How to get started with open source

A guide for beginners in open source
Open Voices, Issue 9
Opensource.com

Copyright

Introduction

Taking the first steps in any journey can be scary. There are new obstacles to hurdle, unfamiliar landscapes to traverse, and usually the destination is shrouded in mystery. But that's also what makes new ventures exciting and worthwhile.

The collection of stories in this eBook are about striking out on the open source way. Whether you're an individual, seeking information on moving away from closed software, or an organization looking for free and open alternatives to the utilities that help your business succeed, these are stories about finding your footing in the world of open source. Several are “origin stories” about how experts discovered open source, and how they've flourished in the time since. Others are focused guides on finding the right software, using a specific open source tool, or introducing your neighborhood or town to the power of the open source way.

Remember that it's not easy to get started with anything new, but, as an old Buddhist saying goes, “It is better to travel well than to arrive.” We hope these stories will help you travel well along the open source way, and keep you traveling for years to come.

Open source as an alternative for small businesses

By Aseem Sharma (originally published February 2014)

Is it safe to use? What alternatives do I have? Is it easy to install?

These were some of the questions asked by Amandeep, a New Delhi based owner of a small scale clothing company, when I pitched to him a few open source solutions that could make his day-to-day operations more efficient. For someone without any IT background (but a sharp business sense), these were brilliant and relevant questions. The answers to these questions won't just help Amandeep, but if shared broadly may help reduce the apprehension of a significant number of small scale business owners, especially in India. My interactions have shown that a lot of these businesses are looking to grow, enhance their productivity, and most importantly, save costs.

Approximately 7,500 miles away from Amandeep in New Delhi lives Nabeel Hussain. Nabeel, a graduate of Conrad Business, Entrepreneurship and Technology Centre, is a new product development and digital marketing specialist actively engaged in Waterloo, one of the top entrepreneurial ecosystems in the world. As an entrepreneur, he is always faced with the challenge of managing limited resources while building traction. He has a plethora of technology solutions at his disposal, and the technical know-how to utilize these solutions. Additionally, he has a robust support system to advise and guide him to the best available solution that fits his needs. For Nabeel, open source solutions provide an inexpensive alternative for crafting early stage prototypes for his ideas and validating them with customers. From using WordPress and its library of plugins, to venturing into OpenShift Origin and Joomla, he has the knowledge to make use of top notch technology to reduce risk, manager resources, and build traction for his venture.

These two different scenarios indicate a categorical gap in the knowledge of entrepreneurs when it comes to adopting open source solutions. Although there is some geographic gap between the entrepreneurs in the developed and the developing world, as well as a gap that spawns from business exposure/experience, the problem is wider than that. There is a difference in productivity and efficiency between entrepreneurs who utilize open source solutions and those who do not. The situation becomes clear when we look at those small scale business owners who are technology pros versus those who are not.

A significant number of businesses, in India in particular and in the developing world in general, are of a *mom-and-pop business* nature. Based on my recent interactions with these small scale business owners, I see widespread misconceptions pertaining to open source software. The questions that Amandeep from New Delhi asked me are critical in nature. In order for small scale businesses to adopt open source solutions, it is vital to address these misconceptions.

Is open source software really safe?

The question arises from the basic process that is followed to write code using open source way. If any hacker can read your code, then why can't they use the knowledge to their personal benefit? Most of those sorts of malicious attempts fail because there are a lot of committed people looking over the source code, finding problems, and fixing them. More eyes tame bugs quickly. And security by obscurity is no security at all. What strikes me at this point of time are the words of security expert

Bruce Schneier, "Public security is always more secure than proprietary security…For us, open source isn't just a business model; it's smart engineering practice."

Developing code in an open source fashion is an expression of a technique. Software, in our world, should be treated as a service which can be customized based on the specific needs of a user, rather than merely as a product.

I know a lot of people involved at different levels of open source projects. All of them are driven by their commitment to reach technical and professional excellence, and to add to the existing body of technology knowledge. The entire ecosystem of open source is built on that commitment. The Linux operating system, for example, with its proven track record of stability and security, forms the backbone of complex infrastructures and data centers world over. The same benefits that help Linux and other open source tools succeed at the enterprise level can be reaped by small businesses, too.

A couple of months back, I read Thomas Friedman's *The World is Flat.* An otherwise helpful and insightful book, the author seems to host a thought process that open source is contrary to the developers' right to make a profit. A lot of people who think that way do not see the forest for the trees. They see free software, they see Linux, but they miss the multi-billion dollar ecosystem that surrounds open source. Its true that Brian Behlendorf, the person who orchestrated Apache web server, did not make a dime off it, but the immense value that this server has added to the economy and the legions of small to medium size businesses that use this infrastructure is an important contribution. Free software developed by a community is not tantamount to insecurity.

Are there quality alternatives available?

Gone are the days when open source was produced only by the engineers, for the engineers. From word processing to calendar applications to servers and to setting up telephone communication networks, small businesses can benefit hugely from open source solutions. Let us take the example of word processing, an activity that almost all small businesses, irrespective of their field, carry out.

Microsoft Word is the premium software in the area but it is cluttered with features that a lot of small businesses won't ever use. The bloating of Microsoft Word has cost its simplicity. There are easy to use, simple, free, and open source word processors available out their. A few of these that I have been using (and suggesting to small businesses) as an alternative to Microsoft Word are:

1. **Apache Open Office**: This software primarily consists of six tools for managing office tasks, namely: Writer as a word processor, Calc as a spreadsheet tool, Impress for multimedia presentations, Draw for diagrams and 3D applications, Base as a database tool, and Math for creating mathematical equations.
2. **AbiWord**: Developed in 1998 with the help of gtkmm, this open source word processor includes both simple word processing features to sophisticated features like multiple views, page columns, and grammar checking.
3. **LibreOffice**: This is my favorite and always at the top of my recommendation list for anyone looking for a free and efficient word processing suite. Although the features are similar to those of ApacheOpen Office, LibreOffice is better when it comes to community support.

There are dozens of other excellent alternative solutions to proprietary software and thousands of open source projects that can serve small businesses. It can sometimes be difficult to select the software which best matches specific needs, but there are plenty of people globally willing to help you make those decisions and help take small businesses down the path to an open and productive future.

How to contribute to OpenStack

By Rich Bowen (originally published February 2014)

Like any open source community, OpenStack has its local quirks when it comes to getting involved in the project. This is further complicated, in the case of OpenStack, by the fact that it's actually a collection of numerous smaller projects which are linked together via shared APIs, but which have independent objectives and developer communities.

This article isn't comprehensive, by any means, but it's an attempt to show you some of the places where you can plug into the community. It's especially important to note that you don't have to be a programmer to be a valuable contributor to OpenStack.

Mailing lists

Since it spans timezones, continents, and languages, most of the communication around OpenStack happens on mailing lists.

There are a LOT of mailing lists, but there's one main one: openstack@lists.openstack.org. You can subscribe or look at the archives. Rather than splitting technical discussion by sub-project, it's all in one place, with subject line tags where appropriate. For example, a discussion of Ceilometer would be indicated by [ceilometer] in the email subject line. The disadvantage of this approach is, of course, the high volume of the list—around 100 messages per day. The advantage, however, is that you don't have to subscribe to a dozen other lists to get a comprehensive picture of what's going on and risk missing out on some sub-project you were unaware of.

There are also lists for cloud operators, for foundation governance, security, and various other sub-teams. View the complete list.

Once you've decided what list(s) you want to be on, you should listen quietly for a while to get an idea of the tone of the conversation and who the main speakers are. Read the mailing list etiquette guide for tips on local cultural conventions.

Git and Gerrit

If you want to submit changes to the code or documentation, you'll need to be familiar with the Git workflow. Although the code repositories reside on GitHub, the patch process doesn't follow the Github pull request model you may already be familiar with. Instead, it uses Gerrit, which ensures that every change to any part of the OpenStack codebase is subjected to the same rigorous testing process, whether it's a major functional change or a tiny documentation tweak. View the fully documented workflow.

Every change is submitted to a system that requires at least two human reviews of the change as well as a successful completion of the testing suite. Once these requirements have been met, the code is automatically merged into the Git repo and becomes part of the master branch.

It is something of a pain to go through this setup process, but once you've done it once, you'll never

need to do it again, and the advantages are enormous.

ask.openstack.org

While discussions about the direction of the project happen on the mailing lists, Q&A conversations happen on http://ask.openstack.org/ in a Stack Overflow style forum. This is the place to go if you have questions about OpenStack, or if you wish to help answer others' questions.

Participants in the forum earn 'karma' points that give them access to greater levels of privilege and responsibility, such as moderating questions, editing questions and responses, and adding tags to conversations. The more you do, the more you can do.

IRC

If you prefer real-time conversation, there are a number of OpenStack-related channels on the Freenode IRC network. They are all fairly quiet, since ask.openstack.org carries the main weight of conversation, but there's generally a large crowd (837 on the #openstackchannel as I'm writing this) and someone will usually answer a question pretty quickly.

The #openstackchannel is for general discussion and Q&A. The#openstack-101 channel is for beginner questions that you might feel embarrassed asking elsewhere.

#openstack-community is a channel where people conduct meetings to discuss a wide variety of OpenStack-related topics. A meeting bot takes notes so that they can be reported back to various parts of the community. #openstack-community is a good place to find out about local OpenStack user groups in your area and discuss various community events.

Events

Speaking of events, there's always something going on somewhere related to OpenStack.

There are two main conferences every year, one in the United States and one elsewhere, where the next six months of OpenStack development is discussed and technical presentations are given on the various aspects of the project.

The next one of these is the OpenStack Summit which will take place in Atlanta, Georgia from May 12-16, 2014. Get more details and register to attend on the website. Then, the next Summit will be held in November 2014 in Paris, France. Many sessions are recorded. You can watch presentations from OpenStack Summit Hong Kong, last year.

In addition to these major events, smaller local events are always happening and listed on the site. OpenStack meetups can be two people meeting to chat to hundreds of attendees gathering for formal presentations. Regional user groups are listed and if there isn't one in your area, consider starting one up. All you need is another person who wants to meet to talk OpenStack.

Do something

In the end, if you want to get involved in OpenStack, or any open source project, the trick is to just do something. If you wait around for someone to tell you what to work on, you'll be waiting a long time. Step up and answer a question, fix a typo, patch a bug, or go to an event.

See you in the community!

Get started in open source online and offline

By Robin Muilwijk (originally published February 2014)

What skills do you need and which projects should you participate in as beginner in open source?

These are common questions for beginners to open source software, hardware, communities, and methodologies. New folks to open source can start their discovery online and offline. Events and projects of many different kinds will help beginners find what they are good at and allow them to get to know their own skills.

Get started with open source online

Codecademy

Codecademy is a website where you can learn several programming languages in an interactive way. Languages such as HTML, PHP, Ruby and Python are a few. With each language, you learn the basics, like syntax and commands, and by finishing assignments, you earn points and badges.

I can recommend Codecademy, as I signed up myself to follow the PHP lessons. The first lessons start real easy, and this course continues to teach you the most common commands and programming structure and syntax. Each lesson ends with practicing what you just learned. All you need for courses at Codecademy is your browser—no extra software is required.

Code School

Codeschool takes a different approach for learning; students take what they call "paths" to Ruby, Javascript, HTML/CSS and iOS. Where Codecademy provides its courses in online reading material, Codeschool presents them through video lessons and challenges.

Each "path" contains several lessons that take you through a specific programming language. Again, no extra setup is required, just the site and your browser. What makes Codeschool interesting is that they also provide a course in programming apps for the iPhone and iPad.

Code.org

Code.org is known for its Hour of Code program and offers similar courses such as Javascript and Python but also tutorials for beginners. These beginner tutorials let you solve puzzles in a Scratch-like environment based on the Angry Birds game, teaching you concepts like repeat-loops, conditionals, and basic algorithms.

Code.org clearly states the age category to which the courses apply, and the requirements to follow them, which in most cases is a browser or an iOS or Android device.

Scratch

For the youngest beginners (age 8+) in open source, there is the popular programming language Scratch.

> *Scratch is a programming language and an online community where children can program and share interactive media such as stories, games, and animation with people from all over the world. As children create with Scratch, they learn to think creatively, work collaboratively, and reason systematically.*

Scratch has a very user and kid-friendly interface. It teaches kids the very basics of programming. Scratch provides information for educators and parents, making it easy to adapt in classrooms or at home.

Get started with open source offline

Local User Groups

If you already have an interest for a specific open source programming language, or platform like Linux, local user groups are a great way to get introduced. These groups typically meet weekly to monthly. A great benefit of this offline approach is the ability to ask questions, share knowledge, and find guidance in what you are learning.

A great example of this are the Linux User Groups (LUGs) and PHP User Groups. Other well known open source projects have user groups as well, like Drupal and MySQL.

Hackerspaces and makerspaces

A hackerspace, also known as hacklab or makerspace, is a community led workspace. It's a place where people meet with a common interest, for example regarding computers, technology, or science.

Hackerspaces can be a great way of discovering the use and development of open source and open hardware. Different from a local user group, a hackerspace can be about more than one topic or interest. This provides a beginner in open source with the opportunity to explore several open source software or hardware projects, and thus, to find out where his or her interest lies.

Hackerspaces are easy to find, just search the Internet and you will most likely find one close to home that you can visit. Some hackerspaces run a website with a list of projects. This is a good way to search for something that you are interested in. To give you an idea, this is the project list for a hackerspace in Amsterdam (Netherlands).

Coderdojos

Exploring Coderdojos was inspired by two of my previously published interviews with Lune van Ewijk (Digital Girl 2013) and Julie Cullen (Ambassador for Ireland during Europe Code Week 2013).

> *"CoderDojo: The open source, volunteer led, global movement of free coding clubs for young people."*

CoderDojo is a non-profit global movement and was founded in 2011 by James Whelton and Bill Liao. Because CoderDojos are open source by nature, every Dojo is different and autonomous. Young people between the ages of 7 and 17 meet at Dojos to learn how to program apps, games, software, and more. In the true spirit of the open source way, CoderDojos are set up, run, and taught at by volunteers.

Note: Read more about how CoderDojos are about more than just coding in my interview with Lune van Ewijk. You'll find they can also be about robotics, playing with open source hardware like Arduino or Raspberry Pi boards, and learning the skill of soldering.

Online versus offline

Where the online options give you lots of opportunities to learn the beginnings in open source programming languages, it's the offline opportunities that really introduce beginners to all the open source projects that are out there.

Young beginners especially off to a great start if they try Scratch or visit a local CoderDojo or hackerspace.

Youth recommendations:

Age 6-8 and up: Code.org beginner tutorials, Scratch, Coderdojos

Middle school: Javascript and Python, hackerspace, user groups

High school: Apps for iOS, hackerspace, user groups

Building an open source community

By Mihai Guiman (originally published February 2014)

I have told the story of how FinTP, the first open source application for processing financial transactions, was born. Here, I would like to present a deeper view on how the community is being built, its structure and governance and why I think people should join such communities. I myself am a founding member of two communities, the first being built together with my friends after our first bachelor party, from a desire to preserve our spirit of joy and good vibes. The second one is the open source community around FinTP, called FINkers United.

An open source project cannot succeed without a powerful community that supports its development. At the same time, Eric Raymond said in *The Cathedral and the Bazaar* that a necessary pre-condition for success in an open source project is having something runnable and testable to play with. Therefore we have the chicken and egg dilemma of which comes first—the product or the community. Fortunately, FinTP had both, being the result of evolution from its previous commercial version to open source and having a company, partners' network and a user base that would become the starting point for the open community FINkers United.

While it may seem that the transition from the closed community to FINkers United is destined to be smooth, there are many adjustments to be made. The roles in the community have to change drastically, from the former way of interaction between supply and demand sides, in order to benefit from all the advantages an open environment has to offer—reduced time-to-market, better results delivered from a strong collaboration, cost savings. Of course, a coherent governance strategy has to be established so the project doesn't go off rails.

Community structure

Communities are groups of individuals sharing common interests. FINkers United is not different, all members having the same goal to improve the processing of financial transactions based on open source applications, pushing the interoperability level in order to achieve a new semantic standardization for financial transactions. For this specific community and mission, several professional profiles of individuals and institutions are welcomed: technical (developers, IT architects and designers, implementers, support, quality assurance), business (business knowledge in banking practice, standards, financial transaction processing, banking or corporate treasury, corporate to bank business etc.), communication, marketing, market analysts, auditing and legal consultants.

In order to successfully establish the strategy and objectives, to manage and control the activity of the community, the community we conceived has the following three governing entities: General Assembly, Board of Directors and Censor. In the early stages of the community, the governance is ensured by the founding company, since it is the sole contributor. For now, the community has a Technical Committee coordinated by the company's CTO, which includes 10 project leaders. In time, new hierarchies will emerge based on merit and contributions, so any member can be elected.

Reasons to join the FINkers United community

The most common question is what's in it for me, *why should I join this community?*

The rewards are very different from the traditional closed-source projects, where there is a clear limit. For suppliers, the types of reward-earning contributions are in the job description, for which you get a monthly paycheck. For the demand side, what you pay is what you get, with limited flexibility from both providers and budget owners.

In open source communities, the benefits are different for every member profile, but the essence is that everybody is involved because of having a shared interest and acts as an entrepreneur—having ideas, initiatives and the freedom to act accordingly to own beliefs.

For the *adopters*, the benefits of using and supporting open source software have been debated in length, and they include:

1. The power to influence projects and strategies
2. Freedom from vendor locking
3. Benefit from better, more secure and reliable products
4. Better TCO and time to market
5. Reduce development and maintenance cost
6. Attract and retain development talent
7. Community based, price affordable, service consistent, guaranteed quality 'escrow agent'

For *contributors*, the satisfactions are mostly triggered by a sense of accomplishment.

1. **Material**—custom professional services can be offered to adopters who lack certain skills or need SLAs
2. **Intellectual**—there’s great opportunity to develop the professional profile by taking on the technical challenges open source projects have to offer
3. **Recognition**—open source communities reward each of their members based on the value of their contributions
4. **Competitive**—the opportunity to work for state of the art technological projects, with high impact on several industries

Speaking for myself, my motivation for joining FINkers United is that being a part of the team that attempts to revolutionize the financial transactions processing arena is a great challenge and experience, not to mention that succeeding would be truly an accomplishment. Moreover, I believe that creativity is original thinking based on openly sharing knowledge and transparently sharing results.

Why would you join an open source community?

5 ways open source is transforming tech in 2014

By Eren Niazi (originally published February 2014)

For the last decade we've watched an epic contest unfold between open source and proprietary technology, and 2014 is the year that this dynamic will radically transform. The lines between open source and proprietary are becoming irrevocably blurred as proprietary firms pour resources into open source development and open source companies dial in their revenue models. Above all else, the open source community is producing the technologies businesses need to be competitive in the data-rich 21st century.

The open source movement, built on the 'marketplace of ideas' and the scalability of collaboration, is swallowing the proprietary world. To understand where the entire tech world is going, I believe we need to keep a close eye on these five open source trends:

1. Data storage clustering

In 2014, open source technology will allow datacenters to cluster together storage systems and thereby grow at a rapid pace. The software will allow us to clone an entire storage system, making it possible to bring new database servers online within minutes. Essentially, open source will crack the problem of absorbing high volumes of data at high speed.

This ability to build up storage space quickly is crucial because our digital universe is expanding rapidly. In a 2011, the International Data Corporation (IDC) found that the global volume of digital data is roughly doubling every 2 years, and by 2020 our digital universe will hold 40 trillion gigabytes. The amount of information managed by enterprise datacenters would grow 50-fold, according to the report. Open source clustering will make this expansion efficient and affordable.

2. Proprietary software companies go open source

Proprietary software providers feel immense pressure to support open source technology. Although Oracle has been known to bash open source, the company has at least 14 active open source initiatives and has become a corporate sponsor of the OpenStack Foundation, with plans to integrate OpenStack capabilities into its products and cloud services. In September 2013, IBM announced that it would invest $1 billion into Linux and open source innovation to help its customer run big data and cloud computing solutions on IBM's Power Systems servers. Just this January, Microsoft open sourced its Azure cloud server design through Facebook's Open Compute Project. The other tech giants are following suit.

Open source will make even further inroads in the public sector this year. The U.S. government has built OpenSource.gov, a back-end site dedicated to helping departments migrate to open source technology. The Pentagon's Defense Advanced Research Projects Agency (DARPA) alone is funding and contributing to over 70 open source projects related to data infrastructure, visualization, and analytics. The NSA even open sourced Apache Accumulo, its own NoSQL database. Thanks to organizations like the Open Source Digital Voting Foundation, U.S. elections may soon utilize digital voting on open source platforms, just like Estonia and now Norway. Mass open source adoption in government will place additional pressure on proprietary software companies to evolve.

3. High performance computing

Open source will produce the next generation of high performance computing clusters that underlie big data analysis and applications. The high speeds of such computing clusters provide the ability to parse gargantuan volumes of data in milliseconds. Enterprises in every single industry now need such capabilities to make use of the data they collect. Clustering can also help enterprise websites support wide fluctuations in visitors and usage without faltering in performance. We created this clustering technology way back in 2004 before data analytics became crucial to business strategies. Today, it will be the key to pinning down valuable data at a reasonable cost.

4. Mobile development

With the relative decline of desktops and rise of mobile devices, open source is becoming the foundation for superior mobile development. Existing platforms like Joomla, Convertigo, Dream Factory, OpenMEAP, and many others allow developers to build their product, get it to market quickly and scale. Open source programming languages, databases, middleware engines and other tools will continue to fuel mobile development and the growth of a mobile open source ecosystem.

5. Universities teaching open source

Universities have long been users and producers of open source technology, and students are getting in on the action. As VentureBeat reported back in November, twenty-two universities including Stanford, MIT, Berkley, and Carnegie Mellon have all teamed up with Facebook to provide students with academic credit for working on open source projects. Universities are adapting their curriculums and welcoming such opportunities to make sure that students graduate with in-demand skills and the best employment prospects. Beyond the Facebook inspired projects, every major tech university has faculty members working on open source projects (often with their students). As high schools, junior highs and middle schools pick up the slack in America’s computing education, I have no doubt open source will find its way into their curricula too.

The open source versus proprietary contest is ending because a critical mass of technologists in business and government recognize that open source technology offers higher performance, scalability, and reliability at a much lower price than proprietary solutions. Open source’s leadership in data storage clustering, high performance computing, analytics and mobile development will secure open source dominance over the next decade. The open source shift among proprietary software providers and the rise of open source in tech education will also reprogram the international tech workforce to study, master and implement open source solutions across every major industry. My hope is that we remember 2014 as the year we began to view open source as the norm rather than the exception in technological innovation.

How to get started in civic hacking

By Andrew Hyder (originally published February 2014)

What is civic hacking?

Seventy people gathered together one sunny Oakland afternoon to volunteer and improve their city. There were no rakes or yard tools normally seen at volunteer-day events though. No paint brushes, no trash bags, no canned soup bins. These seventy people were laden with laptops and were volunteering to improve the city's website.

This group of engaged citizens were building Oakland Answers, a new easy way to get answers for the most common questions asked on the Oakland City website. From finding out how to pay parking tickets, to checking what jobs the City is hiring for, the new website is citizen-focused and community built.

The day long event was called a "writeathon" and the majority of the folks in the room were not web developers but long time Oakland residents who came to write answers. Technologists were there too though, setting up servers and forking the open source code for the site. These web developers, answer writers, and City staff were all taking part in the growing new movement of civic hacking.

> "Civic hacking is people working together quickly and creatively to help improve government."—Jake Levitas

Here are a few more examples of popular open source civic hacking projects:

OneBusAway (on GitHub)
City budget visualizations (on GitHub)
311 Service Tracker (on GitHub)
Flu Shot Finder (on GitHub)
StreetMix (on GitHub)
CityVoice (on GitHub)

Open source civic hacking

Open source software is fundamental to civic hacking. Passionate volunteers write code and invent services that improve their own neighborhoods, but do so in a way that can be repeated in other communities around the world. Being able to easily share code without restriction is what allows for civic technology to scale.

For example, a few years ago in Boston there were severe snow storms that buried the fire hydrants. The same snow was downing powerlines and sparking fires. Some civic hackers saw this problem and created Adopt-a-hydrant (on GitHub), a way for neighbors to volunteer to shovel out the hydrants on their block. The following summer, the same code was forked and redeployed in Honolulu, not for snow but for tsunami sirens. Adopta has since been redeployed dozens of times and is being constantly improved by coders across the country.

Getting started

A great first project is to include your city in an existing service. Take Click that 'hood (on GitHub) for example. It's a fun game that helps teach about a city's neighborhoods. What's great about it is that it has clear instructions for adding your own city to the game. These instructions include using open source tools, collaborating on GitHub, and finding open data—all necessary skills for getting started in civic hacking.

Finding open data

The civic hacking movement is dependent on being able to easily find data about governments and the places they govern. If transit data, like bus schedules and train station locations, aren't available than we couldn't make any useful apps about transit. Luckily, many cities understand the importance of making their data available and have open data portals now. Data.gov has a list of many of the government data portals around the country and world. These portals gather all the available datasets that a city has and puts them all online in one place. The best data portals have that data in a machine readable format, so that it can be easily included in apps. Check your city's website to find if a data portal exists. If not, then working with your city to get one setup is a great civic hacking project to start with.

Open source data portals:

- CKAN
- DKAN
- Socrata

Community

It's important to remember that civic hacking includes both community and technology. All aspiring civic hackers need to join with others to solve our civic problems together. Check out the Code for America Brigade to find a local volunteer group or start your own. The best part of joining up with other civic hackers is finding out how they've achieved successes in their own cities. The Brigade is one of the best resources for discovering the latest open source tools and projects to work on. Also, working with city staff and civic leaders who are part of the Brigade is vital so that the civic technology apps created by civic hackers solve real societal problems. Many different skills and many different perspectives are needed to work on problems that effect many different people. Finally, to get really immersed in the civic hacking movement, consider applying for the Code for America Fellowship program.

Happy hacking!

Who helps your Linux distribution run smoothly? Thank a packager today

By Luis Ibanez (originally published February 2014)

The people behind the scenes who work tirelessly to make your Linux distribution run smoothly are the packagers. The vast majority of Linux packagers are volunteers who dedicate their evenings and weekends to create and maintain the gears of the Linux distributions they love.

The Linux ecosystem is thriving with many different distributions. They satisfy different audiences, needs, and styles. They range from types that are:

- secure and suitable for military and financial applications (RHEL)
- bleeding edge and dynamic (Fedora)
- universal (Debian)
- educational (Edubuntu)
- for the hobbyist (Raspbian)
- for the media artist (Ubuntu Studio)
- for the easy-going user (Mint)
- for the desktop application and cloud (Ubuntu)
- simple and lightweight (Arch)
- faithfully built from source code (Gentoo)
- reliable and unpretentious (Slackware)

And, the list keeps going for tens of other distributions that are continuously evolving.

In all the cases, the Linux distributions are aggregating free and open source packages, configuring them and combining them in a manner that ensure their proper collective behavior. Thanks to all that configuration and testing, adopters of Linux distributions have the peace of mind of simply installing a package (a given application) and having the confidence that it will work nicely with the other packages currently installed in the system.

The "magic elves" behind the scenes who work tirelessly to make this work smoothly are the *packagers*. The vast majority of Linux packagers are volunteers who dedicate their evenings and weekends to create and maintain the gears of the Linux distributions they love.

Back in February 2012, I had the luck of being the first trainee of the Debian Maintainer of the Month (MoM) program, started by Andreas Tille. The goal of the MoM program is to educate and train new packagers for the Debian distribution by pairing trainees with mentors and doing the training while packaging a new real application. The program has all the ingredients for balancing an interesting, challenging, and rewarding task.

During my training, we learned how to package fis-gtm, the open source implementation ofM/MUMPS that is of great importance for the ecosystem of open source Electronic Health Records systems. The fis-gtm package has just been accepted in the Debian unstabledistribution.

Going through the training was eye opening. I learned a great deal of very useful tools and practices, that since then I continue to use daily. From the proper use of GPG keys, to chroot containment and

remote screen sharing with tmux. The collaboration environment was well described by Andreas Tille as: "Waking up in the morning to realize that someone in another continent has already solved for you the problem that had you stuck last night."

In ideal circumstances, a Linux packager works closely with the developers of the upstream package in such a way that new releases of the package can be adapted quickly to be included in the next release of that Linux distribution. In many cases, the process of packaging uncovers issues with the package that require the upstream developers to make changes and adjustments. A packager also works in close coordination with other packagers in the same Linux distribution because many packages have dependencies on other packages or provide services for other packages, making it vital that the community of packagers coordinate their updates to ensure the consistency of the final Linux distribution.

As Linux users, it is often easy to forget (disregard?) how much work goes into the creation and maintenance of a Linux distribution.
Becoming a Linux packager is an excellent way to learn about software development, quality control, project management, and software maintenance in an environment of passionate individuals who deeply care about the output of their work. This is an experience all young developers should have.

After having learned the ropes of Linux packaging, and having seen first hand the dedication of this community, I developed a great deal of respect and appreciation for their work. Now, every time I install a package, whether it is from the command line with

```
sudo apt-get install package
```

or

```
yum install package
```

or any of the equivalent graphical interfaces, I pause and think:

"Thank you to the person who spent many hours configuring and building this application so that I didn’t have to."

With greater permissions, comes greater responsibility

By Heiko Rupp (originally published February 2014)

I came to work with open source after an experience in college. We used a system called Usenet, a world wide distributed discussion forum. At the university, there wasn't an email client I liked, so I wrote one and just gave it (including the source code) to whoever wanted it. This experience introduced me to a community of people who made things and shared them; it also introduced me to a job at my alma mater as a Usenet administrator.

In that position, I was an administrator for one of the (at that time) top 10 Usenet servers in the world. It was running open source software: InterNetNews (INN). Running this server taught me how to support bug reports and send patches to the maintainer. Then, I took over maintaining the FAQ document and did that for some years.

Later, I began working for a company running JBoss Application Server 3. We created some of the required artifacts using XDoclet. We had some pain points with both, so I wrote patches and enhancments for both. After some time, I got access to the source repository and I was then able to directly check on the fruits of my labor and I became more in tune with the inner workings of the project. Working with XDoclet, I incorporated the patches others submitted, answered questions, and submitted bug reports. Eventually, I got the rights to do releases, which felt very special.

The thing about open source is that a lot of it starts with having a pain point, like using software that was given to you to use that you don't like. If you are lucky and that software is open source, you can have a look at the source code. And, if the "pain point" is big enough, you can debug it and create a patch. Or, if you have some questions about an aspect of that software, those questions may get answered by someone on the development team, or you might figure it out for yourself. Then, when the next person comes along and asks that question, you know the answer and can help the project as a whole by answering it.

Answering questions and submitting patches

When you're getting started with open source projects, answering questions and submitting patches will earn you credibility within the project, and at some point it's like that the folks in the "inner circle" will ask you if you want to have the rights to directly commit your patches or to edit documentation pages. With greater permissions, comes greater responsibility... and greater possibilities to make an impact on the project.

If you can't code, remember that there are so many ways to get involved in an open source project *beyond coding*. Like, fixing typos in the documentation or translating it. Many projects also have a bugtracker. Use it to view an old bug report, then try it with the latest version of the software and report the result back in the tracker so that the development team can better judge when and how to fix it.

Here I shared 10 ways to get involved in an open source project without writing any code.

When you contribute, don't be disappointed if folks are too busy to look directly look at your

contribution. This isn't personal! Try to make it easy for them to consume your work by applying a small patch that explains what is broken and what can be done to fix it. Just saying "does not work" without further data, gets everyone nowhere. Similarly, if you create a change in the source code, make sure it applies nicely and does not break any tests. Strive to be a team player and it will earn you credibility which will result in faster feedback and greater accessibility to the open source project.

May open source be with you

By Cory Hilliard (originally published February 2014)

My introduction to open source software began when I was sitting on a server room floor, with my head in my hands, completely frustrated with a Windows 2000 server. Every night there were some services that would crash. Every morning I would get yelled at by my over-bearing boss. I was new to the company, it was my first IT job fresh out of Network Admin college, where I graduated at the top of my class, but I couldn't fix this problem because it was a "known Microsoft issue," and I just had to wait for the update.

I had known about Linux and was following its progress while attending college. I kept reading about its rock-solid operation, so I wanted to test it out in our company (risking my own neck and bacon). Somehow, I convinced my boss that with the purchase of a $200 used computer I could solve all his problems. My plan was to replace one of the services that was causing problems just to see if Linux could handle the job. I searched the Internet and found a lot of information about Red Hat and how brilliant their services were, but there was no way I would be able to convince my boss to buy a Red Hat server subscription. Heck, I don't even think he paid for his Microsoft licenses!

Eventually, I found CentOS. I used an online walk-through for setting up a Dynamic Host Configuration Protocol (DHCP) server on Linux, booted it up, and disabled the MS DHCP service. I was overjoyed to find that the DHCP service ran perfectly. It even gave me more power and control over what was going on. I was truly ecstatic.

After a couple weeks of flawless operation, I suggested we move another problem service over. Fast forward about eight months, and I had moved all services over to Linux and my job became so easy. I had no prior experience with Linux and didn't know what to do if it crashed, but I made thorough notes and could rebuild whatever it was in a matter of minutes. It never crashed. Never needed rebooting. Life was good.

Years after this experience, my Windows XP laptop was taking 30 minutes to boot every morning, includering jittery operation and frustration after frustration on my end. So, I made the switch to Linux. I was sold on Red Hat/CentOS/Fedora because of their community, contributions to open source, and the philosophy that the Fedora distribution has. Fedora believes in building a community distribution that promotes and fosters open source values and beliefs while remaining free for everyone. They are dedicated to freedom and innovation, and they contribute everything back to the free and open source world.

So, I selected Fedora as my full-time personal distro. And, I am now using Fedora with the Cinnamon GUI, because Gnome 3 feels a bit like a tablet interface to me. That's another beautiful thing about Linux, you have the freedom to choose the option that best suits your needs.

Open source Jedi training

Through all these experiences, I found that what I've loved is learning about open source. I love the freedom I feel when installing software on any machine. No limitations. And, it operated cleaner, was easier to personalize, and was free as in freedom. Yes, I'll admit it, I fell in love with open source. I

found that in all my command line hacking, editing scripts, and trying to read code that I couldn't yet read, I wanted more. I wanted to learn how to write open source software. I wanted to contribute to the greater good. I felt like I needed to become an evangelist of open source software and standards. It felt right. It felt pure. I was "on a mission from Gad." I promote Linux to anyone who will listen. I have successfully converted about 20 families to using Linux full time and offer them free support when they need it.

Currently, I am halfway finished with my three year Computer Engineering Technology/Computer Science course at Algonquin College in Ottawa Ontario Canada. It is a great school, and all IT students learn about Linux as part of the course curriculum. Some instructors promote Linux and open source software in their classrooms, while others demand their students run Windows and use proprietary software. I have been trying to promote and encourage the use of Linux and other open source software like LibreOffice for our classes and assignments; I personally try to use only open source software even at the expense of losing marks.

At this point in my open source journey, I have so many ideas for open source projects, and once I complete my training I want to contribute to other ongoing projects like Gnome and Cinnamon GUIs, Adobe Brackets, and Fedora. I feel like a kid in a candy store that is waiting for his allowance so I can spend it all!

I am treating my education as if it is open source Jedi training. I can't wait for my first epic battle to begin. May open source be with you.

Consuming open source software: How to use and buy it

By The Outercurve Foundation (originally published February 2014)

Vendors and original equipment manufacturers (OEM)—and their IT customers, governments, and academics—are all using, buying, and making open source software, and often all three activities at once. This is a good way to think about one's relationship with open source software projects. There are three activities one typically engages in with respect to the open source software project: make, use, buy.

Sometimes one is performing all three activities with respect to a single project, e.g. buying Red Hat Advanced Server for primary deployment in IT, while also using the Centos distribution for development and testing, and contributing (making) a few bug fixes back to the Centos project. At other times one has a different relationship with each project in the enterprise portfolio, e.g. buying JBoss support for the application server, while using Fedora Linux (from Red Hat), and using and contributing back to the Eclipse integrated development environment (IDE) project in development.

Using and buying liberally-licensed open source software is relatively straightforward. One is simply consuming the software. One buys a product that is based on open source licensed software in the same way that one buys other software. One evaluates the vendor's offering against one's own IT requirements and managed procurement risk profiles. Another way to say this is that you don't procure Red Hat Linux server software differently than you historically bought Solaris or might buy Microsoft Windows Server systems.

Using an open source-licensed software project (as opposed to buying a product based on it) adds additional considerations that should factor in the strength of its community and the costs of supporting that choice either through the development of in-house expertise or external specialists. One develops in-house expertise by participating in the open source project's community.

One reviews a project's documentation and tutorials, forum and email list activity, and IRC channels. One also should consider the availability of contracting support from around the community. These considerations are the same whether the open source software under consideration is tools and systems or developer libraries and frameworks. The investigations and trade-offs are no different whether one is an individual with an amount of time to invest to solve the problem or an IT department wanting to use open source software on a large scale.

Once one starts to make open source software, i.e. to produce it, there are different considerations. These are the two scenarios for producing open source:

- Contributing to an existing project, adding value through bug fixes and new functionality (and non-software contributions like documentation and translations).
- Starting a new open source project, which means organizing the infrastructure, developing the initial software, and providing for the early community.

The motivation for contributing to an open source project is simple. People generally start using open source software before they become contributors. They use it because it solves a problem. Once they use the software for a while they will encounter a bug, have a change they want to make, or possibly

document a new use case. Contributions can be given back if the user is comfortable making the software changes and the project community has done a good job of making it easy to contribute.

It sounds easy enough to simply make the necessary change and ignore the contribution back to the community, but living on a personal forked copy of the software comes at a cost. There are two main problems: other enhancements and bug fixes aren't seen and shared except by installing newer versions of the 'official' software, and the forked software will need to be re-integrated if upgrading. This can amount to significant work.

It is far better to contribute one's changes back to the project community if feasible, working with the project maintainers to ensure it is contributed correctly and patched into the main development tree. The onus is on the community to make it easy to contribute and on the on the contributor to do so correctly. The cost of living on a fork gets worse over time as the forked branch drifts further away from the project's developmental course. It's well worth the investment to embrace the efficiencies of the development model and contribute back.

The steps from using to contributing and participating in an existing project are small. A company that chooses to use an open source software project to solve its problems rather than purchasing a product has already invested in learning the project enough such that contribution is an easy step with automatic cost savings over living on a fork. The contribution step is easy—even if one is paying for support.

However, creating one's own open source software project requires more care and investment, and we explore that in the blog post: *Making Open Source Software: the considerations around a project's success*.

Originally posted in a series on the OuterCurve Foundation blog. Reposted using Creative Commons.

What's the best Linux desktop environment for me?

By Meine (originally published February 2014)

When you install a Linux distribution, a set of programs comes along with it. It's easy to add and delete elements of the programs that don't fit with your needs, but what about altering the look and feel of the distribution to suit you? The key is to add a second desktop environment or window manager.

This is an example of how Linux is all about freedom of the user, by the user.

The ability to change and alter the desktop environment is just as important as being able to change and alter the underlying technical components. A few years ago Linus Torvalds complained about the GNOME 3 desktop and reportedly switched to a different one. The Unity environment for Ubuntu caused a similar reaction by users. On the other side of the proprietary divide, Microsoft Windows 8 created turmoil among users with their tiled new look. Users wanted their Start menu back and to be able to get things done as they had before.

Running Linux, changing or adding a different desktop environment is as easy as installing any other program. Just install the software, log out, and log in back again on the new environment. Keep in mind that most desktop environments come bundled with specific text editors and terminal programs—you can decide what you to keep and what to dump. And, choosing a different window manager just changes the way you interact with your Linux box, it doesn't cause you to lose new programs.

Which desktop environment is the best?

First, look at the hardware of your computer. The desktop environment can be beautiful but eat up a lot of your resources. My top four recommendations based on a balance of beauty and resources are:

1. GNOME3
2. KDE
3. Cinnamon
4. Unity

On less powerful systems, a 10-year-old desktop or netbook for example, a more straightforward desktop environment that will do a good job without pushing the cooling fan all the time is a better fit, like: MATE, XFCE, LXDE, OpenBox, or Enlightenment.

Most Linux distributions offer a default desktop environment that fits best with its overall concept. I see it an orchestra of instruments all playing nicely together. Heavy environments also provide very complete software by default. Desktops like XFCE and LXDE come with lighter programs, but still will not let you down on productivity. Of course, you can add complete suites like LibreOffice to a light desktop; that way you have a fast starting computer and file manager, but also real power for writing, calculating, and creating presentations.

Why Linux?

Desktop environments for Linux also helps reduce costs and environmental footprints for individuals, small groups, and large businesses, because you can use the hardware you already own instead of

buying new and still update to the latest and greatest. The German city of Munich recently switched to Linux an officially promotes its use to reduce e-waste in the community. Also, support for Linux distros is typically good with regular updates for at least a year or two, while others keep up support for longer periods, like the LTS versions.

If changing to a new operating system sounds daunting, simply create a backup of your files before switching. Additionally, all of the well-established Linux distributions come with excellent instructions that guide you through the installation.

Tips for installing

There are over 300 Linux distros to choose from, just take a look at most popular ones! If you are unfamiliar with Linux, one of the top 10 distros is a good starting point, although Mint and Ubuntu will be a lot easier for a novice user than Arch. Mint also includes the necessary software to run videos and music. With Fedora this is more difficult because this distro only includes non-proprietary drivers. After a while you'll get used to Linux and you could try something different, like installing a different desktop environment or another distro. But, first stick to the basics as provided until you know more and feel comfortable using the system.

You can try Linux using a LiveCD of the distro of your choice. Download the .iso file and install it on a 4GB USB flash drive. UNetbootin is a program that allows you to create bootable live USB drives without burning a CD, and is available for Windows, Mac and Linux. Just try the Live version for a while before you install it on your computer. If it is not to your liking, just make a bootable Live USB with another distro. If you already use a virtual machine, you can try it from there.

Finally, Linuxers are friendly people and are able to offer help when you get stuck. The DistroWatch pages offer links to the distro specific user forums. Remember, you are not the first one trying, so a lot of questions and answers are already given.

How to teach hacking in school and open up education

By Pete Herzog (originally published February 2014)

Whatever you may have heard about hackers, the truth is they do something really, really well: discover. Hackers are motivated, resourceful, and creative. They get deeply into how things work, to the point that they know how to take control of them and change them into something else. This lets them re-think even big ideas because they can really dig to the bottom of how things function.

Furthermore, they aren't afraid to make the same mistake twice just out of a kind of scientific curiosity, to see if that mistake always has the same results. That's why hackers don't see failure as a mistake or a waste of time because every failure means something and something new to be learned. And these are all traits any society needs in order to make progress. Which is why we need to get it into schools.

Now, there is the expected resistance from school administrations and parents. Mostly because people don't know what hacking really is. Many people who have been called hackers, especially by the media, or who have gotten in trouble for "hacking" were not, in fact, hackers. Most all of them were just thieves and fraudsters. When you read in the news, *Teen girl hacks Facebook to harass a classmate*, what you're seeing is a sensationalized headline. What a hacker reads in that headline is: *Mean girl watched classmate type in her Facebook password and then logged in as her*. That mean people and criminals do bad things with communications medium is not a reason to fear the medium. Schools are there to educate and can embrace this distinction for real change.

Hacking is a type of methodology. It's a way to do research. Have you ever tried something again and again in different ways to get it to do what you wanted? Have you ever opened up a machine or a device to see how it works, read up on what the components are, and then make adjustments to see what now worked differently? That's hacking. You are hacking whenever you deeply examine how something really works in order to manipulate it, often creatively, into doing what you want.

A hacker is a type of hands-on, experimenting scientist, although perhaps sometimes the term "mad scientist" fits better, because unlike professional scientists they dive right in, following a feeling rather than a formal hypothesis. That's not necessarily a bad thing. Many interesting things have been designed or invented by people who didn't follow standard conventions of what was known or believed to be true at the time.

For example...

• The mathematician, Georg Cantor, proposed new ideas about infinity and set theory that caused outrage amongst many fellow mathematicians to the point that one called his ideas a "grave disease" infecting mathematics.
• Nikola Tesla is another person considered a "mad scientist" in his day, but he knew more about how electricity behaved than anyone else. He designed possibly the first brushless motor that ran on AC electricity but is mostly known for the Tesla effect and the Tesla coil.
• Then there was Ignaz Philipp Semmelweis who figured out that doctors need to wash their hands between treating patients to keep diseases from spreading. He wondered if the diseases following him around between patients were his fault, so he decided to try washing hands between his patient visits and sure enough the transmissions disappeared. His ideas went against both the scientific conventions

of what was known at the time about germs (nothing) as well as the convenience of the doctors who felt it was too much hassle to keep washing their hands.

It just so happens that the way the Internet is designed and the huge number of different applications, systems, devices, and processes it has makes it the most common place to find hackers. You could say it's a place where information can run free because it was built open and free by hackers so it's the best playground for hackers. But it's not the only place. You can find great hackers in almost every field and industry and they all have one thing in common: they spend time learning how things work so they can make them work in a new way. These hackers didn't look at something as the original designers did, but instead saw bigger or better potential for it and hacked it to be something new.

What you may think you know about hackers is that they can break into other computers and take over other people's accounts. They can read your email without you knowing. They can look through your web cam without your permission and can see you and hear you in the supposed privacy of your own home. That's not untrue.

Some hackers see network security as just another challenge, so they tinker with ways to trick or fool the system, but really what they're trying to do is out-think the network installers or designers. They discover as much about the network as they can, where it gets its instructions, the rules it uses, and how it interacts with operating systems, the other systems around it, the users who have access to it and the administrators who manage it. Then they use that to try different ways of getting what they want. This kind of hacking can be greatly beneficial to the world for understanding how to be safer and for building even better technology.

Unfortunately though, sometimes the hacking is done by criminals and what they want is illegal, invasive, and destructive. And those are usually the only hackers you read about in the news. A hacker is not someone who posts to someone's account when they leave a social media page open or shoulder-surfs passwords and then logs into their account later. That's not hacking. A hacker also is not someone who downloads a script kiddie tool to break into someone’s email. Those aren't hackers; those are just thieves and vandals.

Hacking itself is not illegal. At least not any more than throwing a rock is illegal. It all comes down to intent. If you throw a rock and your intent is to injure someone, that's a crime. If your intent is not to hurt someone, but someone does get hurt, that may not be a crime, but you are responsible for your actions and will have to pay restitution. An Institute for Security and Open Methodologies (ISECOM) project called the Hacker Profiling Project found that the most damage from hacking comes from young, inexperienced hackers damaging other people's property by accident. Which is something parents and teachers already teach kids when it comes to rock-throwing, but it doesn't translate well when it comes to how to behave in cyberspace. If we are teaching hacking, then we can also teach responsibility, accountability, and make it clear how to behave when hacking around other people's property. This will encourage students to stick to hacking the things they bought and own.

The caveat to that is that there are cases where it may be illegal to hack something you bought and own. There are hackers who have been punished for hacking their own devices and computers. These things were closed to prevent them from being copied or changed despite that they paid for it and own it. These are hackers who hacked programs, music, and movies they bought so it looked, behaved, and sounded the way they wanted to or played on other devices they bought and owned and were prosecuted for it. Especially when they openly shared their ideas with others. Hackers will find that any closed source software they buy may be illegal to hack, even if it's just to check for themselves that it's secure enough to run on their own computer. This is because many of the things that you purchase may

come with Copyright and a contract as an End User License Agreement (EULA) that says you can't. And you agree to it when you open or install the product, even if you can't read it or find out about it after you've opened or installed the product. Yes, that's sneaky and unfair.

But that's all the more reason to teach young people to hack. You see, education is open. It can be legally hacked to teach kids to think openly, be inspired, be curious, and thus, to be a hacker. What hacking is really about is taking control of something if you don't like how it works. Why would you do this? To have the freedom to make something you own do what you want. And to keep others from changing something you own back to the original form or copying all your ideas, drawings, writings, and pictures to a cloud somewhere to be controlled by someone else who claims it's for your "best interest."

As a hacker, you know what your own best interest is. Sometimes you buy something and the company you bought it from will attempt to forcefully or slyly make sure you can't customize it or change it beyond their rules. You can't play it somewhere else or use it any other way than as intended, supposedly to protect you. And that might be okay to agree to as long as you accept the fact that if you break it then you can't expect them to fix it or replace it. That would mean that hacking something you own does more than make it yours, it makes it irrevocably and undeniably yours. As scary as that may sound to some, it certainly has its advantages. Especially if you want to keep others, like the company that made it and the marketing company they're re-selling your information and habits to, out of your stuff.

And finally, of course knowing how to hack makes you more secure. For many, many people, security is about putting a product in place, whether that's a lock or an alarm or a firewall or anything that theoretically keeps them secure. But sometimes those products don't work as well they should, or they come with their own problems that just increase your "Attack Surface," when a security product should be shrinking it. (The Attack Surface is all the ways, all the interactions, that allow for something or someone to be attacked.)

And yeah, good luck getting that product improved in a mass-marketing, pay-as-you-go, copyrighted, closed-source, "you bought it as-is and that's what you have to live with" kind of world. That's why it's so important to know how to hack your security. A hacker wouldn't buy the same padlock you would because a hacker sees locks in terms of how many seconds they would need to open it. Hackers learn to analyze a product and figure out where it fails and how to change it so it works better. Then they might have to hack it some more to keep that company they bought it from, from changing it back to the default!

So hacking in terms of breaking security is just one area that hacking is useful, because without being able to do that, you may have to give up some freedom or some privacy that you don't want to give up. (And some of you may not care right now about certain things you do or say or post, but the Internet has a long memory and it's getting better and better at helping others recall those memories of you. What goes on the net stays on the net. And kids today are pretty much born on the net.) Not to mention technology is getting more and more out of our ability to control it. That mobile phone of yours or that new flatscreen with built-in camera for Skype are likely doing things that you don't know and don't control with what they see and hear. It takes some hacking to wrestle that control back.

Schools and educators who read this and want to teach their students to hack, and what hacking can be, need to be aware upfront that it won't be easy. There will be resistance from closed minds. School administrations may also need to contend with the fact that hacking some things may be illegal in their state, and they will need to get open source hardware and software to try to stay on the legal side of

things. When teaching students how to hack and what hacking is, it can be hard to do with words. Try experiences and putting it into practice to really get your point across.

Free, open projects like Hacker Highschool can help kids develop the skills, feeling, and intuition through practice with support so they don't break the wrong things. The possibility of breaking something is simply part of the process, and should not be a factor keeping teachers and schools from teaching hacking. They should provide that support with an open source and open minded effort.

Get more contributors to your project with better documentation

By Amye Scavarda (originally published February 2014)

It is not uncommon to have a cycle of news around communities being unfriendly to women or newcomers or people who aren't already *there*. By 'news' I mean someone posts something that is close to their heart about some unjustice and other people comment on it or write their own posts and generally, the moral of the story is: *But we should be better than this!*

This is normal and desired behavior as part of the overall community. This cycle is a good thing because it causes people to think about their behavior as community members and what it's like to be an outsider and how they can improve. These are all positive steps because it springs from an honest desire to be better people. That's awesome.

Lately, I've been poking in areas of technology where I don't tend to tread often. One area has been picking up a new piece of technology by looking at the documentation. This particular journey began by being a Glass Explorer, and that's a story that I've talked about on my own blog at amye.org, so I won't recap it here. But, late one night, as I was churning through documentation on what Android will actually do and all of these (frankly amazing) things that your smartphone hardware has the capability of doing (ambient air pressure, anyone?), it struck me that documentation is one of the great levelers of our time.

Everyone has to read documentation. Everyone. You know those times where you jump in and don't bother to read the manual and then wonder why it didn't work? Well, yes... this happens to everyone. The person 'just downloading to try it out' and the person who wrote vast swathes of the code both have a vested interest in documentation. Granted, there are different spectrums to their interest, but they are both valid. I think of it a little bit like the 3rd grade student who's just learning to read music and the professional pianist who can sightread instantly—but, they both need the score to know what the next notes to play are.

Given that context, your documentation in your open source community says something about what you expect your community to be. I realize that I am going to get plenty of comments about how your code is self-documenting. No, dear sirs and madams, it very much isn't. Expecting your code to be self-documenting is like marking individual trees in your forest without paying attention to where the forest lives. Is this in the arctic? The tropics? What sort of creatures am I likely to find under that rock when I pull it up? What am I looking at and where do I expect to be next?

I'm enjoying working through the Android documentation because it's a platform and it lets you know at nearly every turn. It's clearly defined and laid out, but there are still pieces of the vocabulary that are unfamiliar.

I ran across this tweet by @relsqui a while back and it has stuck with me...

> “Theory: Lack of documentation promotes impostor syndrome. 'I must be supposed to know this already/be able to figure it out on my own ...'”

Here's why I've held onto it—it might be a theory but a theory is the highest scientific law, and it's up to us to disprove it. There's plenty that you can do; just keeping a running text document as you get through a project is helpful as well.

Bottom line: If you're not going to document what you did for other people, how about doing it for your future self who will come back in six weeks and not remember where you were going for?

Open source tools to build your best business

By Nabeel Hussain (originally published February 2014)

In January 2013, I started exploring open source solutions to help implement my business idea. I used Wordpress, Joomla, and OpenShift to create FilmBoxFestival, a platform for streaming documentary films. *Note: It is still in the testing phase.*

I created, validated, and gained traction for my busines idea due to the speed which these open source tools offered. So, if you're an entrepreneur, I encourage you to explore open source possibilities. Here are some of the things I learned.

Identify feature sets for functionality

Open source solutions are modular in nature. Often, by breaking down the final product into smaller features it becomes easier to find modules and plugins with the functionality you want. For example, one of the features I wanted to implement for FilmBoxFestival was greater interactivity between readers. Instead of looking for one plugin to do this, I broke it down into a separate set of features, such as sharing and discussion forums, then I was able to find individual plugins for sharing and discussions because open source broadened the choices of solutions available and provided off-the-shelf options to create the solution I wanted.

Look for new tools

Always be on the lookout for new plugins, tools, and approaches to implementing the features you want for your business. The open source community is very active, and new solutions are always being developed. For example, with FilmBoxFestival I wanted an integrated payment system which showcased preview videos and allowed buyers to watch the documentaries they have purchased. I searched for over 3 months for a payment solution with these features before I came across Gumroad, a service designed for streaming content that also emails a secure and accessible link to the buyer.

Document, backup, and test at every step

Because open source solutions are so flexible and modular, it is possible for the functionality to break when modules are installed that have similar or complimentary functionalities. It is imperative that you discipline yourself into documenting, backing up, and testing your web solutions as you go. On at least two occasions I had to rebuild FilmBoxFestival from scratch. On both occasions I was able to recreate the website in under a day because of my systematic approach to documentation, recovery, and testing. Similarly, some plugins I installed worked on one browser but did not work on another. It was only by testing that I found the fault and was able to contact the plugin creator; then we began working on a solution.

While I was building and developing my business, the open source community provided excellent technical support and documentation. As an entrepreneur, open source solutions present a unique opportunity to develop your business in a cost-effective, rapid, and efficient manner.

Things newcomers to open source rarely ask but often wonder

By Shauna Gordon-McKeon (originally published February 2014)

Open Source Comes to Campus is an event series run by OpenHatch that introduces college students to open source tools, projects, and culture. Whenever we get a new-to-us question at an event, we write it down and answer it more fully on our blog. Here's a collection of "Infrequently Asked Questions" that are especially relevant for newcomers to open source projects.

Q: I'm worried that I'll be a burden on a project because I'm so new. What kind of effort does a project have to make to build an open source community?

A: First of all, you should know that expert open source community builders like Linus Torvalds understand this tension between training new contributors and the time it takes from existing maintainers to mentor them. I'll quote two paragraphs from a 2004 post Linus made on the Linux kernel list:

> On Tue, 21 Dec 2004, Jesper Juhl wrote:
>
> *Should I just stop attemting to make these trivial cleanups/fixes/whatever patches? are they more noice than gain? am I being a pain to more skilled people on lkml or can you all live with my, sometimes quite ignorant, patches? I do try to learn from the feedback I get, and I like to think that my patches are gradually getting a bit better, but if I'm more of a bother than a help I might as well stop.*
>
> Linus replied:
>
> *To me, the biggest thing with small patches is not necessarily the patch itself. I think that much more important than the patch is the fact that people get used to the notion that they can change the kernel—not just on an intellectual level ("I understand that the GPL means that I have theright to change my kernel"), but on a more practical level ("Hey, I did that small change").*

Another thing to keep in mind that is, for most projects, the project leader is the only contributor (source). Consider this: The act of *bugging* them with questions as a newcomer can make them more enthusiastic about the project as a whole!

If it were trivial to build an easily accessible open source community, virtually all projects would do it. There is some effort involved. Of course, a lot of the work that makes a project welcoming to newcomers—good documentation and setup guides, a well-maintained issue tracker, active development, standards of conduct in the community—makes the project better for everybody. But it

does take time and energy, which many communities of projects aren't willing to expend. That's why OpenHatch exists! We know that some projects put more effort into welcoming contributors, and we want to help you find those projects.

The kinds of projects that welcome newcomers don't see you as a burden. They see you as a huge help—even when you're struggling to understand bugs in the issue tracker or get the development environment set up. When you ask a member of an open source project for help, that gives them important information about what part of their project is confusing or incorrect. The questions you ask and guidance you need lets them know how help others later on. And of course, once you've gotten familiar with the project, you'll be able to help even more. The right kinds of project see that potential in you and want to work with you to get there.

Two rules of thumb for people who worry about being too much of a bother (which includes us, sometimes!): First, if people on the mailing list or the bug tracker haven't told you to stop talking, then you're probably okay, even if they haven't replied to you yet. Second, if you want another opinion, just join the #openhatch IRC channel and ask us.

Q: How do open source projects review changes, and how do those changes differ from a process like that of Wikipedia?

A: One key cultural element of Wikipedia, at least so far in its history, is that anyone's edits to English Wikipedia immediately become part of the *live version* of the encyclopedia. By contrast, to maintain quality, open source projects typically permit only a handful of people to *directly merge* changes into the main project. Therefore, submitted changes go through review.

Different communities have different standards and processes. Sometimes, there is no review at all, just automatic merging by the maintainer. In other cases, like Linux, submissions go to a mailing list for review. Others, like the OpenHatch web app, use web-based tools like GitHub pull requests. A great reference for a more complicated process is the OpenStack Gerrit Workflow documentation page.

Q: How do you make time to contribute to open source when you're a busy student?

A: By simply using open source programs, you are uniquely positioned to help other users. Don't forget that helping people use the software is a substantial contribution to the community! By becoming an expert in whichever programs you use, you will also gain the knowledge to help convert bugs filed by others into actionable reports for the main developers. And once you reach that point, you might find it easier to just fix the issue!

That said, it can be daunting to add another activity to schedules already stuffed full of classes, paid work, and student life. One way to approach open source is to see it not as an alternative to doing schoolwork but as part of your education. For computer science majors, open source projects are a great way to practice the concepts you're being taught in class. In the others sciences, learning how to use and contribute to open source tools such as R or Octave (or ImageJ or PsychoPy or JMARS) can help you excel when doing laboratory work—and it looks great on your resume. Even in the arts and humanities, learning about open source tools such as Processing can help you succeed.

For students who work while they're at school, open source projects can be potential employers. You can apply for Google Summer of Code or the GNOME Outreach Program and do a paid internship at an open source project. Individual companies working with open source may hire students that show enough interest and ability. You can even employ yourself using open source—for instance, you could do freelance work making websites and applications by using open source tools like WordPress and Drupal.

You can make open source a part of your social life as well. Join (or start) a Students For Free Culture chapter on your campus or invite friends over for a 'bug-fixing party'.

Finally, remember that your contributions to open source can be as big or as small as you want them to be. If you only have time to spend a few hours a month writing documentation or fixing small bugs in a project, that's great—the skills you're learning and connections you're making will serve you well if you ever decide to get more involved.

Q: What is the difference between source downloads for developers and downloads for users? What's a stable release?

A: Open source projects often release the program in a few different ways. Typically, if you are trying to contribute, you will want the latest source code cloned or 'checked out' from the project's version control system. If you are just trying to use the program, you typically want a user-oriented download.

Many programs' source code can't be executed directly on a computer—it must be 'compiled', which turns it from human-readable text into machine-executable binary. The most common languages where this is true are Java, C, and C++. Therefore, the most useful download for a person who wants to run a project like OpenOffice, which is mostly written in C++, would be a compiled version specific to their operating system (for example, a Windows-specific build). By contrast, if you want to contribute to the program, you should act similarly to how the main developer acts—that is, by using a version control system on the editable program text, the source code.

Another important concept is the stable release. During development, people make contributions that break the program, conflict with other changes, or are simply unfinished. At any given point the most recent version might be completely unusable. So, maintainers periodically work towards releases (noun), which are tested to make sure they're usable. If they are, maintainers release (verb) it.

So if you're a user, you'll want the stable release most appropriate to your operating system and computer. If you want to contribute, you'll want the latest, non-compiled version.

Q: Are all open source projects welcoming to newcomers? How can you tell when the communities around a project are filled with jerks?

A: Not all open source projects are welcoming to newcomers. Not all open source projects need to be. Some projects benefit from keeping their communities small and experienced. And, some maintainers simply don't have the energy or the inclination to mentor new people.

There’s nothing wrong with that, but it’s a shame when newcomers try to get involved with a project that isn’t interested in involving them. It’s a frustrating experience for everyone and unnecessary when there are plenty of people who’d be excited to work with them.

So how can you find a good project for you, a newcomer, to contribute to? Here are some good signs to look for:

• A large, active community is more likely to have members who can take the time to mentor. It also means that contributions will be acted on more quickly.
• Good documentation and demos mean that developers have thought about how to introduce people to their project.
• Some projects have Codes of Conduct/Diversity Statements, which demonstrate that the community is trying to be a safe and welcoming space.
• Projects that are part of Google Summer of Code or the GNOME Outreach Program for Women have made a commitment to being good environments for newcomers.

At OpenHatch, we try to identify projects that are especially good for newcomers. We’re also developing a list of OpenHatch-affiliated projects that are working with us to make contributing to their projects as easy as possible. (If you’d like to be on that list, let us know!) Feel free to contact us and ask us for recommendations or if a particular project is known to be good for newcomers.

Of course, there’s a difference between not having the time and energy to help newcomers and being actively mean or even abusive about it. If you experience the latter and our comfortable telling us about it, we’ll do our best to support you. OpenHatch people are always willing to provide advice on projects to join, either to help you find a great experience or to make the most of a bad experience. Find us on on IRC (#openhatch on irc.freenode.net) or email us athello@openhatch.org. You can also join groups such as Systers or the Empowermentors Collective, which may help you navigate the free software community.

More resources

This article has been a collection of information gathered during Open Source Comes to Campus events:

• Infrequently Asked Questions: Wellesley College
• Infrequently Asked Questions: UMass Amherst
• Infrequently Asked Questions: Bloomington

Read more posts about this event series on our blog.

For blog posts about getting involved with OpenHatch, read about the experiences of two of our contributors: Britta and Mandar.

A beginners guide to understanding OpenStack

By Jason Hibbets (originally published February 2014)

The more I learn about OpenStack, the more I see why there is so much buzz about the technology as well as about the community of developers and users. In a poll hosted on Opensource.com, we discovered that many of our readers are curious and eager to learn more about OpenStack. For those new to this technology, OpenStack can be described as a set of software tools for building and managing cloud computing platforms for public and private clouds.

Opensource.com has started gathering resources to help our readers learn what OpenStack is and what it can do:

What is OpenStack?

For an overview, see our What is OpenStack? page or visit openstack.org.

OpenStack contributions and community

See Rich Bowen's article: How to contribute to OpenStack.

OpenStack technology and more

For a deeper dive into the technology (like, the components that make OpenStack scalable) and more on the OpenStack Foundation, begin your discovery with these videos that we recommend.

Recommended videos

Introduction to OpenStack with Sandy Walsh from Rackspace

This is a 30 minute video that is one the best videos I've come across that can help introduce OpenStack to someone completely new to the project. The talk is not too technical and gives a broad overview of the project and the architecture. Walsh talks about what OpenStack is, what it is not, how the project got started, some of the core projects, and he outlines the architecture and scalability of OpenStack.

Overview of OpenStack and the OpenStack Foundation with Mark Collier

This is a 35 minute video with Mark Collier, COO of the OpenStack Foundation, where he gives an overview of OpenStack and goes into details about the community and the OpenStack Foundation.

OpenStack 101 with Joshua McKenty from Piston Cloud Computing

This is a 40 minute video that covers the history of OpenStack, an overview of the project (including debunks), provides a technical overview of each of the component, and covers the project governance. McKenty was part of the original OpenStack release and was the architect and team lead at NASA that

build the Nova volume, network, and compute components prior to OpenStack.

OpenStack Basics with Brian Gracely from Cisco

This is a 10 minute video that is very technical. It compares OpenStack to the existing cloud infrastructure and maps out what many technologist are familiar with to the OpenStack architecture.

OpenStack Architecture with Russell Bryant from Red Hat

This is a 45 minute video that is a technical deep dive into the OpenStack architecture. Bryant details each of the core projects:

Identity (Keystone)

Dashboard (Horizon)

Orchestration (Heat)

Metering (Celiometer)

Object stroage (Swift)

Image service (Glance)

Networking (Quantum now Neutron)

Compute (Nova)

About This Series

The Open Voices eBook series highlights ways open source tools and open source values can change the world. Read more at http://opensource.com/resources/ebooks.