Software Freedom
Law Center

1995 Broadway, 17th Floor
New York, NY 10023–5882
tel +1–212–580–0800
fax +1–212–580–0898
www.softwarefreedom.org

Managing copyright information within a free software project

September 12, 2012

1 Introduction and purpose

Nearly every free software project includes two types of legal information with its source code: the license(s) of the code and the copyright notices of contributing authors. As important as it is to maintain this information, it's not always easy in a collaborative development process. With multiple developers contributing regularly to the code, some information can be left out or buried, and other information can be retained after it is no longer accurate. We wrote this guide to help projects minimize the problems and improve the usefulness of their legal information. It explains the purpose of license and copyright notices, the legal requirements behind them, and the current best practices for maintaining this information in a free software project's source distribution.[1]

License notices tell downstream developers and users what they are permitted to do with the code. We are sometimes asked by new participants in free software development how to "open source" a piece of software; usually the answer is that the copyright holders should make the source code available with a free software license applied to it. Without license information, third parties may (and should) assume that the default rules of copyright apply, i.e. that they have no permission to copy, modify, or redistribute the code. In a project where different licenses apply to different portions of the codebase, the license information should clearly delineate which licenses apply to which source files. Preserving third-party notices not only helps clarify what terms apply to those individual components—it is also required by almost all free software licenses.

2 The importance of good revision control practices

While manually maintained license and copyright notices have served these purposes reasonably well, they are prone to human error and cannot precisely identify the work of multiple contributors within a single file. For this reason, a revision control system (more commonly referred to as a version control system or VCS) is required to maintain accurate records of a project's copyright ownership structure.[2]

[1] While this document discusses copyright licenses and other legal concepts, it is not intended as legal advice. Application of free software license requirements to the circumstances of any specific distribution is beyond the scope of this document. If you have need of specific legal advice, please contact the Software Freedom Law Center or other legal counsel.

[2] The only exceptions to this rule are projects that will only ever have one developer, and will not use any third-party code.

Tracking authorship with a centralized VCS like SVN or CVS requires some manual intervention. Projects using centralized VCSs typically have a relatively small number of maintainers with commit access to the project's canonical repository. The maintainers receive contributions from non-committers in the form of patch/diff files, merge patches they like into their own checkouts of the repository, and then commit them under their own names (perhaps with a comment noting that the change is merged on behalf of the contributor). If kept consistently, these comments can provide relatively complete authorship data, so if your project uses a centralized VCS, you should carefully record the origin of each contribution. But because they're entered manually they're prone to omissions, misspellings, or misattribution. SFLC has been called on to audit projects' code to verify the origin of contributions, and if neither the metadata nor the logs associated with commits reliably identifies the authors, it's a painstaking and often inconclusive process.

Modern distributed version control systems (DVCSs) like Git and Mercurial keep better records for attribution purposes than centralized systems. When contributions are merged from one developer's repository into another, metadata identifying the contributor is preserved, so maintainers need not rely on manually produced commit logs to record who wrote what.

But though DVCSs keep better contributor data than centralized VCSs, they can't do everything – developers must observe some best practices to get really accurate authorship information out of them. First, when moving code around within a project, use DVCS commands to do it whenever possible. For example, on DVCSs that implement a true `move` command (rather than just an `add` followed by a `remove`), use that command rather than copying the file on the file system and using the `add` and `remove` commands to commit it. Second, commit messages should still be used to identify any third-party code. If a commit is made up both original code and code copied and pasted from another free software project, the metadata won't reflect the authorship of the third-party code. In the commit message, identify the author, origin, and license of the code. Do the same for code moving from one file to another; if one developer moves code written by another developer into a library, for example, the metadata will identify the commit to the library with the developer who moved the code. Use commit logs to identify the code's actual author. Finally, keep a good Changelog that identifies contributors with significant commits; a curated summary of the project's history will serve as a helpful index to the VCS logs, should an audit ever be necessary.

Proper use of a VCS is good development practice as well as good administrative practice. While keeping thorough commit logs may seem tedious at times, it is far less tedious than combing through poorly versioned code trying to establish its provenance. And particularly for large projects that use lots of third-party free software, it can make it drastically easier to track compliance with all of those licenses.

3 Best practices for maintaining copyright notices

Contrary to popular belief, copyright notices aren't required to secure copyright. Each developer holds copyright in his or her code the moment it is written, and because all the world's major copyright systems—including the US after 1976—do not require notices, publishing code without a copyright notice doesn't change this. However, notices do have some legal effect. For example, someone who infringes the copyright of a program published without a notice may be able to claim that the infringement was "innocent" because he or she had no notice of the developers' copyright claim, and thus seek reduced damages.

There are other good reasons to include copyright notices as well. They acknowledge the developers' contributions to the project. They also serve as a record of people who claim rights in the codebase, which may be needed if the project later wishes to seek the contributors' permission to change its license. Finally, when you incorporate third-party free software into your project, you must include the corresponding copyright notices—nearly every free software license requires it.

3.1 When to add and remove copyright notices[3]

Not every contribution to a free software project is copyrightable—some may exhibit too little originality, usually because they represent one of very few means of expressing particular functionality. The originality requirements are out of the scope of this document, but are covered thoroughly by our publication *Originality Requirements under U.S. and E.U. Copyright Law*.[4] In short, there's no clear rule by which to judge whether a contribution is copyrightable.

Since copyright notices are not mandatory, there is generally no harm in under-using them, particularly if authorship is recorded in VCS logs. Over-use, however, can cause problems. Some developers configure their text editors to insert copyright notices into every file they touch in a given commit; if all they did was move or rename the file, this is almost certainly improper. A proliferation of unwarranted copyright notices can cause the project administrative headaches, for example if it ever needs to contact all of the copyright holders for consent to a change in the license. It's a good idea to have developers add their own copyright notices, since a copyright notice is a legal representation being made regarding the developer's rights, but the project should also discourage contributors from indiscriminately adding notices.

But be careful when removing the notices of other developers. Since free software licenses require licensees to preserve notices, wrongfully removing one is a violation of the license from that contributor and may be copyright infringement. If it's absolutely clear that every remnant of a developer's contribution has been removed, then it is probably OK to remove the associated copyright notice; otherwise, it's best to keep it around. However, a requirement to "preserve" or "reproduce" a developer's copyright notice does not necessarily require that the notice be kept in exactly the same place it started; it's usually acceptable to move notices from individual source files to a central attribution file, for example.

3.2 Comparing two systems: file-scope and centralized notices

One common system for maintaining copyright information in a free software distribution is to include copyright notices at the top of every file for each developer who contributed code to that file—we'll call this the "file-scope" approach. The other system is a "centralized" approach, where copyright notices are collected in a single top-level file (e.g. AUTHORS or COPYRIGHT). We'll describe best practices for each, as well as their respective strengths and weaknesses.

Maintaining file-scope copyright notices According to the file-scope notice approach, each time a contributor modifies a file significantly, his or her copyright notice should be added to the top of the file. If a notice for that developer already exists, it should be updated as necessary to reflect the year the new contribution was made.The developer may include a description of the modification along with the notice:

> Copyright 2012 Jane Hacker <jhacker@example.org> Fixed random number generator to output numbers other than 9.

Well-maintained file-scope copyright notices make it possible to tell at a glance which developers made modifications to a given file. This eases the task of auditing the code in the case of an authorship dispute and can provide useful evidence in license enforcement actions. If a file is used separately from the rest of the codebase, its authors can still be clearly identified.

[3] Under U.S. law, a properly formatted copyright notice should include: 1) the © symbol, the word "Copyright", or the abbreviation "Copr."; 2) the year of the work's first publication; and 3) the name of the copyright owner. For example, "Copyright 2012 Jane Hacker." If contributions were made during several years, use commas to separate nonconsecutive years and hyphens to indicate contributions over several consecutive years, e.g.: "Copyright 2007, 2009, 2010 – 2012 Jane Hacker."

[4] http://softwarefreedom.org/resources/2007/originality-requirements.html

However, there are also several problems with file-scope notices, particularly in large projects. They require a considerable amount of maintenance and attention over time as the size and developer base of the project increase. In a small program consisting of a few files, file-scope copyright notices are relatively easy to maintain. In a large project with dozens of developers and several committers, it is more likely that notices will be left off. File-scope notices can also become outdated when developers' contributions disappear as a result of subsequent modifications. The result is an inaccurate record of a file's contributors. Even if well-kept, file-scope notices may present a misleading picture of a project's copyright ownership structure. A single contribution may involve changes to several files, all related to a single task (e.g. a new feature, a bug fix, a reorganization). If the contribution is copyrightable, the copyright is in the contribution as a whole (or possibly multiple contributions to a single version of the program)[5] rather than in individual changes to separate files. File-scope notices imply the opposite.

File-scope notices became customary before the rise of DVCSs. Most of the things that file-scope notices do well, DVCSs do better: they record with much finer granularity which developers contributed to which files, and they allow comments describing modifications to be associated with individual commits rather than entire files. They also make it possible to determine conclusively when a given developer's code has disappeared from a file or from the project entirely. (These benefits assume that commits are always associated with their authors. If committers commonly submit code "on behalf of" contributors, the evidentiary benefits of DVCSs decrease, but this practice appears less common among projects that use DVCSs than those that use centralized VCSs.)

There is one advantage of file-scope notices that using a DVCS can't substitute for: if a file is separated from the codebase using a mechanism other than the DVCS (i.e. it is manually copied elsewhere), the individual authorship information will not be preserved. We discuss below how to address this system in a centralized-notice system.

Maintaining centralized copyright notices The centralized notice approach consolidates all copyright notices in a single location, usually a top-level file. This file should contain all of the copyright notices provided project contributors, unless the contribution was clearly insignificant. It may also credit—without a copyright notice—anyone who helped with the project but did not contribute code or other copyrighted material.

This approach captures less information about contributions within individual files, recognizing that the DVCS is better equipped to record those details. As we mentioned before, it does have one disadvantage as compared to the file-scope approach: if a single file is separated from the distribution, the recipient won't see the contributors' copyright notices. But this can be easily remedied by including a single copyright notice in each file's header, pointing to the top-level file:

> Copyright 2012 The Foo Project Developers. See the COPYRIGHT file at the top-level directory of this distribution and at http://example.org/project/COPYRIGHT.

4 Best practices for maintaining license information

License information can also be maintained in individual files, in a central location, or in some combination of both. Most projects use a hybrid approach, placing the primary license in a top-level COPYING or LICENSE file, and also including some license information in each file's header.

[5] U.S. Copyright law wasn't designed with free software in mind. Its model is the one-time or occasional publication of "editions" or installments of a work, not the near-constant publication of small changes to a publicly available work. Because of this, it's difficult to generalize about what the copyrightable unit of a free software project or contribution is.

4.1 Maintaining file-scope license information

There are two common approaches to providing file-scope license information in file headers. The first is the one recommended by the Free Software Foundation in the GPL's epilogue "How to Apply These Terms to Your New Programs."[6] The FSF recommends placing the following information in each source file's header: 1) one sentence naming and describing the program; 2) the copyright notice of the author(s); 3) a statement that the program is free software and naming the license(s) under which it is available; 4) a brief warranty disclaimer; and 5) a URL pointing to a full copy of the license. The Apache Software Foundation recommends a very similar notice for Apache-licensed projects.[7]

The other popular approach is to include the entire license in the file's header. For obvious reasons, this is much more common for projects that use very short licenses, such as BSD or MIT variants—longer licenses such as the GPL, MPL, or even Apache make impractical headers. Both approaches to file-scope license information are intended to serve basically the same goals—to ensure that no matter where the code ends up, recipients will be aware of their rights and obligations under the license, and the developers' warranty disclaimer will still protect them from liability. File-scope license notices also make it possible to distinguish individual files that are licensed under different free software licenses from the distribution as a whole.[8]

File-scope license information, like file-scope copyright notices, can degrade or become difficult to manage over time. When each file has only one license, file-scope notices work well. When code under another license is introduced, the license header not only grows but becomes less useful, because it can't define which license applies to which portions of the file. File-scope licenses also make changing or qualifying the project's license cumbersome, requiring a change to every file in the code base. If any files are missed, the licensing of the whole code base can become unclear. Finally, scattered license information can make it difficult for downstream users to determine the license of the project; keeping a definitive collection of all applicable licenses in one place makes this easier, and may encourage adoption by risk-averse consumers.

The problem of license proliferation within individual files can be avoided by keeping third-party code distinct in your codebase. SFLC's whitepaper *Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers*[9] provides guidance on this. However, the other issues already discussed are endemic to the file-scope approach; their seriousness is directly proportional to the project's scale and complexity. Centralizing this information can make it easier to maintain.

4.2 Centralizing license notices

A centralized approach to license notices addresses the inherent problems with file-scope notices by trading specificity of file-level information for manageability. All licenses that apply to the code are collected in a central location, usually in a file or files at the top level of the code. For projects with only one license, one file (COPYING is traditional, but LICENSE is perhaps less ambiguous) is sufficient. A project with a more complex licensing structure might prefer to keep a handful of files—a central LICENSE file explaining the overall license of the project, listing the applicable component licenses and describing any exceptions, and separate files (linked or referred to by the central file) containing the actual text of the individual licenses. File notices are bare-bones, leaving most of the work to the top-level LICENSE file. For example:

> This file is part of Foo Project. It is subject to the license terms in the LICENSE file found in the top-level directory of this distribution and at http://www.example.org/foo/license.html.

[6] http://www.gnu.org/licenses/gpl.html#howto

[7] http://www.apache.org/licenses/LICENSE-2.0.html#apply

[8] For an example, see the SFLC white paper *Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers*, http://www.softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html.

[9] http://www.softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html

> No part of Foo Project, including this file, may be copied, modified, propagated, or distributed except according to the terms contained in the LICENSE file.

As long as the license at that URL is maintained, this notice directs anyone with a copy of the file (even if it has been separated from the rest of the code) to the project's license information as reliably as a file-scope notice, but it also has two advantages: it means that changes to the LICENSE file propagate without a cross-file find-and-replace, and it's much shorter than typical license notices. While brevity may not be essential for compiled languages, which discard comments (including notices) in compilation, it can be important for Javascript libraries and other interpreted code.

In many situations, a semi-centralized approach will work best. Consider a GPL-licensed project whose codebase contains a number of permissively-licensed libraries with custom modifications. While the license of the resulting program will be GPL, because of GPL's copyleft provision, the source for the libraries can be maintained under their individual licenses.[10] In this scenario, it makes sense to keep license information for the individual libraries at the top level of the directories they're kept in, as well as in the license information for the entire project.

4.3 Handling license exceptions

Many copyleft-licensed projects offer an exception allowing their code to be linked to a particular library or other application with an incompatible free software license.[11] If your project offers a linking exception and your license information is centralized, you should include the exception in the LICENSE file (either verbatim or by reference to a separate file containing the exception, as with longer license texts). If you are using file-scope license information the exception should be referenced in all of the license notices.

5 Conclusion

The copyright and license information maintained by a free software project is the primary record of the project's ownership and license structure. By following the guidelines above, projects can improve that record's usefulness and hopefully save effort down the road, should a license change become necessary or a license dispute arise. If your project encounters issues related to license and copyright notices that aren't addressed in this guide and would like help with your specific situation, please contact the Software Freedom Law Center at help@softwarefreedom.org.

[10] See *Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers*, http://www.softwarefreedom.org/resources/2007/gpl-non-gpl-collaboration.html

[11] http://www.gnu.org/licenses/gpl-faq.html#GPLIncompatibleLibs