Why you should use a BSD style license for your Open Source Project

Bruce Montague <brucem@alumni.cse.ucsc.edu >
Revision: 43184

FreeBSD is a registered trademark of the FreeBSD Foundation.

Intel, Celeron, Centrino, Core, EtherExpress, i386, i486, Itanium, Pentium, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document, and the FreeBSD Project was aware of the trademark claim, the designations have been followed by the "™" or the "®" symbol.

2013-11-13 by hrs.

Table of Contents

1. Introduction 2
2. Very Brief Open Source History 2
3. Unix from a BSD Licensing Perspective 2
4. The Current State of FreeBSD and BSD Licenses 3
5. The origins of the GPL 3
6. The origins of Linux and the LGPL 5
7. Open Source licenses and the Orphaning Problem 5
8. What a license cannot do 6
9. GPL Advantages and Disadvantages 6
10. BSD Advantages 7
11. Specific Recommendations for using a BSD license 8
12. Conclusion 10
13. Addenda 10

1. Introduction

This document makes a case for using a BSD style license for software and data; specifically it recommends using a BSD style license in place of the GPL. It can also be read as a BSD versus GPL Open Source License introduction and summary.

2. Very Brief Open Source History

Long before the term “Open Source” was used, software was developed by loose associations of programmers and freely exchanged. Starting in the early 1950's, organizations such as SHARE and DECUS developed much of the software that computer hardware companies bundled with their hardware offerings. At that time computer companies were in the hardware business; anything that reduced software cost and made more programs available made the hardware companies more competitive.

This model changed in the 1960's. In 1965 ADR developed the first licensed software product independent of a hardware company. ADR was competing against a free IBM package originally developed by IBM customers. ADR patented their software in 1968. To stop sharing of their program, they provided it under an equipment lease in which payment was spread over the lifetime of the product. ADR thus retained ownership and could control resale and reuse.

In 1969 the US Department of Justice charged IBM with destroying businesses by bundling free software with IBM hardware. As a result of this suit, IBM unbundled its software; that is, software became independent products separate from hardware.

In 1968 Informatics introduced the first commercial killer-app and rapidly established the concept of the software product, the software company, and very high rates of return. Informatics developed the perpetual license which is now standard throughout the computer industry, wherein ownership is never transferred to the customer.

3. Unix from a BSD Licensing Perspective

AT&T, who owned the original Unix implementation, was a publicly regulated monopoly tied up in anti-trust court; it was legally unable to sell a product into the software market. It was, however, able to provide it to academic institutions for the price of media.

Universities rapidly adopted Unix after an OS conference publicized its availability. It was extremely helpful that Unix ran on the PDP-11, a very affordable 16-bit computer, and was coded in a high-level language that was demonstrably good for systems programming. The DEC PDP-11 had, in effect, an open hardware interface designed to make it easy for customers to write their own OS, which was common. As DEC founder Ken Olsen famously proclaimed, “software comes from heaven when you have good hardware”.

Unix author Ken Thompson returned to his alma mater, University of California Berkeley (UCB), in 1975 and taught the kernel line-by-line. This ultimately resulted in an evolving system known as BSD (Berkeley Standard Distribution). UCB converted Unix to 32-bits, added virtual memory, and implemented the version of the TCP/IP stack upon which the Internet was essentially built. UCB made BSD available for the cost of media, under what became known as “the BSD license”. A customer purchased Unix from AT&T and then ordered a BSD tape from UCB.

In the mid-1980s a government anti-trust case against ATT ended with the break-up of ATT. ATT still owned Unix and was now able to sell it. ATT embarked on an aggressive licensing effort and most commercial Unixes of the day became ATT-derived.

In the early 1990's ATT sued UCB over license violations related to BSD. UCB discovered that ATT had incorporated, without acknowledgment or payment, many improvements due to BSD into ATT's products, and a lengthy court case, primarily between ATT and UCB, ensued. During this period some UCB programmers embarked on a project to rewrite any ATT code associated with BSD. This project resulted in a system called BSD 4.4-lite (lite because it was not a complete system; it lacked 6 key ATT files).

A lengthy series of articles published slightly later in Dr. Dobbs magazine described a BSD-derived 386 PC version of Unix, with BSD-licensed replacement files for the 6 missing 4.4 lite files. This system, named 386BSD, was due to ex-UCB programmer William Jolitz. It became the original basis of all the PC BSDs in use today.

In the mid 1990s, Novell purchased ATT's Unix rights and a (then secret) agreement was reached to terminate the lawsuit. UCB soon terminated its support for BSD.

4. The Current State of FreeBSD and BSD Licenses

The so-called new BSD license applied to FreeBSD within the last few years is effectively a statement that you can do anything with the program or its source, but you do not have any warranty and none of the authors has any liability (basically, you cannot sue anybody). This new BSD license is intended to encourage product commercialization. Any BSD code can be sold or included in proprietary products without any restrictions on the availability of your code or your future behavior.

Do not confuse the new BSD license with “public domain”. While an item in the public domain is also free for all to use, it has no owner.

5. The origins of the GPL

While the future of Unix had been so muddled in the late 1980s and early 1990s, the GPL, another development with important licensing considerations, reached fruition.

Richard Stallman, the developer of Emacs, was a member of the staff at MIT when his lab switched from home-grown to proprietary systems. Stallman became upset when he found that he could not legally add minor improvements to the system. (Many of Stallman's co-workers had left to form two companies based on software developed at MIT and licensed by MIT; there appears to have been disagreement over access to the source code for this software). Stallman devised an alternative to the commercial software license and called it the GPL, or "GNU Public License". He also started a non-profit foundation, the Free Software Foundation (FSF), which intended to develop an entire operating system, including all associated software, that would not be subject to proprietary licensing. This system was called GNU, for "GNU is Not Unix".

The GPL was designed to be the antithesis of the standard proprietary license. To this end, any modifications that were made to a GPL program were required to be given back to the GPL community (by requiring that the source of the program be available to the user) and any program that used or linked to GPL code was required to be under the GPL. The GPL was intended to keep software from becoming proprietary. As the last paragraph of the GPL states:

“This General Public License does not permit incorporating your program into proprietary programs.”[1]

The GPL is a complex license so here are some rules of thumb when using the GPL:

- you can charge as much as you want for distributing, supporting, or documenting the software, but you cannot sell the software itself.
- the rule-of-thumb states that if GPL source is required for a program to compile, the program must be under the GPL. Linking statically to a GPL library requires a program to be under the GPL.
- the GPL requires that any patents associated with GPLed software must be licensed for everyone's free use.
- simply aggregating software together, as when multiple programs are put on one disk, does not count as including GPLed programs in non-GPLed programs.
- output of a program does not count as a derivative work. This enables the gcc compiler to be used in commercial environments without legal problems.
- since the Linux kernel is under the GPL, any code statically linked with the Linux kernel must be GPLed. This requirement can be circumvented by dynamically linking loadable kernel modules. This permits companies to distribute binary drivers, but often has the disadvantage that they will only work for particular versions of the Linux kernel.

Due in part to its complexity, in many parts of the world today the legalities of the GPL are being ignored in regard to Linux and related software. The long-term ramifications of this are unclear.

6. The origins of Linux and the LGPL

While the commercial Unix wars raged, the Linux kernel was developed as a PC Unix clone. Linus Torvalds credits the existence of the GNU C compiler and the associated GNU tools for the existence of Linux. He put the Linux kernel under the GPL.

Remember that the GPL requires anything that statically links to any code under the GPL also be placed under the GPL. The source for this code must thus be made available to the user of the program. Dynamic linking, however, is not considered a violation of the GPL. Pressure to put proprietary applications on Linux became overwhelming. Such applications often must link with system libraries. This resulted in a modified version of the GPL called the LGPL ("Library", since renamed to "Lesser", GPL). The LGPL allows proprietary code to be linked to the GNU C library, glibc. You do not have to release the source to code which has been dynamically linked to an LGPLed library.

If you statically link an application with glibc, such as is often required in embedded systems, you cannot keep your application proprietary, that is, the source must be released. Both the GPL and LGPL require any modifications to the code directly under the license to be released.

7. Open Source licenses and the Orphaning Problem

One of the serious problems associated with proprietary software is known as “orphaning”. This occurs when a single business failure or change in a product strategy causes a huge pyramid of dependent systems and companies to fail for reasons beyond their control. Decades of experience have shown that the momentary size or success of a software supplier is no guarantee that their software will remain available, as current market conditions and strategies can change rapidly.

The GPL attempts to prevent orphaning by severing the link to proprietary intellectual property.

A BSD license gives a small company the equivalent of software-in-escrow without any legal complications or costs. If a BSD-licensed program becomes orphaned, a company can simply take over, in a proprietary manner, the program on which they are dependent. An even better situation occurs when a BSD code-base is maintained by a small informal consortium, since the development process is not dependent on the survival of a single company or product line. The survivability of the development team when they are mentally in the zone is much more important than simple physical availability of the source code.

8. What a license cannot do

No license can guarantee future software availability. Although a copyright holder can traditionally change the terms of a copyright at anytime, the presumption in the BSD community is that such an attempt simply causes the source to fork.

The GPL explicitly disallows revoking the license. It has occurred, however, that a company (Mattel) purchased a GPL copyright (cphack), revoked the entire copyright, went to court, and prevailed [2]. That is, they legally revoked the entire distribution and all derivative works based on the copyright. Whether this could happen with a larger and more dispersed distribution is an open question; there is also some confusion regarding whether the software was really under the GPL.

In another example, Red Hat purchased Cygnus, an engineering company that had taken over development of the FSF compiler tools. Cygnus was able to do so because they had developed a business model in which they sold support for GNU software. This enabled them to employ some 50 engineers and drive the direction of the programs by contributing the preponderance of modifications. As Donald Rosenberg states "projects using licenses like the GPL...live under constant threat of having someone take over the project by producing a better version of the code and doing it faster than the original owners." [3]

9. GPL Advantages and Disadvantages

A common reason to use the GPL is when modifying or extending the gcc compiler. This is particularly apt when working with one-off specialty CPUs in environments where all software costs are likely to be considered overhead, with minimal expectations that others will use the resulting compiler.

The GPL is also attractive to small companies selling CDs in an environment where "buy-low, sell-high" may still give the end-user a very inexpensive product. It is also attractive to companies that expect to survive by providing various forms of technical support, including documentation, for the GPLed intellectual property world.

A less publicized and unintended use of the GPL is that it is very favorable to large companies that want to undercut software companies. In other words, the GPL is well suited for use as a marketing weapon, potentially reducing overall economic benefit and contributing to monopolistic behavior.

The GPL can present a real problem for those wishing to commercialize and profit from software. For example, the GPL adds to the difficulty a graduate student will have in directly forming a company to commercialize his research results, or the difficulty a student will have in joining a company on the assumption that a promising research project will be commercialized.

For those who must work with statically-linked implementations of multiple software standards, the GPL is often a poor license, because it precludes using proprietary implementations of the standards. The GPL thus minimizes the number of programs that can be built using a GPLed standard. The GPL was intended to not provide a mechanism to develop a standard on which one engineers proprietary products. (This does not apply to Linux applications because they do not statically link, rather they use a trap-based API.)

The GPL attempts to make programmers contribute to an evolving suite of programs, then to compete in the distribution and support of this suite. This situation is unrealistic for many required core system standards, which may be applied in widely varying environments which require commercial customization or integration with legacy standards under existing (non-GPL) licenses. Real-time systems are often statically linked, so the GPL and LGPL are definitely considered potential problems by many embedded systems companies.

The GPL is an attempt to keep efforts, regardless of demand, at the research and development stages. This maximizes the benefits to researchers and developers, at an unknown cost to those who would benefit from wider distribution.

The GPL was designed to keep research results from transitioning to proprietary products. This step is often assumed to be the last step in the traditional technology transfer pipeline and it is usually difficult enough under the best of circumstances; the GPL was intended to make it impossible.

10. BSD Advantages

A BSD style license is a good choice for long duration research or other projects that need a development environment that:

- has near zero cost
- will evolve over a long period of time
- permits anyone to retain the option of commercializing final results with minimal legal issues.

This final consideration may often be the dominant one, as it was when the Apache project decided upon its license:

“This type of license is ideal for promoting the use of a reference body of code that implements a protocol for common service. This is another reason why we choose it for the Apache group - many of us wanted to see HTTP survive and become a true multiparty standard, and would not have minded in the slightest if Microsoft or Netscape choose to incorporate our HTTP engine or any other component of our code into their products, if it helped further the goal of keeping HTTP common... All this means that, strategically

speaking, the project needs to maintain sufficient momentum, and that participants realize greater value by contributing their code to the project, even code that would have had value if kept proprietary."

Developers tend to find the BSD license attractive as it keeps legal issues out of the way and lets them do whatever they want with the code. In contrast, those who expect primarily to use a system rather than program it, or expect others to evolve the code, or who do not expect to make a living from their work associated with the system (such as government employees), find the GPL attractive, because it forces code developed by others to be given to them and keeps their employer from retaining copyright and thus potentially "burying" or orphaning the software. If you want to force your competitors to help you, the GPL is attractive.

A BSD license is not simply a gift. The question "why should we help our competitors or let them steal our work?" comes up often in relation to a BSD license. Under a BSD license, if one company came to dominate a product niche that others considered strategic, the other companies can, with minimal effort, form a mini-consortium aimed at reestablishing parity by contributing to a competitive BSD variant that increases market competition and fairness. This permits each company to believe that it will be able to profit from some advantage it can provide, while also contributing to economic flexibility and efficiency. The more rapidly and easily the cooperating members can do this, the more successful they will be. A BSD license is essentially a minimally complicated license that enables such behavior.

A key effect of the GPL, making a complete and competitive Open Source system widely available at cost of media, is a reasonable goal. A BSD style license, in conjunction with ad-hoc-consortiums of individuals, can achieve this goal without destroying the economic assumptions built around the deployment-end of the technology transfer pipeline.

11. Specific Recommendations for using a BSD license

- The BSD license is preferable for transferring research results in a way that will widely be deployed and most benefit an economy. As such, research funding agencies, such as the NSF, ONR and DARPA, should encourage in the earliest phases of funded research projects, the adoption of BSD style licenses for software, data, results, and open hardware. They should also encourage formation of standards based around implemented Open Source systems and ongoing Open Source projects.

- Government policy should minimize the costs and difficulties in moving from research to deployment. When possible, grants should require results to be available under a commercialization friendly BSD style license.

- In many cases, the long-term results of a BSD style license more accurately reflect the goals proclaimed in the research charter of universities then what occurs when results

are copyrighted or patented and subject to proprietary university licensing. Anecdotal evidence exists that universities are financially better rewarded in the long run by releasing research results and then appealing to donations from commercially successful alumni.

- Companies have long recognized that the creation of de facto standards is a key marketing technique. The BSD license serves this role well, if a company really has a unique advantage in evolving the system. The license is legally attractive to the widest audience while the company's expertise ensures their control. There are times when the GPL may be the appropriate vehicle for an attempt to create such a standard, especially when attempting to undermine or co-opt others. The GPL, however, penalizes the evolution of that standard, because it promotes a suite rather than a commercially applicable standard. Use of such a suite constantly raises commercialization and legal issues. It may not be possible to mix standards when some are under the GPL and others are not. A true technical standard should not mandate exclusion of other standards for non-technical reasons.

- Companies interested in promoting an evolving standard, which can become the core of other companies' commercial products, should be wary of the GPL. Regardless of the license used, the resulting software will usually devolve to whoever actually makes the majority of the engineering changes and most understands the state of the system. The GPL simply adds additional legal friction to the result.

- Large companies, in which Open Source code is developed, should be aware that programmers appreciate Open Source because it leaves the software available to the employee when they change employers. Some companies encourage this behavior as an employment perk, especially when the software involved is not directly strategic. It is, in effect, a front-loaded retirement benefit with potential lost opportunity costs but no direct costs. Encouraging employees to work for peer acclaim outside the company is a cheap portable benefit a company can sometimes provide with near zero downside.

- Small companies with software projects vulnerable to orphaning should attempt to use the BSD license when possible. Companies of all sizes should consider forming such Open Source projects when it is to their mutual advantage to maintain the minimal legal and organization overheads associated with a true BSD-style Open Source project.

- Non-profits should participate in Open Source projects when possible. To minimize software engineering problems, such as mixing code under different licenses, BSD-style licenses should be encouraged. Being leery of the GPL should particularly be the case with non-profits that interact with the developing world. In some locales where application of law becomes a costly exercise, the simplicity of the new BSD license, as compared to the GPL, may be of considerable advantage.

12. Conclusion

In contrast to the GPL, which is designed to prevent the proprietary commercialization of Open Source code, the BSD license places minimal restrictions on future behavior. This allows BSD code to remain Open Source or become integrated into commercial solutions, as a project's or company's needs change. In other words, the BSD license does not become a legal time-bomb at any point in the development process.

In addition, since the BSD license does not come with the legal complexity of the GPL or LGPL licenses, it allows developers and companies to spend their time creating and promoting good code rather than worrying if that code violates licensing.

13. Addenda


```
[1] http://www.gnu.org/licenses/gpl.html

[2] http://archives.cnn.com/2000/TECH/computing/03/28/cyberpatrol.↺
mirrors/

[3] Open Source: the Unauthorized White Papers, Donald K. ↺
Rosenberg, IDG Books,
    2000. Quotes are from page 114, ``Effects of the GNU GPL''.

[4] In the "What License to Use?" section of
    http://www.oreilly.com/catalog/opensources/book/brian.html

This whitepaper is a condensation of an original work available at
http://alumni.cse.ucsc.edu/~brucem/open_source_license.htm
```