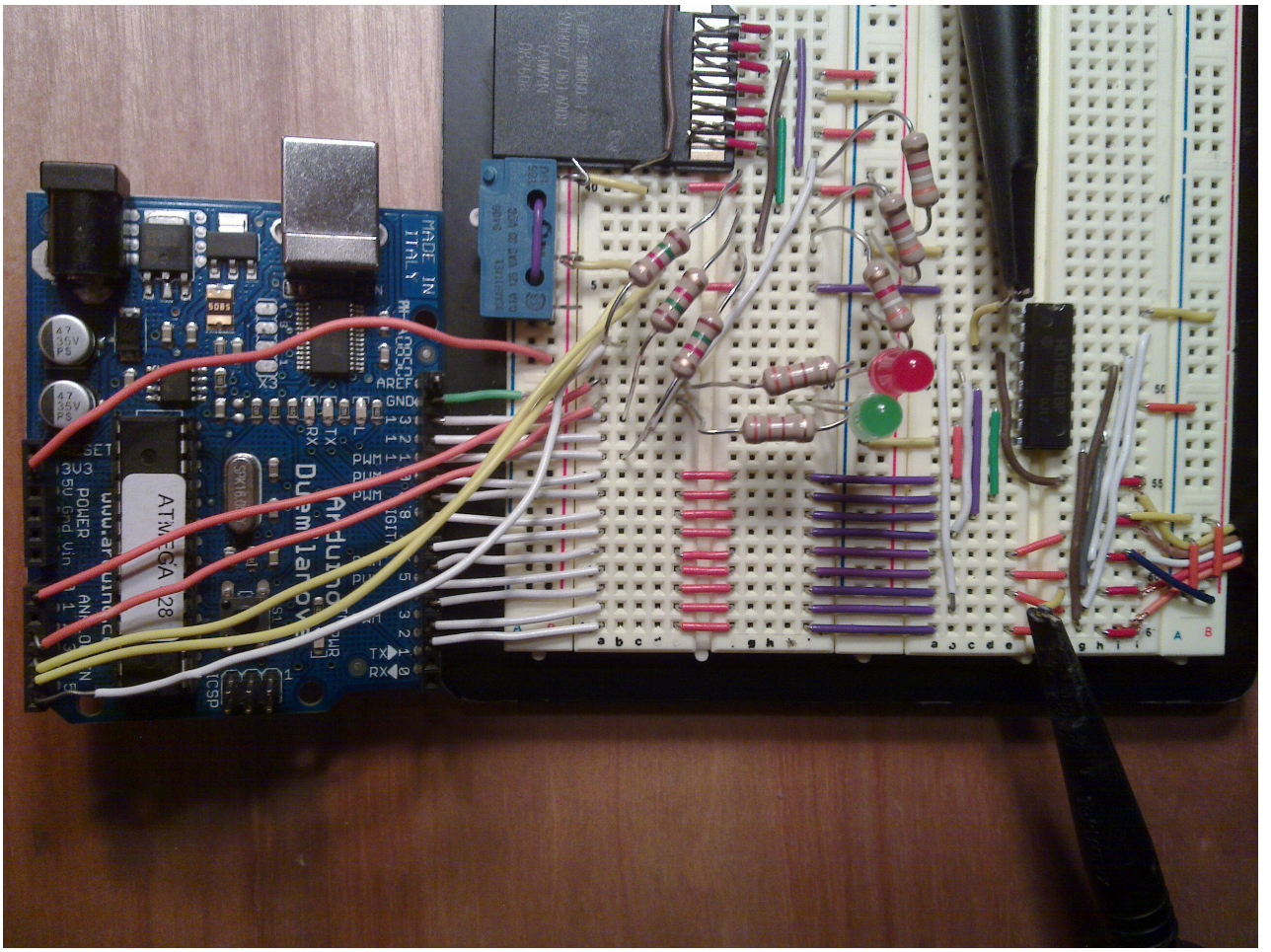


Figure 3. Fragile doesn't begin to describe the SD card pin connections.

haphazard parts on an old breadboard was extremely fragile. After contemplating what it would take to get the device to survive a plane flight intact, I made the wise decision to abandon the design.

My next attempt was, in theory, going to be far more simple from a wiring perspective by employing only the GPIO pins on a Raspberry Pi 1 Model B along with some resistors to allow it to handle the 5v voltage coming from the console. It didn't take me long before I knew I needed help, in part because I was out of my depth on the electrical engineering front. Folks like zid helped me in long troubleshooting sessions, but before that, they patiently had to help me sort out a mental model of how input from a controller actually works.

When I was a kid, I was very confused about how an NES controller could possibly work. I noticed there were eight buttons on the controller, but I saw only seven pins on the oddly shaped NES connector, and I assumed each button needed an individual wire. Early designs found on Atari and similar consoles did exactly that, but as more buttons were added to controllers, it made sense to use a shift register that could ingest all buttons in parallel but send the data down a single serial data line toward the console to reduce cabling costs.

The process of reading player input from a normal controller on an NES, SNES or any similar serial controller reading console goes like this. First, consider a situation where a player is pressing down one or more buttons—let's assume Right and B to Run Right for Great Justice. The game sends a latch signal to the controller effectively telling it to be prepared to be read, and then a clock line signal is sent at which point the state of the first button in an agreed-upon sequence is returned from the controller to the console on the serial data line; let's say it's the A button. Since that button isn't being pressed, a binary 0 is sent back by holding the line high (the logic is inverted on an NES). The process repeats for each button, and in this case, Right and B would be sent back as a 1 by holding the serial line low after the appropriate clock. All of those individual button presses eventually will be stored as a byte (or, for SNES, two bytes) in memory to be operated on by the game. That's all relatively straightforward, but the challenge is that while the time between latches is usually measured in

Figure 4. Latch, Pulse (Clock) and Data Line Example

milliseconds, the time between each clock pulse is potentially on the order of nanoseconds.

Getting back to the poor Raspberry Pi—GPIO polling in nanosecond timeframes isn't one of its strong points. We quickly discovered it would frequently miss clock signals and sometimes even latch signals due to the Linux kernel's inadequate polling frequency of the GPIO pins. I was the President of the North Bay

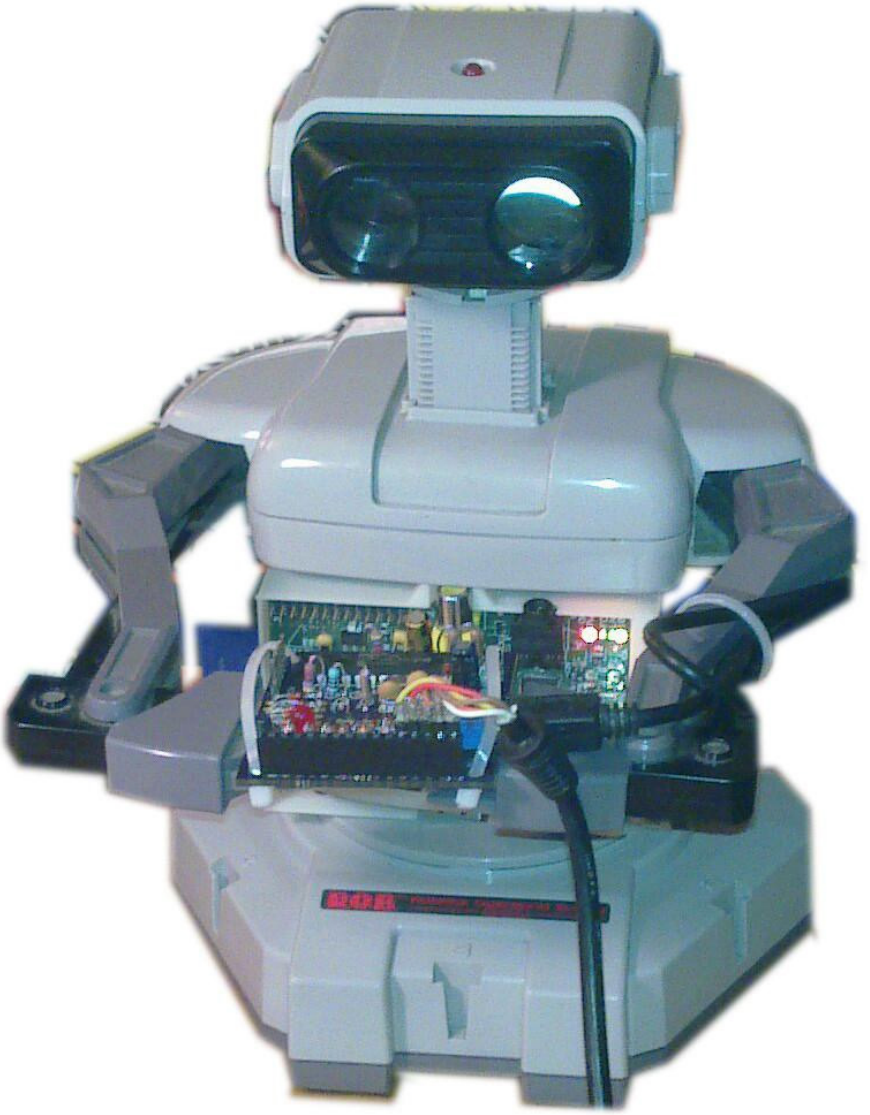


Figure 5. TASBot in His Early Days before LEGOs

Linux Users' Group by that point, and I wanted to represent Linux as best I could, so I pushed hard to find a way to incorporate it. The solution was using a device designed by true that he named his NES/SNES replay device to buffer controller input that allowed the Raspberry Pi to send serial data at a much more achievable rate. The fusion of the two boards mounted to an R.O.B. robot inspired me to call the whole thing a ROBBerry Pi, but the name did not stick around for long. Possibly as a result of my (in retrospect) somewhat terrible name, multiple community members converged on the name "TASBot", and the name stuck. I now consider myself to be the keeper of TASBot, a one-of-a-kind individual with his own personality who uses a variety of replay devices to partake in his favorite pastime of smashing through games inhumanly fast.

Snake, Pong and ACE

For TASBot's debut at AGDQ 2014, we started with the NES game *Gradius*, but it desynchronized a minute or so in. A desync is akin to the paper roll jamming or tearing during the middle of playing it back on a player piano; the piano roll keeps spinning, but you can generally be certain that nothing sane will come of it. In the case of *Gradius*, the result of the input desynchronizing with what the game state needed to be at that moment was instant death at the hands of a volcano. We set *Gradius* aside grudgingly and moved on to a TAS replay of *Mario Kart 64* led by Weatherton and micro500, which went flawlessly and restored our confidence.

The real stand-out piece at AGDQ 2014 was *Super Mario World*. Masterjun had somewhat recently at the time discovered an Arbitrary Code Execution (ACE) glitch in SMW that allowed him to take total control of the game using techniques impossible for humans to achieve, just like those more-than-ten-fingers-required piano roll compositions. Initially, we were just going to use it to jump straight to the end credits, but in the hours leading up to our portion of the event, he was able to pull off something a lot more interesting in the form of causing the game to glitch, go to a black screen, and then suddenly start playing *Pong* with Mario's head. This was followed up with an implementation of the classic game *Snake*. We even handed the controller off and had humans play to show it wasn't a trick.

Figure 6. TASBot with micro500's TASLink Board Held by dwangoAC

The compounded results of the event were beyond anything we had ever anticipated. News outlets including *ExtremeTech* and especially *Ars Technica*'s Kyle Orland covered the exploits and hundreds of thousands watched the shenanigans on YouTube after the fact. Over the course of eight GDQ events and several auxiliary appearances since then, TASBot has repeatedly demonstrated the amazing talent of hackers, coders

and brilliant minds like Ilari, p4plus2, total, Lord Tom and many others. TAS content at GDQ events has conservatively raised more than $400,000 for charity, and for their part, GDQ events have raised more than $16M for charity while often breaking $2M in a single event with live viewership often exceeding 200,000 live viewers at any given time. Multiple new replay devices have been made like true's unique standalone multireplay device, micro500's TASLink 4-port and extremely versatile device, and total's PSoC5-based high datarate devices.

Personally speaking, TASVideos site staff members, including Nach and Mothrayas, invited me to join them, and I came on staff as the official TASVideos Ambassador. I went on to present what we had done at DEFCON, GeekPwn, Thotcon and other conferences with the help of graphics from Ange Albertini and showed how hunting for glitches in games using TAS techniques can help teach reverse-engineering skills.

The Many Ways TASBot Plays

I'm still astounded at what the teams I've led for have accomplished. I may be the primary organizer and presenter, but none of what we've done ever could have been possible without the talent and drive of a huge team of folks. TASBot.net has a full listing where you can see some really wild stuff:

- *Super Mario Bros.* 1, 2, 3 and *Lost Levels* by agwawaf beaten simultaneously with the same sequence of button presses using micro500's TASLink 4-port replay device.

- *Super Mario Bros.* being played **inside** *Super Mario World* thanks to a truly inspired setup conceived and executed by p4plus2.

- micro500's *Brain Age* consisting of artwork like drawing a picture of a dog's head and still answering math problems with the correct answer.

- *Pokemon Plays Twitch* from p4plus2 and many others where we messaged Twitch chat with Linux scripts and displayed it on Pokemon Red inside a Super Game Boy.

- Sk'Hype where two NES consoles for stereo audio plus an SNES for video were used to perform a remote video call scripted entirely from Linux at 10 fps with help from a truly massive team spearheaded by micro500, MediaMagnet, p4plus2, total, Ilari, fuzyll and many more.

It's the Cables

So if you're feeling masochistic, you may be asking yourself if you can do this at home on your own retro console. The answer is a fairly easy yes in one sense, as several

Figure 7. Your cables probably will look something like this.

reference designs exist from which to choose—the PSoC5 DevKit board in particular is only $15, although it can be tricky to flash with total's firmware load using Linux. That's not the biggest problem, however, as that honor goes to cables. The biggest issue is that the higher datarate runs require one or sometimes even two extra NES or SNES data lines that are usually used only for things like the light gun and aren't populated on extension cables, which are the easiest way to connect to the console ports. It's surprisingly difficult to add extra wires to the connectors, because they often are nothing more than thin foil to allow the controller cables to be flexible.

Even if you stick to simple runs that require only a single data line, it can require rather delicate soldering work to connect an extension cable to something durable enough to hang off a breadboard or directly off a PSoC5. Beyond that, dealing with the 5v nature of the consoles also can be a bit of a challenge. We've often thought about what it would take to mass produce a replay device that connects directly to NES or SNES consoles, but so far, it isn't clear if it would be market-viable.

But...Linux?

With all of that context out of the way, it's finally time show something involving a Linux terminal. It's worth mentioning that every GDQ event has been primarily driven by Linux aside from some obvious exceptions, such as playing back the Windows version of VVVVVV using the Hourglass rerecording framework. Some runs have used Linux the same way the first *Gradius* run did as merely a method to shove a bitstream of data stored on a Linux device through a USB-attached serial interface, but several payloads have done far more extreme things.

There's one setup in particular that pushes an unmodified SNES to the limits—a full 16-bit 32kHz stereo audio player I've dubbed JukeSNES created by total using ideas and snippets of code from a large swath of hackers who came before him. If you want to play along at home, you'll need a rather daunting list of equipment including an SNES, a copy of *Super Mario World*, every dataline on both controller ports connected to cables, and a device fast enough to handle the datarate like the PSoC5 design from total.

All of the scripts and code listed below can be found in this GitHub repository. After

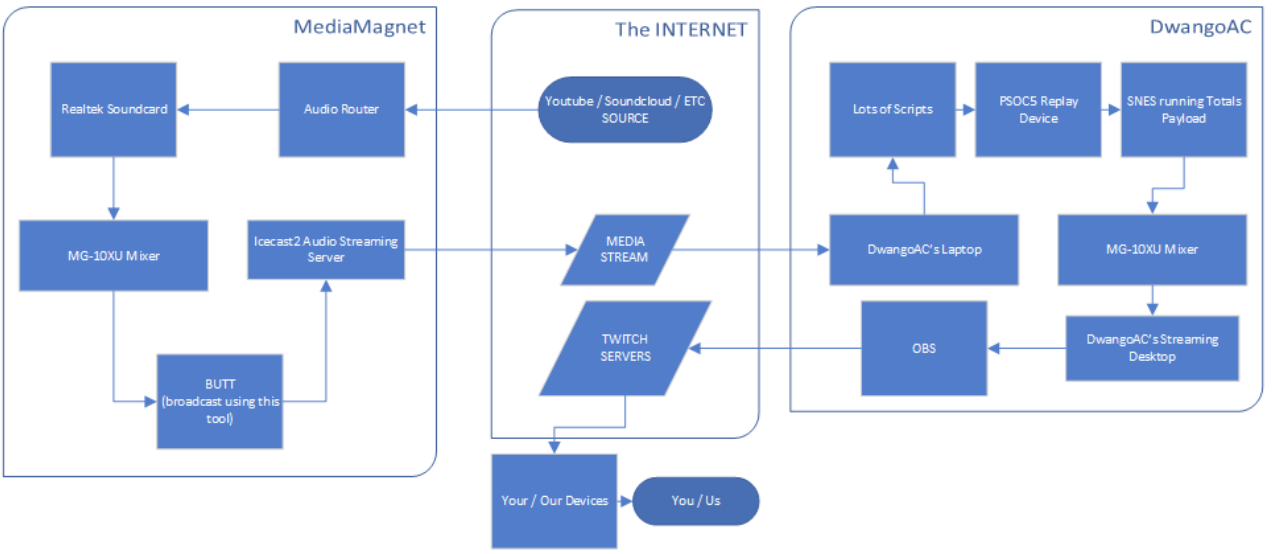


Figure 8. Why keep audio routing simple if you can make it ridiculously complex?

connecting everything with the power off and no saves on the SMW cartridge, run `python3 play_r16y.py /dev/ttyACM0 smw_stereo_pcm_v6.r16m`. The first argument to the script is the serial interface the replay device is connected to, and the second argument is the file containing the bitstream of data to send consisting of the raw string of bits that needs to be sent to the console. (As a side note, there are separate "dump scripts" written for individual emulators that create these bitstream files, but the files themselves contain no metadata about how many controllers are present, so there's an ongoing desire to find someone to step up and create a headerfile format.)

Once everything is staged and connected, the next step is to power on the console. If everything goes right, the game will start, and Mario will appear to do some rather strange and random things in the first level, eventually triggering a glitch that causes the screen to change to a static view showing text about playing audio. The next step is to issue a command similar to this:

```
ffmpeg -i http://radiotasbot.com:9989/ocremix" -hide_banner
 ↪-v quiet -f s16le -ac 2 -ar 32000 -acodec pcm_s16le - |
 ↪./tasbot_stream_snes_stereo16.py
```

where the section after `-i` can be any ffmpeg source. There's a lot to unpack here, but essentially what's happening is ffmpeg is massaging the audio data to use stereo, 16-bit, 32-kHz raw PCM audio, which is piped into the streaming script, which then rearranges the data into exactly the bitstream format we need it in for the replay device. The result is beautiful sounding audio that demands every last clock cycle out of the poor SNES. I personally use the device to play background music with the help of MediaMagnet when I'm streaming on Twitch just for the sheer geek cred, but even I confess it's absurdly complex.

The TAS Journey from Here

The road to this point has been long. There have been some painful losses of past volunteers. There have been times where we just couldn't get an idea to pan out (*Donkey Kong Country*, I'm looking at you). I've personally made some leadership decisions I now regret. Despite all the missteps, desynchronized runs and bungled performances, I'm proud of what the teams have done.

Processors are now fast enough to emulate consoles as new as the Wii using the Linux-compatible Dolphin emulator for Wii and Gamecube, and 2018's GDQ events have featured TASBot playing Gamecube games as a result. Miraculously, after I started writing this article, a new Linux SDL rerecording framework named libTAS was created by Clement Gallet, which makes it possible to TAS even complex OpenGL games directly within Linux. I'm continually amazed at the levels of perfectionism on display in new submissions at TASVideos. The vibrant community at #tasbot on Freenode IRC and bridged at http://Discord.TASBot.net as well as my own Twitch streams at http://Twitch.tv/dwangoAC has bloomed into an amazing example of awesome people with a shared vision—namely using TAS content to raise money for charitable causes. It's been a personal blessing to be a part of something greater than myself and an absolute joy that it has involved my two favorite things: video games and Linux. ■

Allan Cecil (dwangoAC) is the President of the North Bay Linux Users' Group. He acts as an ambassador for TASVideos.org, a website devoted to using emulators to complete video games as quickly as the hardware allows. He streams at Twitch.tv/dwangoAC and participates in Games Done Quick charity speedrunning marathons using TASBot to entertain viewers with never-before-seen glitches in games. By day, he is a senior engineer at Ciena Corporation working on OpenStack NFV orchestration and Linux packet performance optimization testing.

Resources

- TASVideos.org
- Morimoto's Run
- TASVideos Official Site History
- Doom Done Quick
- Black MIDI
- YouTube Video of *Super Mario Brothers* Played on a BS2 by jaku@hax.by
- Instructables Guide for NESBot: Arduino Powered Robot Beating *Super Mario Bros* for the NES
- SpeedDemosArchive.com
- GamesDoneQuick.com
- TASBot.net
- Arbitrary Code Execution (ACE) glitch in SMW
- GitHub Repository: snes-pcm-streaming — Tools and Code for Streaming High-Quality PCM Audio to the SNES through the Controller Ports
- Linux-compatible Dolphin emulator for Wii and Gamecube
- libTAS, Created by Clement Gallet
- http://Discord.TASBot.net
- Author's Twitch Streams at http://Twitch.tv/dwangoAC

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

Review: *Thrones of Britannia*

A look at the recent game from the *Total War* series on the Linux desktop thanks to Steam and Feral Interactive.

By Marcel Gagné

Back in 878 CE (or AD, if you prefer), the British Isles and England were far more exciting than what we see there today, especially if you are into historical dramas with lots of kings and knights in armor fighting for God and glory. Throw in Vikings launching an attack on the adolescent England, the retreat of its king who then gathers support from the provinces for a decisive counter-offensive against the horned warriors, and it makes for pretty cool story telling. Even if it is all true.

In January of that year, Vikings did attack at Chippenham, which is a little Northeast of Bath and Bristol. Alfred the Great, King of the Anglo-Saxons, was caught by surprise and retreated only to gather reinforcements from the counties of Somerset, Wiltshire and Hampshire. In May, Alfred's combined forces met King Guthrum's army in Edington in Wiltshire, formerly known as Ethandun. To make a long story short, the Brits won, the Vikings lost, and the rest as they say, is history.

The cool thing about fiction, and video games for that matter, is that you can take a decisive period in history such as this and re-imagine it by asking what would have happened if the Vikings had won. Or maybe some lord from one of the counties would have seen this as a good opportunity to get rid of Alfred, make some backroom deals with other counties and districts, and maybe even the Vikings, and you'd have a very different England today.

Figure 1. Choose your faction.

Total War Saga: Thrones of Britannia by Creative Assembly and SEGA, along with Feral Interactive (who brings us the Linux version of the game), asks us to envision these types of scenarios and rewrite history. *Thrones of Britannia* is a turns-based strategy game, with real-time elements, that asks you to join one of several factions (the British, the Welsh or even the Vikings) from which to build your empire (Figure 1). Each faction comes with its own regional strengths, goals, features and troops. No faction gives you a definitive advantage over the other, so take the time to read up on each and see what feels right to you. So far, I've played the Welsh and Gaelic kingdoms.

A substantial part of the game involves planning, researching, growing and building up your forces for the inevitable battles. You start out with only a small force that you must build, slowly, over time. This happens while you deal with the politics of the region, appease regional leaders, feed your people, acquire resources, keep the economy going and pretty much do anything that concerns the development of a

Figure 2. A Richly Detailed Landscape and Seascape

young country. While you're busy with all those things, you need to figure out how you're going to build up the kind of force that will let you take decisive control of this not-so-merry old England.

Game play happens over a beautifully detailed map of the British Isles (Figure 2) with cities and towns visible on the landscape. Your vantage point is overhead, but you can zoom in and out, pan, and even drop down to ground level, which is pretty cool when you get to actual fighting. The whole thing creates a feeling of place that adds to the immersive aspects of the game.

Finally, there's the fighting. In case I haven't made it clear, this is a historical battle simulation. You won't find wizards sending mystical bolts into the enemy line or towering heroes crushing ordinary forces under their mighty toes. It's a slow grind, recruiting troops, growing and taking care of your people, making sure they don't starve to death, and keeping them happy so they don't turn and desert you long

Figure 3. Up Close and Personal on the Battlefield

before the going gets tough.

Then, on the field of battle, it's all strategy, timing and a little bit of luck. A bigger army doesn't immediately guarantee you a win. You select and control individual battle groups, direct them toward the enemy line, and modify your strategy in real time. Pan, rotate and even drop down to ground level to get as close to the action as possible (Figure 3).

The game is played via campaigns—solo and against other players. If you don't want to wait, the game does have a "quick battle" mode that pits you against the game's AI.

Technical Details

The game will benefit from a good accelerated graphics card and probably best with your manufacturer's proprietary driver, but even on my Acer notebook (Core i5, 12GB of RAM and GeForce 940MX card, running Kubuntu 18.04), it performed well

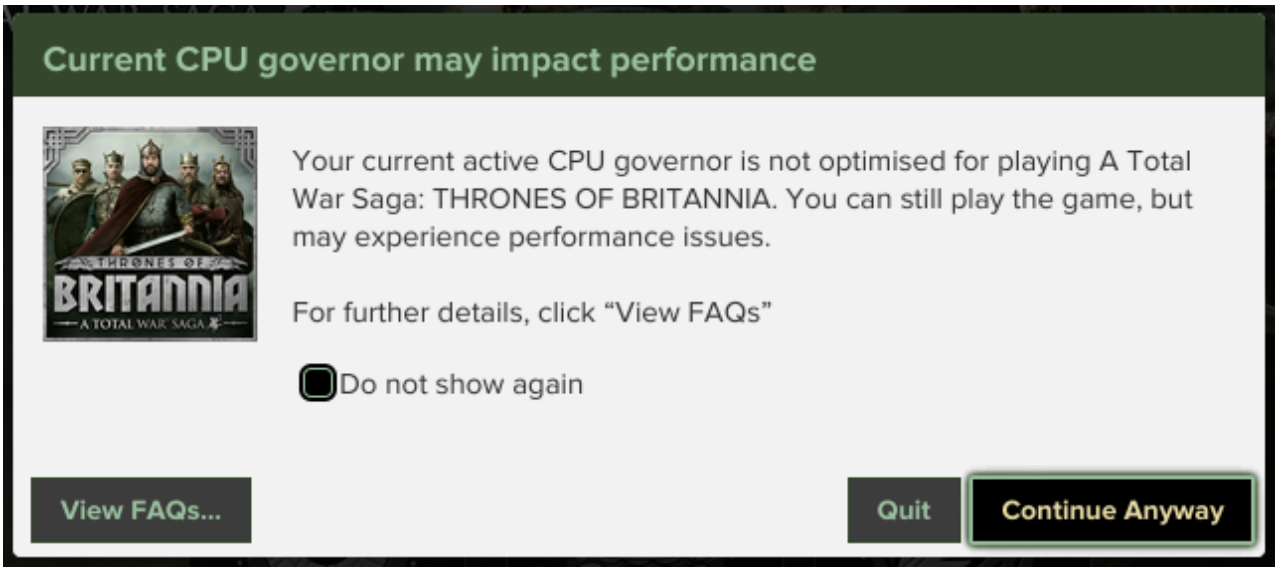


Figure 4. Warning! Marcel runs sub-standard hardware.

enough. However, I did get warned about potential performance issues whenever the game started (Figure 4). To alleviate potential lags, the game offers plenty of options and tweaks, so chances are, it will play well for you too.

I probably also should mention that the Linux version of the game is available through the Steam store, so you will need to install and run the Steam engine. Luckily, this isn't a big deal with most modern Linux distributions—for instance, Steam is right there in the Ubuntu repositories. I've played Steam games on Fedora, Arch, openSUSE and a few others.

Final Thoughts

Thrones of Britannia has a lot going for it. It's complex, wonderfully detailed and totally immersive, dropping you right into the middle of battle where you can see your strategy play out. Unfortunately, to me at least, that felt like part of the problem. The gameplay feels slow and dry, despite the historical depth and richness of detail. Over and over, I found myself going back to the "quick battle" mode in order to sharpen my skills without all that waiting.

For the *Total War* fan who is dedicated to realism, tactics and head-down strategy, the game surely offers a great deal more. This is a thinking player's game, not something you fire up for a quick skirmish or to dispatch a few vikings for relaxation. It also helps if you happen to have played games in the *Total War* series before this one, because as a starting point, let me just say that daunting might be a good word to describe the challenges that await you. There's a lot here, and there's a huge learning curve as well. *Thrones of Britannia* is impressive, but it may not be for everyone. ■

Marcel Gagné is Writer and Free Thinker at Large. The Cooking With Linux guy. Ruggedly handsome! Science, Linux and technology geek. Occasionally opinionated. Always confused. Loves wine, food, music and the occasional single malt Scotch.

Resources

- *Thrones of Britannia from Feral Interactive*
- Thrones of Britannia from the Steam Store

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

ModSecurity and nginx

nginx is the web server that's replacing Apache in more and more of the world's websites. Until now, nginx has not been able to benefit from the security ModSecurity provides. Here's how to install ModSecurity and get it working with nginx.

By Elliot Cooper

Earlier this year the popular open-source web application firewall, ModSecurity, released version 3 of its software. Version 3 is a significant departure from the earlier versions, because it's now modularized. Before version 3, ModSecurity worked only with the Apache web server as a dependent module, so there was no way for other HTTP applications to utilize ModSecurity. Now the core functionality of ModSecurity, the HTTP filtering engine, exists as a standalone library, libModSecurity, and it can be integrated into any other application via a "connector". A connector is a small piece of code that allows any application to access libModSecurity.

A Web Application Firewall (WAF) is a type of firewall for HTTP requests. A standard firewall inspects data packets as they arrive and leave a network interface and compares the properties of the packets against a list of rules. The rules dictate whether the firewall will allow the packet to pass or get blocked.

ModSecurity performs the same task as a standard firewall, but instead of looking at data packets, it inspects HTTP traffic as it arrives at the server. When an HTTP request arrives at the server, it's first routed through ModSecurity before it's routed on to the destination application, such as Apache2 or nginx. ModSecurity compares the inbound HTTP request against a list of rules. These rules define the form of a malicious or harmful request, so if the incoming request matches a rule,

ModSecurity blocks the request from reaching the destination application where it may cause harm.

The following example demonstrates how ModSecurity protects a WordPress site. The following HTTP request is a non-malicious request for the index.php file as it appears in Apache2's log files:

```
GET /index.php HTTP/1.1
```

This request does not match any rules, so ModSecurity allows it onto the web server.

WordPress keeps much of its secret information, such as the database password, in a file called wp-config.php, which is located in the same directory as the index.php file. A careless system administrator may leave this important file unprotected, which means a web server like Apache or nginx happily will serve it. This is because they will serve any file that is not protected by specific configuration. This means that the following malicious request:

```
GET /wp-config.php HTTP/1.1
```

will be served by Apache to whomever requests it.

This is where ModSecurity offers protection to an application accepting HTTP data. In this case, the free, core ModSecurity ruleset contains rules to deny any HTTP request that attempts to access any sensitive file in a WordPress installation. The core ruleset also contains rules for another popular CMS, Drupal.

The core ruleset also contains rules covering the many other ways that HTTP requests can be constructed maliciously to gain access or confidential information from a website. These methods include SQL injection, vulnerability scanning, Java and PHP exploits and many more. ModSecurity also supports

custom rules, so you can protect your HTTP application against specifically targeted attacks by writing your own rules.

First let's install the core ModSecurity library, libModSecurity, and then let's install the nginx connector that enables nginx to use ModSecurity. Before version 3, it wasn't possible to use ModSecurity with nginx. If you are using Apache2, you should continue to use ModSecurity version 2, as the Apache2 connector is still quite buggy and not recommended for production use.

Compiling and Installing libModSecurity

ModSecurity3 isn't available via the package manager for any of the major Linux distributions. Instead, you'll need to clone the ModSecurity GitHub repository and build the library from its source code. Before you can do that though, you must install all of the required build tools and dependencies. The following list of packages provides all of the required and most of the optional discrepancies on Debian and Ubuntu distributions: bison, flex, make, automake, gcc, pkg-config, libtool, doxygen, git, curl, zlib1g-dev, libxml2-dev, libpcre3-dev, build-essential, libyajl-dev, yajl-tools, liblmdb-dev, rdmacm-utils, libgeoip-dev, libcurl4-openssl-dev, liblua5.2-dev, libfuzzy-dev, openssl and libssl-dev.

Note that some of those packages have different names on Red Hat-based distributions. This page will help you figure out what the specific package names are.

After installing those packages, you can move on to compiling the library. These instructions are distribution-agnostic.

First, clone the libModSecurity git repository, which will download all the source code you need to build the libModSecurity. Use the /opt/ directory as the destination for all source code. Move to the /opt/ directory, and clone the libModSecurity git repository with the following commands:

```
cd /opt/
git clone https://github.com/SpiderLabs/ModSecurity
```

Next, move into the new directory that you created when you cloned the ModSecurity repository, and switch to the v3 branch. You'll also need to pull in a couple necessary sub-modules:

```
cd ModSecurity
git checkout v3/master
git submodule init
git submodule update
```

You're now ready to build libModSecurity. This should be a familiar process to anyone who has compiled a program from source code. You need only the following three commands to compile and install the library:

```
sh build.sh
./configure
make
make install
```

The `make` command takes a few minutes if you are running this on a modest virtual server. The libModSecurity library now is installed at /usr/local/modsecurity/lib/libmodsecurity.so. However, it can't do anything until you install an application and connector that will redirect the HTTP data to the libModSecurity library along with some rules. The next section looks at installing the nginx connector and the core ruleset provided by the ModSecurity developers.

Compiling the nginx Connector

Let's compile the nginx connector by utilizing nginx's capability of dynamic loading of third-party modules. nginx has been able to do this since version 1.11.5. This version or one higher is not available from the standard repositories of most of the major distributions. nginx provides repositories for the current stable releases of Red Hat/CentOS, Debian and Ubuntu that contain a version that supports dynamic module loading. This page lists these repositories along with information for adding the nginx repository to your distribution. After you have added the nginx

repository to your repository configuration, you need to install nginx using your package manager. When you have installed nginx, find the version you installed with this command:

```
nginx -v
```

When you have the version number, change to the /opt/ directory and download the source code that matched your nginx version from this page, and unpack the archive that you downloaded.

Next, you need to clone the git repository for the ModSecurity nginx connector. From the /opt/ directory, run the following command to clone this repository:

```
git clone https://github.com/SpiderLabs/ModSecurity-nginx
```

Now change into the new directory that you created when you unpacked the nginx source archive. In that directory, run the following commands to compile the connector:

```
./configure --with-compat
 ↪--add-dynamic-module=/opt/ModSecurity-nginx
make modules
```

Now you need to copy the connector module into the nginx modules directory with this command:

```
cp objs/ngx_http_modsecurity_module.so /etc/nginx/modules/
```

Now that you've compiled the nginx connector and copied it to the right location, you need to configure nginx to use it. In addition, you also need to download the rules that libModSecurity will use to filter the HTTP data.

First, move to the nginx configuration directory:

```
cd /etc/nginx/
```

and add the following line to the nginx's main configuration file at /etc/nginx/nginx.conf:

```
load_module modules/ngx_http_modsecurity_module.so;
```

You need to put this line in the first section under the line that begins `pid` and not in either the `events` or `http` sections.

Next, create a new directory and load the ModSecurity rules and configuration into it:

```
mkdir /etc/nginx/modsec
cd /etc/nginx/modsec
git clone https://github.com/SpiderLabs/
↪owasp-modsecurity-crs.git
```

Use the ModSecurity rules configuration file that was downloaded from the git repository by renaming it with the following command:

```
mv /etc/nginx/modsec/owasp-modsecurity-crs/
&rarrhkh;crs-setup.conf.example /etc/nginx/modsec/
↪owasp-modsecurity-crs/crs-setup.conf
```

Now you need to copy the ModSecurity configuration file from the directory where you built libModSecurity to /etc/nginx/modsec/:

```
cp /opt/ModSecurity/modsecurity.conf-recommended
 ↪/etc/nginx/modsec/modsecurity.conf
```

Finally, create a new configuration file that loads these two configuration files and all the rules files. This file will be invoked by a couple lines in an nginx server configuration block, which will invoke the use of ModSecurity. Create and start editing

this file with a text editor:

```
nano /etc/nginx/modsec/main.conf
```

Add the following three lines to this file:

```
Include /etc/nginx/modsec/modsecurity.conf
Include /etc/nginx/modsec/owasp-modsecurity-crs/crs-setup.conf
Include /etc/nginx/modsec/owasp-modsecurity-crs/rules/*.conf
```

You've now completed building and installing nginx, libModSecurity, the nginx connector and ModSecurity rules. Now you can start or re-start nginx to load the new configuration. If everything is working, you won't see any errors printed when you restart nginx.

Testing ModSecurity

Let's test ModSecurity by adding a couple lines to the "default" server and making a request that will be blocked by ModSecurity. The default server configuration is the configuration that nginx uses on installation and is listening only on localhost and not on the internet-facing network interface. This makes it secure to start nginx before any custom server configuration has been created, because the default configuration is inaccessible from the internet.

The default server configuration is located at /etc/nginx/conf.d/default.conf. Open this file with a text editor, and add the following two lines under the `server_name` line:

```
modsecurity on;
modsecurity_rules_file /etc/nginx/modsec/main.conf;
```

Restart nginx again to load this new configuration. Now, all you need to do to test that ModSecurity is working is make an HTTP request that matches a banned rule.

ModSecurity has two modes of operation. The default is that it will only log any queries that match banned rules but allow them to pass to the application. This mode allows system administrators to run ModSecurity for a period and ensure that no false positive requests are getting blocked that would interfere with the normal operation of the website. ModSecurity records these requests that match banned rules to /var/log/modsec_audit.log.

You can create an HTTP request that will be recorded to that log file by using `curl` to make a request that contains a banned user agent header. The following command makes an HTTP request that contains the header "User-Agent: masscan". This is a banned user agent, so ModSecurity records that it witnessed a banned HTTP request. This command looks like:

```
curl -H "User-Agent: masscan" http://localhost/
```

nginx returns the default welcome page as raw HTML, but ModSecurity creates a lengthy log entry in /var/log/modsec_audit.log that begins:

```
ModSecurity: Warning. Matched "Operator 'PmFromFile'
 ↪with parameter 'scanners-user-agents.data' against
 ↪variable 'REQUEST_HEADERS:User-Agent' (Value: 'masscan' )
```

This indicates that ModSecurity successfully intercepted and matched the malicious HTTP request.

When you want to toggle ModSecurity from logging malicious HTTP requests to blocking them actively, edit the line in /etc/nginx/modsec/modsecurity.conf from:

```
SecRuleEngine DetectionOnly
```

to:

```
SecRuleEngine On
```

and restart nginx. Now the same `curl` request will result in a 403 error:

```
curl -H "User-Agent: masscan" http://localhost/
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.12.2</center>
</body>
</html>
```

The blocked request also will be logged to /var/log/modsec_audit.log.

Additional ModSecurity Connectors The new modular nature of ModSecurity means that any application that accepts or processes HTTP data can use ModSecurity and its rules to analyze HTTP data. At the time of this writing, ModSecurity v3 has been of release quality only for a few months, so there aren't many additional connectors that enable applications to hook into libModSecurity.

The Google Summer of Code has produced a couple interesting new connectors. The first extends the ability of Snort v3 to use the ModSecurity rules. Snort is an intrusion-detection and real-time packet-sniffing and logging application. This connector allows Snort to send captured HTTP data to libModSecurity and get it checked against the ModSecurity ruleset. The home page for this project is here.

A second Google-sponsored connector targets the node.js server. Node.js is a JavaScript runtime that enables the creation of scalable network applications. This connector routes all inbound HTTP requests via ModSecurity, thereby adding a security layer to the Node application. You can read more about this project at its home page.

The release of ModSecurity v3 has transformed ModSecurity from an Apache module to a flexible application that is easily leveraged by any application that accepts HTTP data. Given that more and more of the applications that people depend on are moving from their local computers into data centers, the need to ensure the security of those applications and that data is becoming increasingly important. ■

Elliot Cooper has worked for 18 years as a Linux systems administrator and technical documentation writer for several Linux and open-sourced-based UK hosting and domain registration companies. He currently is working as a part-time English Teacher for the University of Da Nang in Vietnam, writing technical content and providing remote technical support for a Linux hosting company. When he's not working, he enjoys blogging about Linux server administration, reading and eating out with friends. You can contact him via his personal website or blog.

The freenode project celebrates its 20th anniversary this year at the second annual freenode #live conference

At We The Curious in Bristol, UK
November 3-4, 2018 — 9am Saturday - 6pm Sunday

Keynote speakers include Bradley M. Kuhn, Chris Lamb, Kyle Rankin, Leslie Hawthorn and VM Brasseur! More to come...

Registration and call for participation is open now at

HTTPS://FREENODE.LIVE

What Is the Point of Mozilla?

Glyn Moody has been writing about the internet since 1994, and about free software since 1995. In 1997, he wrote the first mainstream feature about GNU/Linux and free software, which appeared in *Wired*. In 2001, his book *Rebel Code: Linux And The Open Source Revolution* was published. Since then, he has written widely about free software and digital rights. He has a blog, and he is active on social media: @glynmoody on Twitter or identi.ca, and +glynmoody on Google+.

Is Mozilla a software organization or an advocacy group?

By Glyn Moody

Few journeys in the world of open source have been as exciting as Mozilla's. Its birth was dramatic. Netscape, the pioneering company whose Netscape Navigator browser shaped the early Web, had enjoyed the most successful IPO up until then, valuing the 18-month-year-old company at nearly $3 billion. That was in 1995. Three years later, the company was in freefall, as the browser wars took their toll, and Microsoft continued to gain market share with its Internet Explorer, launched alongside Windows 95. Netscape's response was bold and unprecedented. On January 27, 1998, it announced that it was making the source code for the next generation of its web browser freely available under a GPL-like license.

Although of huge symbolic importance for the still-young Free Software world—the term "open source" was coined only a month after Netscape's announcement—the release and transformation of the code for what became the Mozilla browser suite was fraught with difficulties. The main problem was trying to re-write the often problematic legacy code of Netscape Navigator. Mozilla 1.0 was finally released in 2002, but by then, Internet Explorer dominated the sector. The failure of the Mozilla browser to make much of an impact ultimately spurred development of the completely new

Firefox browser. Version 1.0 was launched in 2004, after three years of work.

Microsoft's failure to update its flabby Internet Explorer 6 browser for more than five years meant that successive releases of Firefox were steadily gaining market share—and fans. As I wrote in *Linux Journal* in June 2008:

> Three things are striking about the recent launch of Firefox 3. First, the unanimity about the quality of the code: practically everyone thinks it's better in practically every respect. Secondly, the way in which the mainstream media covered its launch: it was treated as a normal, important tech story—gone are the days of supercilious anecdotes about those wacky, sandal-wearing free software anoraks. And finally—and perhaps most importantly—the scale and intensity of participation by the millions of people who have downloaded the software in the last week.

My hope then was that the evident success and enthusiasm would drive Firefox to ever-greater market share, and help spread open source and its values to a wider audience. As the Statcounter graph of browser market share worldwide shows, Firefox did indeed continue to achieve greater market penetration for a while. In November 2009, it held around 32% of the sector globally. But since then, Firefox's market share has steadily but inexorably fallen; it now stands at around 5%. Google's Chrome, meanwhile, has ascended to 59%. Even if we are rightly doubtful about the detailed accuracy of those figures, the trends are inarguable: Firefox peaked a decade ago and shows no sign of halting its slow decline.

The Mozilla project has stumbled in various ways during that time. Valuable energy and resources were diverted to the Firefox phone project, which started in 2013 and closed in 2016. In 2014, Mozilla foolishly flirted with placing ads on Firefox. The next year, Mitchell Baker, Chair of the Mozilla Foundation and self-styled Chief Lizard Wrangler, posed the question of "whether Mozilla remains the best organizational and legal home for Thunderbird", Mozilla's standalone email client. This caused many to wonder whether Thunderbird's days were numbered. Fortunately, in 2017, Mozilla confirmed that it would continue to "serve as the legal and fiscal home for the Thunderbird project".

Perhaps Mozilla's biggest blunder was its decision to add support for the closed-source DRM W3C standard Encrypted Media Extensions (EME) in Firefox. As well as ignoring good technical reasons why this was the wrong thing to do, Mozilla's blessing of EME effectively legitimized DRM and validated it as a standard part of the hitherto open web. Although supposedly "only" for video streams, EME sets a precedent. Given the insatiable appetite the copyright industry has for control, it seems only a matter of time before DRM is applied to web pages themselves—no copying allowed. What's particularly sad is Mozilla's weak attempt in 2014 to justify the move:

> We have come to the point where Mozilla not implementing the W3C EME specification means that Firefox users have to switch to other browsers to watch content restricted by DRM.

Despite Mozilla's kowtowing to the video streaming industries, Firefox users have continued to switch to other browsers anyway: market share has dropped from 14% in 2014 to today's woeful 5%. In other words, Mozilla betrayed its core mission to preserve the open internet, for no gain whatsoever.

Meanwhile, Mozilla has flourished financially. The most recent "State of Mozilla" report says that in 2016 various deals with search engines brought in an astonishing $520 million. And to its credit, Mozilla has started to deploy those resources in all kinds of interesting and innovative ways. Reflecting this, in 2016, it released its strategy document "Fueling the Movement", with the subtitle "Ensuring the Internet is a global public resource, open and accessible to all". That's rather ironic, given its support a couple years earlier for DRM, which has the sole purpose of making online material private, closed and inaccessible for most people.

The 2016 State of Mozilla report describes some of the organization's more recent work in the field of advocacy:

> The Mozilla Foundation runs campaigns to educate and empower citizens across the web. In 2016, this included an education campaign on the critical role encryption plays in

everyday internet life. Launched amidst the February 2016 Apple v. FBI case, campaign videos simplified complex online security issues for the public and the media. Over the course of 2016 and 2017, Mozilla also ran campaigns around the need for more creative, internet-friendly copyright in the European Union and on the continued importance of protecting net neutrality in the United States. In late 2017, Mozilla launched a "privacy not included" holiday shopping guide reviewing toys and other electronics—this was a first step in a large scale plan for consumer engagement on data and privacy issues.

The organization launched what it called its "Mozilla Manifesto" back in 2007, which includes the first hints of an increasing emphasis on work outside the software field. The 2016 State of Mozilla report, and the projects that have followed it, confirm that this is an increasingly active area for the project. That's hugely welcome; privacy, encryption, net neutrality and security are under threat as never before. But it raises an important question: should Mozilla be a software project that uses some of its resources for key advocacy work or an advocacy organization funded by its programs?

Mozilla's ill-judged adoption of the EME standard downgraded Firefox from a long-standing beacon of software freedom to a well-featured browser much like any other. By contrast, Mozilla's not-for-profit status, exceptional financial resources, and worldwide network of smart people passionate about the open internet and its values, mean that it has unique advantages as a trusted advocacy organization.

Open source has won, so Mozilla's original mission to promote free software is no longer a priority. Instead, what we do desperately need is a powerful, truly independent voice willing to speak up for ordinary users of the internet on today's key issues and to defend uncompromisingly their broader interests in the online world. Mozilla is probably the only organization capable of doing this credibly. It urgently needs to broaden and deepen its already substantial advocacy activities yet further—for the common good, and its own relevance. ■

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