Lab 1, Part 2: Introduction to Free and Open Source Software

Adapted from Greg DeKoenigsberg article
http://teachingopensource.org/index.php?title=Introduction_to_Free_and_Open_Source_Software&oldid=3610

Introduction

Free and Open Source Software, or *FOSS* for short, is software for which the *source code* can be freely shared, with anyone, for any purpose.

There are more rigorous definitions, and there are many licenses that help to ensure these freedoms in subtly different ways. We examine these details later in the book. For now, focus on this simple idea: the freedom to share source code is the essential element of free and open source software.

Upon successful completion of this assignment, you should be able to:

- Describe some of the benefits of using FOSS;
- Describe some of the benefits of participating in FOSS projects;
- List some FOSS projects that might interest you and explain why;

Why does/should FOSS matter to me?

Free and Open Source Software matters because it is real, widely-used, and allows anyone to see the source code. Having contributed to or somehow worked with FOSS is a very useful experience to have when applying for software development jobs—it shows that one has experience in real-world software projects (which are often large complex).

Without FOSS, getting experience in real software projects requires access, and probably permission, to see the source code. For students, that access is usually limited to those who can get *internships* or positions in *co-op* programs. Not everyone has the opportunity to spend the summer interning with a company that does large-scale software development, meaning that a vanishingly small number of students have the opportunity to work with large codebases. And even if they do, those students typically cannot show their work to anyone outside of the sponsoring company.

In the world of FOSS, the source code is available to anyone who wants to see it. Not only is the source code available -- also available are all of the interesting challenges that go with managing large software projects.

Source Control

How do fifteen software engineers work on the same piece of software together? When two software engineers decide independently to edit the same line of code, what happens? In the world of FOSS, we make use of version control systems to help avoid these kinds of problems. Without version control, it is a disaster.

Build Systems

Complex software is built in different *modules*. Those modules have different names in every language -- *packages*, *libraries*, etc. -- but modularity is always introduced to help manage the complexity of large projects. One software project may *import* dozens, or even hundreds, of previously built software modules. What happens when one of those modules changes? If you don't know how those modules fit together and don’t have a systematic way for managing all these modules, it is a disaster.

Documentation

There is also a lot more to creating good software than writing good code. How do you make sure people have the resources and knowledge they need to find and run (and contribute to) the software that you make? Beautiful code that doesn't get used is just as useful as code that was never written.

Tracking Bugs

It is difficult to have software systems with no bugs. Even the very best code can still have bugs, which means the ability to find and eliminate bugs is a critical skill for software engineers. If you don't know how to find the bugs your users find, you're in trouble.

Exercise 1 - Finding a Cool Project

Imagine that you have just been hired as a programmer for FOSS Inc., and your manager has told you that you must spend 20% of your time to work on a FOSS project that matters to you.

1. First, search the web and **find sites that *host* open source projects**. Use keywords "open source projects". There are many. List the names of two sites that host such projects.
   https://beanstalkapp.com/
   https://github.com/
2. Second, **browse through several of these sites** and find several projects that are interesting to you. You might be interested in projects that benefit others. You might be interested in tools that support work that you do. Or, it might be that you might find something that strikes your fancy that you never considered before! Take this exercise as an opportunity to explore broadly. Name two projects you found interesting and discuss why you found them interesting.
   One of my favorite projects that I've found on GitHub in particular is the VALORANT-rank-yoinker. Valorant is a 5v5 shooter game that I enjoy playing, and you used to be able to see other people's ranks in-game and you can't anymore. This project allows you to see everyone's ranks, as well as kill/death ratio's and headshot percentage with a pop-up. Another project I found interesting is the Osiris skin changer for CS:GO. CS:GO has skins for its different guns and some of them are really expensive, and the program allows you to get those skins (client sided only) for yourself in your inventory.

Source Code: To Share, or Not To Share?

Obviously, not all software is FOSS.

Most software developers do not share their source code -- especially companies that produce software with the intention of selling it to their customers. Microsoft, for example, does not share the source code for the Windows operating system and many of their other software products (e.g., MS Office).

Even freeware -- programs that are downloadable for free from the internet -- may not share their source code with the world. You can get the program for free, but if it breaks, or if you think of a way to make it better, there is no good way to fix it. For example, you can get the Flash Player from Adobe for free, but if you find a bug that crashes Firefox, you are stuck with that bug until Adobe fixes it.

There are definite advantages to keeping source code hidden, especially if your goal is to sell the software itself. It is harder to sell a software program when anyone is free to take the source code and use it for any purpose. If Microsoft were to release the Windows source code under an open source software license, anyone would then be able to take that source code, build "Bob's Own Operating System," maybe make a few changes, and then re-sell that product as a competitor to Microsoft Windows. Obviously, most companies who are selling commercial software do not want that to happen.

The value of sharing

That is not to say that programmers who write open source software never make money. Some of the most successful companies in the world use open source software to power their businesses. Google, Amazon, Wall Street, the US Department of Defense -- some of the world's largest and most innovative companies, government agencies, and industries are writing software using FOSS every day. They do not sell that code; they share it, and by sharing, create more value for their organizations.

The amazing thing about contributing to FOSS software is that you do not have to work for a large company to see the benefits of working in a free and open manner. As a developer, you might write a utility that solves a particular problem. By sharing it, others might discover the utility of your tool. Others might extend it and help see it grow. At this point, what started as a hack has become something valuable for many. At this point, we begin to see how FOSS development practices can provide demonstrable advantages over proprietary software development practices. Among them:

- **Shared development cost.** Writing software can be expensive, at least in terms of time. Good software takes time to develop, and time is money. And if writing software is expensive, then maintaining it is even more expensive. In the FOSS model, the cost of the writing and maintaining the software can be spread out over several individuals and/or companies.
- **Users can fix their own bugs.** This is not a freedom that is obviously useful to everybody. Not every software user is knowledgeable enough to fix a bug when they find it. That's fine; FOSS also means that users can find other people to fix their bugs for them. Not everybody who owns a car is able or

willing to change their own oil, but the freedom to change your oil, or fix a flat tire, or rebuild your own brakes -- or the freedom to be able to go to any mechanic or any mechanically inclined friend and ask them to do it for you -- is a crucial freedom to car owners. FOSS extends that freedom to software.

- **(Arguably) better software.** Allowing users to fix bugs can often lead to better software. One of the problems with proprietary software is that there is a limit to the number of people you can pay to fix code -- that limit is usually directly proportional to how many software licenses the company can sell. FOSS projects have the potential to build a huge base of participants, far greater than the number of developers that any one company could pay. The Apache HTTP server project is a great example of a FOSS project with many developers, both commercial and independent -- that has created demonstrably more popular and arguably better software than any of its proprietary counterparts.
- **Software that outlives its creator.** There are literally thousands and thousands of pieces of software, written for outdated computers that are no longer useful for any purpose. If we had source code for these pieces of software, we might be able to extend them to new computers, making them continually more useful and more interesting -- but because we do not have the source code for these programs, we have very little use for them anymore. There is a word for this kind of software: *abandonware*. In FOSS, there is no such thing as abandonware. Sure, people may stop working on a piece of software, but the source is always there, ready to be picked up and carried forward by anyone who has the time and interest to do so. Every dead FOSS project has a chance to be reborn.
- **The freedom to fork.** Sometimes software projects go wrong. If a project is proprietary, no one has any recourse if they do not like the direction of the project: the owner of the project decides the direction of the project, period. But because FOSS guarantees everybody the right to redistribute and modify the source code, developers can always take a FOSS project and move it in a new direction, without anybody's permission. This process is called *forking*. Forks are usually regarded as last resorts, since contentious forks can divide scarce developer resources and confuse users. However, a number of FOSS projects have benefited greatly from forks; the X.org server and Inkscape are notable successful forks.

Exercise 2 - List of Software

Look at the programs installed on your laptop and write up a list of all the software that you use most frequently (aim for 8-10). Do some research and find out whether each piece of software is FOSS. For those that are not FOSS, see if you can find any FOSS equivalents.

Here are the 8 most used apps on my computer.

1. Chrome
2. Spotify
3. VSCode/GitHub Desktop
4. Notion
5. Word
6. OneDrive
7. Messages
8. WeChat

Out of these programs, GitHub Desktop is the only FOSS software.

Submission:

Submit your answers for exercises 1 and 2 in a .doc, .docx, or .pdf file.