O'REILLY ONLamp.com LAMP: THE OPEN SOURCE WEB PLATFORM

What Is Free Software

by Karl Fogel, author of *Producing Open Source Software: How to Run a Successful Free Software Project*

09/29/2005

In This Article:

Free software is software that may be modified and redistributed freely by anyone, with no significant restrictions on how the code may be changed, the uses to which it may be put, or the parties with whom it may be shared.

From this simple definition flow many unexpected consequences. Today, free software is a large body of high-quality code on which much of the internet depends for critical functions, and it constitutes the core operating system for an increasing number of desktop machines as well. But free software is much more than just a collection of programs. It is also a political movement, a programming methodology, and a business model--although not necessarily to the same people at the same time. Indeed, even the term *free software* is controversial; as we'll see later, some people prefer to call it *open source software*. The story of how free software became so technologically successful, even as it became ideologically fractious, starts in the early days of the computer industry.

From Free to Proprietary

In the beginning, most software was free by default--free not only in the sense of "zero cost," but also in the sense of "freedom." The early computer industry was organized mainly around selling hardware, with each company offering its own unique design, incompatible with everyone else's. The customers, mostly engineers and scientists, were encouraged to improve the manufacturer-supplied software, and even to share their improvements with each other. Because hardware was not yet standardized, and since software portability tools such as compilers and interpreters were not yet commonplace, there was little risk of such improvements being useful on a competitor's machine anyway.

But as the industry developed, it slowly standardized on a few basic hardware designs, with multiple manufacturers for each design. At the same time, advances in compiler and interpreter technology made software portable in source code form. (*Source code* is the set of human-readable instructions that define how a program behaves; to study or modify a program, you need its source code.) With these developments, it became normal to write a single program and expect it to run on different kinds of machines. This had deep implications for the manufacturers: it meant that a customer could now undertake a major software engineering effort without being locked to a particular brand of computer. Furthermore, as computer architectures became standardized, raw performance differences between them got smaller and smaller. Manufacturers realized they would need to distinguish themselves on something other than just the quality of their hardware, and treating software as a sales asset began to make more and more sense.

Thus the era of easy and informal code sharing slowly faded away, and software became a source of proprietary value. People still did share, of course, sometimes legally and sometimes not. But an important mental shift had taken place: unrestricted sharing was no longer the assumed default. One had to first check to make sure it was OK to share, or else share covertly.

Richard Stallman and the Free Software Foundation

In some places, however, sharing was preserved as a standard practice. For example, in universities, the free exchange of information was a cultural norm, and academia was at least partially insulated from the commercial pressures of the computer industry. One such haven was the Artificial Intelligence Laboratory at MIT, where a young programmer named Richard Stallman worked in the 1970s and early '80s. As he later wrote:

> *We did not call our software "free software," because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.*
>
> (from www.gnu.org/gnu/thegnuproject.html)

Around 1980, however, industry trends finally started to affect even the AI Lab. A private company hired away many of the Lab's programmers to work on a proprietary operating system. Since their work would now take place under an exclusive license, they would not be free to share with their former colleagues anymore. At the same time, the AI Lab acquired new computer equipment that also came with a proprietary operating system; the members of the Lab would not be free to examine or change the source code without permission from the company that sold them the machine.

Stallman saw the situation as a stark political choice:

> *The modern computers of the era, such as the VAX or the 68020, had their own operating systems, but none of them were free software: you had to sign a nondisclosure agreement even to get an executable copy.*

> *This meant that the first step in using a computer was to promise not to help your neighbor. A cooperating community was forbidden. The rule made by the owners of proprietary software was, "If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them."*

His response was to resign from the AI Lab and form an independent nonprofit organization, the Free Software Foundation (FSF). Its flagship project would be GNU, a whimsically named but quite serious effort to build a completely free operating system, in which users would be guaranteed the right to study, modify, and share the source code. He was, in other words, trying to re-create what had been destroyed at the AI Lab, but on a worldwide scale and without the vulnerabilities that had led to the AI Lab's demise as a sharing community.

The GNU General Public License

He did this not only by writing code--though he wrote a lot, some of it quite good and widely used--but also by devising a copyright license whose terms guaranteed that his code would be perpetually free. The result, the GNU General Public License (GPL), is a clever piece of legal judo. It says that the code may be copied and modified without restriction, but that both copies and derivative works (that is, modified versions) must be distributed under the same license as the original, with no additional restrictions.

he GPL thus uses copyright law to achieve an effect opposite that of traditional copyright: instead of limiting the software's distribution, it prevents anyone, *even the author or copyright holder*, from limiting it. For Stallman, this was better than simply putting his code into the public domain. If it were in the public domain, any particular copy of it could be incorporated into a proprietary program.

While such incorporation wouldn't diminish the original code's continued availability, it would mean that Stallman's efforts could benefit the enemy--proprietary software. The GPL is a form of protectionism for free software: it prevents nonfree programs from taking advantage of any GPL'd code, while allowing all GPL'd programs to cooperate among themselves by sharing unrestrictedly.

The Rise of Open Collaboration

In some ways, Stallman's plan succeeded wildly. By means of the GPL and other writings, such as The GNU Manifesto, he put free software on the map as a political concept--even programmers who disagreed with the FSF's position still had to acknowledge and consider it. And Stallman eventually got the wholly free operating system he wanted, though many pieces were contributed by people not affiliated with the Free Software Foundation.

But in other ways, his original goal was overshadowed by the sheer technical success of collaborative programming under open copyright, an activity that was never the exclusive province of the FSF. Many others were doing it, and while less ideologically motivated than Stallman and the FSF, they were just as good at shipping code.

For example, the Berkeley Software Distribution (BSD), a gradual reimplementation of AT&T's Unix operating system, did not make overt political statements about the need for programmers to band together and share with one another. But the BSD group certainly knew how to do it in practice: they coordinated a massive distributed development effort, in which the Unix command-line utilities and code libraries, and eventually the operating system kernel itself, were rewritten from scratch mostly by volunteers. The BSD project released its code under a license very similar to the GPL, but without the clause insisting that all derivative works must be under the same license. Thus BSD code could be incorporated into proprietary systems, although of course that didn't detract from the freedom of the original code.

This separation of ideology from practice turned out to be one of the most important developments in free software in the late 1980s and early 1990s. It turned out that many programmers were happy to contribute their time to free software projects even when they didn't have a strong philosophical commitment to source code freedom. Programmers by nature hate duplicated effort, and if there's one thing free software is good at, it's avoidance of duplicated effort. When two programmers need to solve the same problem and there's no marketing or business reason for them to keep their work private, their instinct is to join forces, even if they've never met. What the free software movement did was create a standard framework for this sort of spontaneous collaboration. Programmers learned how to use computer networks to organize loosely knit groups of volunteers into functioning meritocracies, how to make projects inviting to both software developers and users, how to make decisions collectively, and how to handle conflicts between people who only know each other online.

Open copyright was crucial to the development of this system. Although free software licenses differ in some minor details, they all do basically the same thing: they prevent power monopolies in software projects, by giving any disaffected party the right to copy the code and take it in a new direction. In free software, this is known as *forking*: one copy continues along the original path, while another takes a different "fork" in the road. Most projects manage to avoid forks, but this is precisely because the implicit threat of a fork moderates everyone's behavior. Every participant knows that the only force holding things together is people's shared belief that they are better off working together than separately. Even a highly opinionated programmer will suddenly be motivated to compromise when the alternative is to go it alone.

The Free Software Foundation played a large role in developing this culture. It developed coding and documentation standards, and provided infrastructure support, such as mailing lists and file-sharing servers, for certain important projects. It also released some very useful programming tools as free software, which helped give the nascent community technical standards to organize around. But the FSF was by no means the only agent. Many people collaborated independently of the FSF, and there was a tremendous amount of cross-pollination between FSF and non-FSF projects.

While not all programmers agreed that all software should be free all the time, it became a cultural norm to put ideology aside and work together when there was code to be written. This norm arose because no individual participant's ideology could affect the freedom of the code anyway. Code written under a free license stays free, no matter what its various authors may think of software freedom in general. You may not always see eye to eye with your neighbor on economic policy, but if you both agree that the street outside needs plowing, then sharing the cost of a snowplow makes sense.

The difference between the two kinds of licenses--ones such as the GPL, which prohibit proprietary derivative works, and ones such as the BSD license, which allow them--thus turned out to not matter very much, at least as far as making software was concerned. Programmers who were willing to volunteer their time to work on free code at all generally didn't seem to care whether that code's license allowed proprietary derivative works. Some cared, of course, but for most, the decisive factor was the software's functionality. As long as the license allowed the basic freedoms necessary for unrestricted development and forking, it didn't matter if that license *also* permitted proprietary derivative versions.

Is It Free or Open Source?

As more and more people got involved with free software, the global pool of free programs expanded quickly. Because their source code was open to inspection by anyone, and because they could take advantage of massive parallelism in testing and debugging, many of these programs--particularly programming and networking tools--were of very high quality, and they gradually became part of the internet's basic infrastructure.

At the same time, the free software movement kept getting tripped up by an unfortunate linguistic coincidence: in English, the word *free* means both "costing no money" and "having liberty." In practice, free software satisfies both definitions: because you can always find someone to share a copy with you, the price of all copies is driven to zero by simple market dynamics. But for the FSF, the second definition, liberty, was the important one. After all, it's possible to have software that is available for no charge yet whose license prohibits redistribution or modification. Such software would not be "free" in the vital "freedom" sense. The FSF kept reminding everyone, "It's free as in freedom, not as in beer," but newcomers to free software still were regularly confused, because the language fails to distinguish carefully between low prices and liberty.

This recurring confusion frustrated a lot of people, and in 1998 a group of programmers came up with open source as a replacement for *free software*, creating the Open Source Initiative (OSI) to promote the new term. At first it wasn't clear that a schism had opened up in the free software world. But gradually it became obvious that the OSI was advocating a change not merely in terminology but also in attitude. As they saw it, the constant talk about freedom was off-putting to the corporate world, which was slowly beginning to wake up to the advantages of running and supporting free software. The OSI's position was: keep the same licenses, keep the same collaborative practices, but lose the talk of freedom and ideology; that would allow the movement to enter the mainstream and get many more participants and resources.

The OSI may have been correct. At least, it is undeniable that the term *open source* has caught on very effectively in the corporate world. How much of this is due to a desire to avoid talking about freedom, and how much comes from a marketing sense that the word *free* is fatally ambiguous, is impossible to say. But under the name "open source," an influx of for-profit dollars has changed the landscape of free software remarkably. Most major free software projects now have corporate backing of one form or another, and many of these companies have become quite adept at working with the volunteer developers who participate in the projects, and the users who report bugs and suggest new features.

The schism, such as it is, is an odd one, representing a profound philosophical disagreement that nonetheless has few practical consequences beyond terminology. Some people still say "free software" exclusively, because they want to remind people that freedom is the important thing. Some say "open source," either because they're not taking a position on the freedom question or because they find "free" too easily misunderstood. Some use the two terms interchangeably. (I'm in that camp, though in this article I've stuck to "free software" to match the title.) Any license that permits the basic freedoms necessary to allow unrestricted development is considered both a free software license and an open source license, and those freedoms, rather than the word used to describe them, seem to be what programmers look for when deciding whether to participate in a project.

The Future of Free Software

Those freedoms are also at the core of the business case for free software, even when the word *freedom* is not used. Free software offers a promise that few proprietary products can match: the promise that no one can take away from you that which you have invested time in learning and maintaining. When a corporation deploys a piece of software, even just internally, the corporation is making an investment. Employees will have to be trained; various internal processes will have to be adjusted to fit the software; documentation will have to be updated. Over time, important parts of the corporate infrastructure may grow to depend on the software's presence.

The more dependent a corporation is on a piece of software, the more it is at the mercy of the supplier of that software. When the software is proprietary, it means the corporation is at the mercy of the business decisions of a single software manufacturer. In theory, of course, that manufacturer doesn't want to disappoint its customers; it wants to keep them happy. But it might disappoint them anyway, by going bankrupt or by being bought by another company that makes a competing product. Even when those events are unlikely (say, when the supplier is Microsoft), the supplier might still disappoint its customers by failing to do the right usage research, by dropping backward compatibility in order to push customers along an upgrade path that they're not ready for, or by refusing to implement protocols that make it easy to interoperate with competing products.

A free software project, on the other hand, cannot be unilaterally shut down or taken down a wrong path. This doesn't mean that free software authors never make bad decisions; sometimes they do, just like any programmers. But the risk for users is much smaller, because a user who cares enough can simply take out the changes she doesn't like and put in the ones she does. When that user is a corporation, this is not just an abstract ideal but also a realizable practice. A corporation with a decent IT department can spend the effort necessary to customize software to its requirements, even if that means modifying the source code. It's a short step from there to contributing the changes back to others who use the software, and it's usually in the interests of everyone who uses the software to share their changes in common, since that reduces the maintenance burden on any single participant. The fact that no one, not even the original supplier, has the right to remove the software from circulation means that everyone is guaranteed that the effort they have put in so far can never be taken away.

Because of these freedoms and their practical consequences, I expect free software's share of desktop and office installations to continue growing rapidly, and for its current dominance in servers to solidify even further. But as a movement, it has implications beyond just the software we run on our computers. Free software's success, even at this early stage, calls into question some of the fundamental premises of intellectual property. If people will produce complex works of software without the monopoly control given by traditional copyright, will they do the same for books, songs, and movies? Custom tells us that copyright was designed to subsidize creation; but the vitality of the free software scene today, so strongly intertwined with the spread of the internet, hints that copyright may really have served to subsidize *distribution*, and that as distribution costs go to zero, people will start cooperating on works other than software. The degree to which this will come to pass is still an open question, but the free software movement has, at least, made it a question no one can ignore.

Karl Fogel has worked for CollabNet, Inc., since early 2000, managing the creation and development of Subversion, a version control system written from scratch by CollabNet and a team of open source volunteers. He is the author of Open Source Development with CVS, 3rd Edition, and the upcoming Producing Open Source Software, from O'Reilly Media.