This lesson is being piloted (Beta version)

Software Licensing (/software-licensing/index.html)

Terminology and Background on Intellectual Property

Overview

Teaching: 5 min
Exercises: 5 min
Questions

- What is the primary form of intellectual property typically associated with software?
- What is the purpose of a license for software?
- At what point can you assert copyright over your software?

Objectives

- Be able to differentiate the terms copyright, patent, trademark, and license.
- Understand that your creative works (including software) are copyright at creation.

Terminology: copyright, patents, trademarks, and licenses

Intellectual property (IP) is a general term referring to intangible creations of human intellect. There are multiple types of IP recognized in law in most jurisdictions. The types that are most often associated with software include:

- **Copyright** grants the creator of an original work (e.g., a software package) exclusive rights to its use and distribution, including limits on derivative works.
- A **patent** grants the inventor of something new, useful, and non-obvious (which may be embodied in software) the rights to its production, use, and distribution.
- A **trademark** is a sign, design, or expression which identifies products or services from a particular source (e.g., your software), as distinguished from other sources.

Licenses are a legal tool to transfer (selected) rights in a work, invention, or mark (forms of intellectual property) from one party to another. When we talk about licenses for software, we're primarily focused on copyright. But some software licenses include clauses pertaining to patents and trademarks related to that software.

Your software starts out copyrighted

Under the law, the software you write is subject to **copyright on creation**. You don't have to do anything special to claim copyright.

Normally, the creator of the work owns the copyright in the work. But it is "work for hire" (i.e., as part of your job), often the employer will own the copyright. Employment contracts often make IP rights explicit. If your employer owns the copyright, you probably have to get formal permission to license and distribute your software.

Unless you specify some license for your software, all rights in the software are reserved to the copyright owner.

Special case: U.S. government works

Works created by the US government (and its employees) cannot be copyrighted. They are considered to be in the **public domain**. The motivation for this was to ensure public access to the U.S. legal code. This does not apply to works created by *contractors* for the U.S. government (e.g., federally funded research).

Activity

Who owns the copyright in the software you create? You or your employer?

If you don't know the answer, it might be useful to find out. If you don't have a copy of your employment contract, consider asking your Human Resources department for a copy. Your supervisor or your institutional Technology Transfer office may also be able to help you answer this question.

Key Points

- Copyright is the primary form of intellectual property associated with software. Patents and trademarks may also be relevant.
- A license is a legal tool to transfer selected intellectual property rights from one party to another.
- Creative works, including software, are subject to copyright protections from the moment of creation.

Why You Should Choose a License

Overview

Teaching: 6 min
Exercises: 20 min
Questions

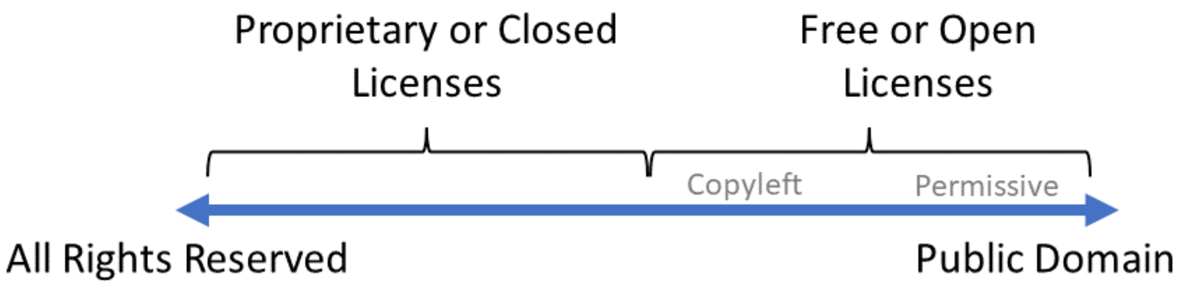
- What are the two basic categories of software licenses?
- What are the benefits of specifying a license for your software?

Objectives

- Understand the reasons to specify a license for your software.

The spectrum of software licenses

As we've mentioned, licenses provide a means to convey selected rights from the owner of those rights to others. Different licenses can be defined that convey different rights. You can think of software licenses as spanning a spectrum.



At one extreme, all rights are reserved to the owner of the copyright. This is the situation that applies when you do not specify a license, but it is also common to see "All rights reserved" stated explicitly as a form of license.

Proprietary licenses, also referred to as closed-source licenses, typically convey rights to use the software but reserve rights to access or distribute the source code. Software that is distributed under a proprietary license is most often provided in the form of an executable, though in some cases, licensees may receive the source code (e.g., so that they can build it on platforms that the copyright owner may not have access to), but be restricted from redistributing it.

Free or open licenses generally convey more rights to the licensee, typically including access to the source code and the right to redistribute it. Within the range of open licenses, they can be broadly categorized as "copyleft" or "permissive". We'll take a deeper look at open licenses in the next episode.

At the other extreme of the licensing spectrum is the public domain, which is not so much a license as a disclaimer of all rights in the work. Works in the public domain do not have a copyright or a copyright holder. Anyone can do whatever they want with such works.

Activity

Identify one software package that’s important to your work that has a proprietary license.

Identify one software package that’s important to your work that has an open-source license.

Is there a software package that’s important to your work that doesn’t specify a license?

Why license your software

Licenses play an important role for both developers and users of software. The license provides guidance as to how developers can contribute to the software and how users can approach using it.

Developers should expect that their contributions to the code base will be licensed on the same terms as the original work (unless other arrangements are made). With closed-source licenses, the developer may lose subsequent access to their contributions once they are subsumed into the proprietary code base. Open source licenses, on the other hand, provide for access to the source code and the ability to redistribute the code. Developers contributing to open-source projects can therefore expect ongoing access to their contributions and the ability to redistribute the code.

The user’s perspective is somewhat similar. With a proprietary license, they are likely to be limited to using executables that the copyright holder chooses to make available. If the source code is not available, the user has no way of understanding what the code is actually doing. Open-source licenses, on the other hand, ensure the availability of the source code, and so the user’s ability to (try to) build the software on different computer platforms. The user can review the code to see what it is doing when it is run. While there may well be helpful user communities in either case, the fact that everyone in the community has access to the source code of an open-source package may make it more likely that the user can obtain assistance from people besides the copyright holder.

Discussion

Suppose you hear someone at a conference talk about a software package that might be very useful in your work with a few modifications. You chat with them about the possibility of collaborating around their software.

If they mention that the software is proprietary, would that influence your decision about pursuing a collaboration?

If they mention that the software is open source, would that influence your decision about pursuing a collaboration?

Ultimately, the choice of how to license your software should be thought of as a *tool* in pursuing the greater goals of your software and your project.

Key Points

- The two basic categories of software licenses are *proprietary* and *open-source*.
- Specifying a software licenses provides guidance for would-be contributors and users about how they can engage with the software.

What is Open Source?

Overview

Teaching: 17 min
Exercises: 15 min
Questions

- What organization is considered to be the arbiter of whether or not a license is open source?
- What are the 'four freedoms' by which the Free Software Foundation defines free (aka open-source) software?
- What is the difference between a permissive and a copyleft license?
- Is there a licensing scheme comparable to open-source for non-software works?

Objectives

- Know where to check whether a license is open-source.
- Understand how open-source software is defined.
- Understand the difference between copyleft and permissive open-source licenses.
- Be aware of the Creative Commons licenses for non-software artifacts.

Open source is a popular choice in scientific research, for reasons we'll explore in the next episode. But before that, let's take a deeper look at what we mean by "open source" and some nuances in the spectrum of open-source software licenses.

The major names in open source

When it comes to defining open-source or free software, there are two major organizations to be aware of. The Free Software Foundation (https://fsf.org) (FSF) was founded in 1985 by Richard Stallman. In addition to advocacy for free software licensing (http://fsf.org/licensing), the FSF also maintains a sizable number of software products, including GNU Emacs and many of the packages at the core of the GNU/Linux operating system.

The Open Source Initiative (http://opensource.org/) (OSI) was founded in 1998 by Eric Raymond and Bruce Perens. The primary mission of the OSI is to assess licenses and maintain a list of those which they judge to qualify as "open source." They also engage in advocacy related to open-source software.

"Free" vs "open source"

It may not be surprising, given their names, that the Free Software Foundation tends to use the term "free software", whereas the Open Source Initiative prefers "open source." Although the FSF uses the term "free" in licensing discussions to refer to the freedom to do certain things with the software, the term often gets conflated with "free" as in no cost, which quickly muddles the discussion. Hence, some prefer to use the term "open source" for clarity. You may also see the term "libre" (Spanish for "free") used in place of or together with "free" (i.e., "free/libre") in the context of software.

This lesson generally uses the term "open source."

Defining free software: The four freedoms

The FSF has a concise definition of free software (https://www.gnu.org/philosophy/free-sw.en.html#four-freedoms) that revolves around the freedom to do certain things with the software:

0. The freedom to **run the program** for any purpose.
1. The freedom to **study how the program works**, and **change it** so it does your computing as you wish.
2. The freedom to **redistribute copies** so you can help your neighbor.
3. The freedom to **distribute copies of your modified versions** to others. By doing this you can give the whole community a chance to benefit from your changes.

Note that **access to the source code** is a precondition for freedoms 1-3.

The OSI has a definition of open source (https://opensource.org/osd/) software which is longer but amounts to the same thing for most purposes. The OSI definition includes two requirements that are implicit in the FSF freedom 0, but which are worth noting:

- No discrimination against persons or groups, and
- No discrimination against fields of endeavor.

Permissive vs copyleft open-source licenses

One of the most important distinctions in the spectrum of open-source software licenses is whether they are considered “copyleft” (also called “restrictive”) or “permissive.” These terms have to do with how the license treats derivative works (which we’ll define more thoroughly in a few moments).

A permissive license allows the licensee to distribute derivative works as they see fit. This includes the possibility of relicensing the derivative work under another license, possibly even a proprietary license. Examples of permissive licenses include the Apache License (https://opensource.org/license/apache-2-0/), the BSD License (https://opensource.org/license/bsd-3-clause/), and the MIT License (https://opensource.org/license/mit/).

Copyleft licenses, on the other hand, require that the licensee distribute derivative works under the same license as the original work. The FSF is one of the main proponents of copyleft licenses, and they created two of the most widely used examples: the GNU General Public License (https://opensource.org/license/gpl-3-0/) (GPL) and the GNU Lesser General Public License (https://opensource.org/license/lgpl-3-0/) (LGPL).

What is a derivative work?

Wikipedia tells us that *a derivative work (https://en.wikipedia.org/wiki/Derivative_work) is an expressive creation that includes major copyright-protected elements of a previously created first work.* For software, this amounts to modifications to someone else’s software. So derivative works are extremely common, especially in collaborative software development.

But what about linking to a library? (And does it matter whether the linkage is static or dynamic?) Or software that interacts via pipes? Or software that is used as a component in a coupled multiphyscs application? Are these also modifications to someone else’s software? Opinions differ on such questions. The Free Software Foundation’s GPL license considers everything in a single executable to be a derived work. (The GPL is sometimes referred to as a “viral” license because it “infects” everything that “touches” it.) The key difference between the GPL and the LGPL is that the latter says that linking is *not* considered a derivative work. The FSF refers to the GPL as “strong copyleft” or “strongly protective” (of software freedom) and the LGPL as a weakened version. The definition of derivative work matters less for permissive licenses because they are not so rigidly tied to the license of the original work.

But because of these different approaches to dealing with derivative works, concerns about the “compatibility” of licenses may arise when you are combining software under different licenses. A later episode will explore this in greater detail, but for now, the easiest way to avoid problems with license compatibility is to avoid *distributing* other works with yours. In other words, don’t ship someone else’s software package as part of yours. (Some may consider it a convenience to bundle together all of the dependencies required to build a software package.) Let the end user put them together (i.e., build the executable that combines them).

This is because of an important fact about open source licenses, even strong copyleft licenses: you’re not required to distribute a derived work! The requirement is that *if* you do distribute it, you do so in conformance with the terms of the license of the original work. So you can make changes to a piece of software and you’re not required to share the derived work with anyone else. And you can finesse license compatibility issues by letting the end user put everything together rather than you shipping the combined work.

If you are concerned about the licensing of your dependencies, there are numerous paid and free automated tools to check for license compatibility. However, you are ultimately responsible for ensuring license compatibility and passing a tool check does necessarily mean you have no conflicts (though a failing check should be addressed!).

Activity: Is this an open source license?

The following is a real-world example of a software license (lightly obfuscated to protect the identity of the software). Read it and decide whether it qualifies as "open source."

In order to acquire access to the code sources, the recipient agrees:

1. to compile/use the XYZZY source code AS IS without modification; users however are welcome to request changes, or to contribute modifications subject to approval of the authors;
2. if the copy of the XYZZY downloaded by the authorized user is made available to third parties, to ensure that the user agreement is followed by the third parties;
3. to send a one-time email to xyzzy@example.com describing planned research using that module;
4. prior to publication, to email a draft of the article/letter/note to xyzzy@example.com; and
5. to include in published results or presentations the proper code name(s) and appropriate references.

Hint: focus on the first two clauses.

Solution No. Clauses 1 (especially) violate the freedom of being able to modify the code and the freedom to distribute copies of your modified version of the code to others. And clause 2 requires that if you distribute copies of the *unmodified* original, it is under the same license terms. Why might someone have felt clauses like these were necessary to include in their software license? Perhaps they've had problems in the past with users distributing modified code with errors that they felt reflected poorly on the original code. Or perhaps they want to impose some measure of quality control over modifications. A possible alternative solution would be to include a requirement that derivatives must be clearly distinguished from the original (e.g., different name). Some open source licenses include such clauses.

Discussion

Now take a close look at clauses 3-5 in the license above. What do you think the copyright owner intended to achieve with those clauses?

Would you be inclined to comply with these license terms? Do you think others comply?

Do you think the copyright holder tries to enforce these terms? (If you have to sign the agreement, they know who has the software.) If you were the copyright holder, do you think it would be worth the effort to try to enforce these terms?

Can you think of better ways to achieve the same things?

Comments It seems like clauses 3 and 4, charitably interpreted, are intended to give the copyright owner awareness of how people are using the software. Going back to our speculation about why they might not want anyone to modify the code, perhaps they're implicitly seeking to exert some quality control over work done using the code. If you send them a draft paper, do you think they would let you know if they found a problem with how you had used the code or interpreted the results? Clause 5 is a requirement that the code be cited in work where it is used. This probably seems quite reasonable, on its face – appropriate citation of software should be encouraged. There are other ways to make this request, though they lack the legal force of putting it in the license. The primary alternative is to make the request in a prominent file in the repository. `CITATION` is the conventional name for this file, though some people put it in the `README` file. The Citation File Format (https://citation-file-format.github.io/) (CFF) is a lightly structured YAML schema, designed to be both human- and machine-readable, to indicate your preferred citation for the work. These files are conventionally named `CITATION.cff` and in addition to be being readily visible in your repository can be interpreted by tools like GitHub, Zenodo, and Zotero to automatically display the preferred citation.

Open licensing of non-software artifacts

OSI approves open-source licenses for *software*. But there are many other creative works related to software (or not) that we might want to treat similarly (like documentation for your software, or this lesson). The Creative Commons (https://creativecommons.org/) (CC) is a family of licenses analogous to open-source but for things other than software. Another class of work that can fall under Creative Commons licenses is data. Be aware that if you utilize data with a CC BY-SA license, you may be limited in your in your licensing options as CC BY-SA is similar to a copyleft, requiring derivative works to use the same license. Variants of the Creative Commons license allow you to impose various restrictions, similar to choosing different licenses for software:

- CC BY (Attribution)

- CC BY-SA (Attribution-ShareAlike)
- CC BY-ND (Attribution-NoDerivs)
- CC BY-NC (Attribution-NonCommercial)
- CC BY-NC-SA (Attribution-NonCommercial-ShareAlike)
- CC BY-NC-ND (Attribution-NonCommercial-NoDerivs)

The Attribution clause, which is part of all CC licenses, requires that the user include the appropriate attribution (title, author, source, license) when the work is used. The ShareAlike clause requires that adaptations (derivatives) be shared under the same terms as the original (analogous to copyleft). The NoDerivs clause says that no derivatives of the work are permitted. The NonCommercial clause says that only non-commercial uses of the work are permitted. Without this clause, commercial uses are allowed.

The Creative Commons has developed a set of badges (https://creativecommons.org/about/downloads#badges) and icons (https://creativecommons.org/about/downloads#icons) that provide quick visual indicators of the chosen license. For example, this lesson is licensed under CC BY 4.0:

There is also a "CC0 Public Domain Dedication" which can be used to indicate intent to place the artifact in the public domain. However this does not satisfy the legal requirements in all jurisdictions, so if you're serious about placing a work in the public domain, you might want to investigate further.

Key Points

- The Open Source Initiative (OSI) is considered the arbiter of open-source licenses.
- The four freedoms include: running the software for any purpose, studying and changing the source code, and distributing copies of the original or modified source.
- A permissive license allows derivative works to be licensed differently than the original; a copyleft license requires that the derivative use the same license as the original.
- Creative Commons is a licensing scheme for non-software works that is similar to the open-source spectrum for software.

Why Choose Open Source Licensing?

Overview

Teaching: 18 min
Exercises: 20 min
Questions

- What are some of the reasons for preferring open-source licensing over proprietary?
- Does open-source licensing prevent you from making money off of your software?
- Does open-source licensing guarantee the sustainability of your software?

Objectives

- Understand some of the reasons for preferring open-source licensing over proprietary.
- Understand that the choice of license is a tool for your software and your project goals.

The philosophical reasons

One of the most common reasons that developers of scientific software choose open-source over proprietary licensing is because they consider it to be consistent with the scientific method. The scientific method requires transparency and reproducibility, and in computationally-based science, this implies that the "apparatus" (i.e., the software) be available for others to inspect and understand and that others should be able to use it to reproduce the relevant (computational) experiments.

Another philosophical reason that many cite is that the results of publicly-funded research (e.g., software produced with research funding) should be publicly available.

And, finally, there's the altruistic reason that releasing the software as open source may help others.

Other considerations favoring open source

Even if you're not completely swayed by the philosophical arguments above, there may be other, more practical reasons to lean towards open-source licensing.

One very simple, but often compelling, reason is that the sponsor of your research may require (or encourage) you to release your software products as open source. At this point in time, many (most) U.S. federal research sponsors are encouraging "open science" with policies that explicitly or implicitly encourage open source. Within the U.S. Department of Energy, several programs have adopted a policy (https://science.osti.gov/-/media/ascr/pdf/research/docs/Doe_lab_developed_software_policy.pdf) that prefers open-source release unless there is a reason not to do so.

Another common reason to favor open-source licensing is to facilitate building a community around your software. Understandably, an open and accessible code base is likely to be more attractive and have a lower barrier to entry for potential contributors than closed source. On the other hand, having to complete an explicit license agreement is a barrier to use (or contribution) of closed-source software. At most institutions, only a few people are authorized to sign legal agreements on behalf of the organization. Usually, a license agreement would have to be reviewed and executed by an IP lawyer, which can cause delays. In some cases, the institution and the licensor may be unable to come to an agreement on the terms and it may be impossible to obtain the license.

And, on a related note, if you're using a proprietary license, you have to manage and archive all of the paperwork associated with those licenses so that you know who your licenses are. Some find that this is more trouble than it's worth.

Debunking some arguments *against* open-source

There are also a variety of reasons that some people argue against open-sourcing software, which don't hold up if you dig a little deeper.

Myth: You can't sell open-source software

It is a common misconception that open-sourcing software prevents you from making money off of it. In fact, there are many different business models that are commonly used around software, and nearly all of them are as applicable to open-source as to proprietary software.

Approach	Proprietary	Copyleft	Permissive
Sell the software	yes	yes	yes
"Freemium" or "dual licensing" allows free use by some, paid by others	yes	yes	yes
Relicense to proprietary	n/a	no	yes
Sell convenience, e.g., packaging, installation media, pre-compiled executables	yes	yes	yes
Sell professional services around the software, e.g., training, technical support, consulting	yes	yes	yes
Sell custom development services, e.g., proprietary extensions, accelerated development	yes	yes	yes
Sell software-as-a-service (SaaS)	yes	yes	yes
Sell the research	yes	yes	yes

I don't want others to profit from my open-source software

If you're using a permissive license, someone else can take derivatives proprietary. But, with the wealth of permissively licensed software out there, this is not a common experience. If you're still concerned, you might prefer a copyleft license, which will prevent this scenario.

But there might be other considerations at play, too. For example, what if you *do* want a commercial entity to use your software – for example, for it to be adopted by a computer vendor or distributed in a Linux or similar large distribution of software? This is a way of getting your software broader exposure and broader distribution. Assuming you're not expecting financial compensation, this kind of collaboration becomes much easier with open-source licensing, and more specifically with *permissive* licenses.

Commercial entities prefer permissive licenses

Many commercial entities find copyleft licenses scary. They are concerned about how far the viral nature of copyleft licenses reaches into other parts of their product. Legal opinions on this differ, and that is little or no case law on this yet. Since lawyers tend towards conservative answers, they will often advise their commercial clients to avoid copyleft software. As a result, many companies will not consider working with copyleft software, only permissively licensed software. Some (typically larger) companies consider staff working on copyleft software to be "contaminated" and will not allow them to work on other software.

The software-as-a-service conundrum

"Software-as-a-service" (SaaS) is a popular way of making software products available today. Many SaaS products make extensive use of open-source software. Some developers don't like the possibility that another company can trivially monetize (other people's) software by turning it into a SaaS product. It may compete with the developer's own SaaS offering. And the SaaS provider can keep enhancements proprietary while making the benefits available in the SaaS product.

Use in a SaaS product is not considered distribution of the software per se. But some licenses, such as the GNU Affero General Public License (https://opensource.org/license/agpl-v3/) include "network" clauses that require that the source be made available to remote users of the service. Other ways of addressing these concerns tend to result in licenses that are not open source. In some cases, key modules are changed to proprietary licenses while others remain open.

An article on the Ars Technica (https://arstechnica.com/) website discusses the SaaS conundrum further: In 2019, multiple open source companies changed course—is it the right move? (https://arstechnica.com/information-technology/2019/10/is-the-software-world-taking-too-much-from-the-open-source-community/)

I want to protect my intellectual property

Another concern that people sometimes raise about openly accessible code is that others can use the novel ideas embodied in it to "scoop" them. Proprietary licenses, by their nature, allow you to keep the source code private, so you can avoid this concern. But there are also strategies that you can use with open source to provide functional protection.

First, as we discussed earlier, open source licenses do *not* require that you make derived works public, only that *if* you do, you make the source available. So the basic strategy is not to disclose your novel derived work until you've had a reasonable chance to exploit the results of your work. For example, you might wait until you've published the initial papers about the method and results that might not be obtainable with other methods. Or you might give yourself (or your team) a fixed "exploitation period" (e.g., one year) before publishing the source code. This is similar to a compromise that's often used in academic publishing, where a sponsor wants the publications to be open access, but they allow the publisher a proprietary exploitation period (also often one year) before making the document openly available.

Licensing as a tool

As we've suggested, the licensing of your software should be viewed as a tool to help you pursue your goals for the software and the associated project.

Basically, you want to ask yourself (and your collaborators) what rights you want to grant to others or retain for yourselves:

- Who can use the program?
- Can users see the source code?
- Can users modify the source code?
- Can the users redistribute the original or modified code?

And think about how these choices will affect your project, would-be contributors to the software, and would-be users of the software.

🔔 Discussion

Have you ever been involved in a discussion of proprietary versus open source licensing for a software package?

What arguments were made in favor of proprietary licensing? What arguments were made in favor of open-source?

Was there a particular argument that carried the day, in either direction?

Avoid magical thinking: Open-source is no guarantee of sustainability or community

Open-source is a great tool to help you build a community around your software. But you shouldn't imagine that simply slapping an open source license on your software makes it sustainable. Besides having software that is potentially useful to others, you'll need to work at it if you want to build a community that contributes to and helps support your software. Many open source software projects never receive any outside contributions. In a webinar entitled What I Learned from 20 Years of Leading Open Source Projects (https://ideas-productivity.org/events/hpcbp-056-20yearsopensource), Wolfgang Bangerth, one of the founders of the deal.II package, offers his experience of what it took to build a small single-group software project into a truly community-based resource – and what it takes to keep it going.

Discussion

Does your research community include any truly community-based software packages? Packages which are both widely used *and* widely contributed to?

If you happen to be involved in such a project, what is your role? User? Contributor? Maintainer? What is your experience with the community?

Key Points

- Philosophical reasons to choose open-source licenses include consistency with the scientific method and openness of publicly funded research results. Another reason is that it facilitates building a community around your software.
- Most software-related business models work as well for open-source software as for proprietary.
- Open-source doesn't guarantee that outsiders will engage with your software. You'll need to work to build a community of contributors and users.

Choosing an Open Source License

Overview

Teaching: 13 min
Exercises: 15 min
Questions

- What are some of the reasons for going with an established open-source license instead of creating a new one?
- What are some of the most popular open-source licenses?
- Name a tool that can help with a more detailed understanding of common open-source licenses?

Objectives

- Be able to identify some of the most common open-source licenses.
- Know about a tool that can help you select an open-source license that meets your needs.

Don't reinvent the license

If you want to use an open-source license with your software, the first advice is to use an existing license rather than inventing your own. The OSI has approved more than 80 different licenses as qualifying as open source. They cover a wide range of situations, and with that many options, you're pretty unlikely to have a need that's not already covered. Moreover, the OSI feels that there are too many open-source licenses already, and has been reluctant to review and approve new licenses to control the proliferation.

Another reason to choose an OSI-approved license is that there are some publication venues (e.g., the Journal of Open Source Software (https://joss.theoj.org/) (JOSS)) that will only accept OSI-approved licenses. There is at least one case in which JOSS rejected a submission for a software package that was licensed under an institution-specific variant of the 3-Clause BSD License (https://opensource.org/license/bsd-3-clause/) which was not OSI-approved. While there are other options besides JOSS for publishing your software, it is important to be aware of such restrictions when selecting a license.

Considerations in selecting an open-source license

The most significant decision in open-source is between permissive or copyleft licenses. Technically, this is a decision as to whether derivative works can be changed to a new license or not. But it can have knock-on effects, particularly in the area of license compatibility.

License compatibility comes into play when you start combining software to get your work done. As we discussed earlier, there are different interpretations of what kinds of combinations do or do not result in derived works which, under copyleft licensing, might become subject to the terms of the original work's license. Permissive licenses have fewer compatibility issues.

On a related note, it is worth considering the norms of the community you and your software are engaging with. If, for example, "everyone" in your field uses a particular license, it may be easier for your software to be accepted by others if you follow the same approach – unless, of course, you have strong reasons for doing otherwise.

Another clause that appears in many open-source licenses has to do with the labeling of derived works, requiring that derived works be identified differently than the original. Why would we want this? What if someone took your code, and in modifying it introduced a bug that made all of the results it produced subtly wrong? That could easily give your code a bad name – unless the problem code had a *different* name that enabled the community to easily distinguish them.

Patents in software licenses

A patent is a different form of intellectual property than a creative work like a piece of software. But they are often connected in the software, and increasingly software licenses also include patent-related clauses.

Patents cover an invention that is useful and non-obvious. That invention could be embodied in software. Some people make strong arguments against the idea that inventions embodied in software should be patentable at all. But in the legal sense, they are a reality. If you're using a piece of software (even open source) that is covered by a patent and you don't have a license for the patent, you're infringing. Not being aware of the patent does not excuse the infringement. And you could be sued for monetary damages.

Historically, many open-source licenses were silent on patents – they said nothing at all about them. More recently, since the courts have decided that software inventions are patentable, some open-source licenses have started including patent-related clauses.

The most common type of patent clause grants royalty-free (i.e. no cost) right to use patented content owned by the copyright holder(s) (e.g. Apache 2.0 (https://opensource.org/license/apache-2-0/), GPLv3 (https://opensource.org/license/gpl-3-0/)). (Obviously, the copyright holders can't provide licenses for other people's patents – which is important to remember. It is still possible that a code has (presumably unknowingly) infringed on some other patent.) Another form of patent clause involves retaliation, effectively saying "If you sue me for patent infringement, your license to use this software is terminated", (e.g. Apache 2.0 (https://opensource.org/license/apache-2-0/)). A weak retaliation clause is triggered by an action related to the specific software, whereas a strong retaliation clause is triggered by any patent action against the licensor.

Although it is no longer listed by the OSI, there is also a BSD 3-Clause Clear License (https://choosealicense.com/licenses/bsd-3-clause-clear/) which explicitly states that no patent rights are granted by the license.

Popular OSI-approved licenses

Some of the most widely used OSI-approved licenses are listed below, along with notes as to their permissiveness, compatibility, and what type of patent clause(s) it has. Any license on this list is a good choice because they are among the most popular and well-known open-source licenses.

License	Type	GPL-Compatible	Patent Clause(s)
Apache License, Version 2.0 (https://opensource.org/license/apache-2-0/)	Permissive	v3, not v2	Grant, Weak retaliation
Common Development and Distribution License 1.0 (https://opensource.org/license/cddl-1-0/)	Permissive	No	Grant, Weak retaliation
Eclipse Public License version 2.0 (https://opensource.org/license/epl-2-0/)	Weak Copyleft	Yes	Grant, Weak retaliation
GNU General Public License version 2 (https://opensource.org/license/gpl-2-0/)	Copyleft	Yes	Implied grant

License	Type	GPL-Compatible	Patent Clause(s)
GNU General Public License version 3 (https://opensource.org/license/gpl-3-0/)	Copyleft	Yes	Grant, Weak retaliation
GNU Lesser General Public License version 2.1 (https://opensource.org/license/lgpl-2-1/)	Weak Copyleft	Yes	Implied grant
GNU Lesser General Public License version 3 (https://opensource.org/license/lgpl-3-0/)	Weak Copyleft	Yes	Silent
GNU Library General Public License version 2 (https://opensource.org/license/lgpl-2-0/)	Weak Copyleft	Yes	Implied grant
Mozilla Public License 2.0 (https://opensource.org/license/mpl-2-0/)	Permissive	Yes	Grant, Weak retaliation
The 2-Clause BSD License (https://opensource.org/license/bsd-2-clause/)	Permissive	Yes	Silent
The 3-Clause BSD License (https://opensource.org/license/bsd-3-clause/)	Permissive	Yes	Silent
The MIT License (https://opensource.org/license/mit/)	Permissive	Yes	Silent*

* In Why so little love for the patent grant in the MIT License? (https://opensource.com/article/18/3/patent-grant-mit-license), Scott Peterson argues that the MIT license, which provides the right to "deal with the Software without restriction," includes the right to use associated patents based on the language used.

ChooseALicense.com

If you want more choices for your open-source license or are interested in clauses other than those in the table above, check out ChooseALicense.com (https://choosealicense.com). This tool, which was developed by GitHub and is openly curated through a GitHub repository (https://github.com/github/choosealicense.com) starts with three very simple suggestions:

- Use the license preferred by your community
- If you want a permissive license, they recommend MIT
- If you want a copyleft license, they recommend GPLv3

But then their Licenses (https://choosealicense.com/licenses/) page lists eight licenses that span a broad spectrum and provide analyses of thirteen different characteristics. And their Appendix (https://choosealicense.com/appendix/) has a table of more than forty licenses analyzed in terms of the thirteen different characteristics. The characteristics include:

- Commercial use
- Distribution
- Modification
- Patent use
- Private use
- Disclose source
- License and copyright notice
- License and copyright notice for source
- Network use is distribution
- Same license
- Same license (file)
- Same license (library)
- State changes
- Liability
- Trademark use
- Warranty

By understanding which characteristics are important for how you want to license your software, you can use the table in the ChooseALicense.com appendix (https://choosealicense.com/appendix/) to identify specific licenses that are worth looking at more deeply. Once you have some candidates, you should read each of them carefully – there may be additional clauses that you may or may not want in your license.

And remember, even the list of licenses that ChooseALicense.com has analyzed is less than half of the number of OSI-approved open-source licenses. So don’t give up!

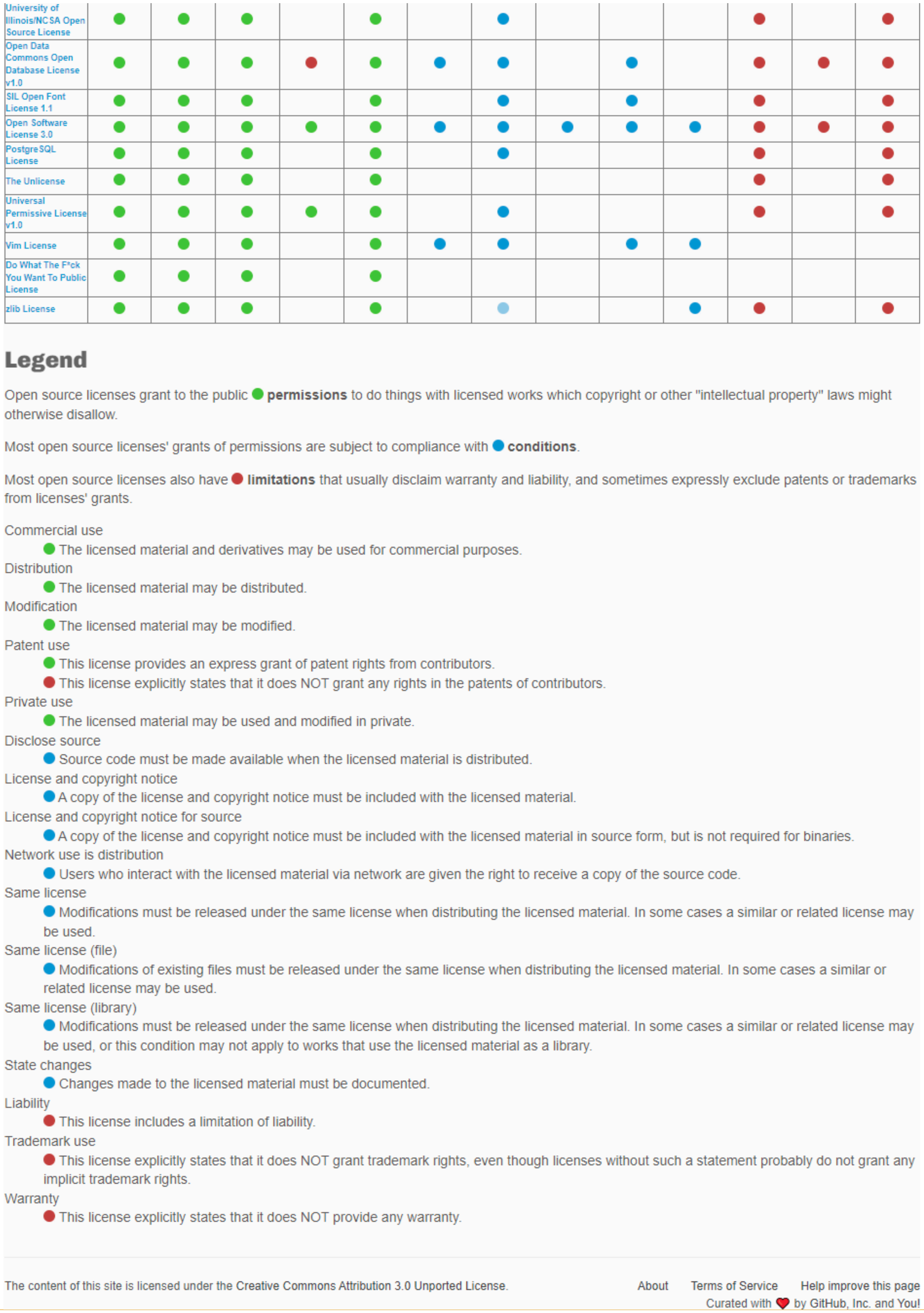
Home

Appendix

For reference, here is a table of every license described in the **choosealicense.com repository**.

If you're here to choose a license, **start from the home page** to see a few licenses that will work for most cases.

License	Commercial use	Distribution	Modification	Patent use	Private use	Disclose source	License and copyright notice	Network use is distribution	Same license	State changes	Liability	Trademark use	Warranty
BSD Zero Clause License	🟢	🟢	🟢		🟢						🔴		🔴
Academic Free License v3.0	🟢	🟢	🟢	🟢	🟢		🔵			🔵	🔴	🔴	🔴
GNU Affero General Public License v3.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵	🔵	🔵	🔵	🔴		🔴
Apache License 2.0	🟢	🟢	🟢	🟢	🟢		🔵			🔵	🔴	🔴	🔴
Artistic License 2.0	🟢	🟢	🟢	🟢	🟢		🔵			🔵	🔴	🔴	🔴
BSD 2-Clause "Simplified" License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
BSD 3-Clause Clear License	🟢	🟢	🟢	🔴	🟢		🔵				🔴		🔴
BSD 3-Clause "New" or "Revised" License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
BSD 4-Clause "Original" or "Old" License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
Boost Software License 1.0	🟢	🟢	🟢		🟢		○				🔴		🔴
Creative Commons Attribution 4.0 International	🟢	🟢	🟢	🔴	🟢		🔵			🔵	🔴	🔴	🔴
Creative Commons Attribution Share Alike 4.0 International	🟢	🟢	🟢	🔴	🟢		🔵		🔵	🔵	🔴	🔴	🔴
Creative Commons Zero v1.0 Universal	🟢	🟢	🟢	🔴	🟢						🔴	🔴	🔴
CeCILL Free Software License Agreement v2.1	🟢	🟢	🟢	🟢	🟢	🔵	🔵	🔵	🔵		🔴		🔴
CERN Open Hardware Licence Version 2 - Permissive	🟢	🟢	🟢	🟢	🟢		🔵			🔵	🔴		🔴
CERN Open Hardware Licence Version 2 - Strongly Reciprocal	🟢	🟢	🟢	🟢	🟢	🔵	🔵		🔵	🔵	🔴		🔴
CERN Open Hardware Licence Version 2 - Weakly Reciprocal	🟢	🟢	🟢	🟢	🟢	🔵	🔵		○	🔵	🔴		🔴
Educational Community License v2.0	🟢	🟢	🟢	🟢	🟢		🔵			🔵	🔴	🔴	🔴
Eclipse Public License 1.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵		🔵		🔴		🔴
Eclipse Public License 2.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵		🔵		🔴		🔴
European Union Public License 1.1	🟢	🟢	🟢	🟢	🟢	🔵	🔵	🔵	🔵	🔵	🔴	🔴	🔴
European Union Public License 1.2	🟢	🟢	🟢	🟢	🟢	🔵	🔵	🔵	🔵	🔵	🔴	🔴	🔴
GNU Free Documentation License v1.3	🟢	🟢	🟢		🟢	🔵	🔵		🔵	🔵	🔴		🔴
GNU General Public License v2.0	🟢	🟢	🟢		🟢	🔵	🔵		🔵	🔵	🔴		🔴
GNU General Public License v3.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵		🔵	🔵	🔴		🔴
ISC License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
GNU Lesser General Public License v2.1	🟢	🟢	🟢		🟢	🔵	🔵		○	🔵	🔴		🔴
GNU Lesser General Public License v3.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵		○	🔵	🔴		🔴
LaTeX Project Public License v1.3c	🟢	🟢	🟢		🟢	🔵	🔵			🔵	🔴		🔴
MIT No Attribution	🟢	🟢	🟢		🟢						🔴		🔴
MIT License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
Mozilla Public License 2.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵		○		🔴	🔴	🔴
Microsoft Public License	🟢	🟢	🟢	🟢	🟢		🔵					🔴	🔴
Microsoft Reciprocal License	🟢	🟢	🟢	🟢	🟢	🔵	🔵		○			🔴	🔴
Mulan Permissive Software License, Version 2	🟢	🟢	🟢	🟢	🟢		🔵				🔴	🔴	🔴



University of Illinois/NCSA Open Source License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
Open Data Commons Open Database License v1.0	🟢	🟢	🟢	🔴	🟢	🔵	🔵		🔵		🔴	🔴	🔴
SIL Open Font License 1.1	🟢	🟢	🟢		🟢		🔵		🔵		🔴		🔴
Open Software License 3.0	🟢	🟢	🟢	🟢	🟢	🔵	🔵	🔵	🔵	🔵	🔴	🔴	🔴
PostgreSQL License	🟢	🟢	🟢		🟢		🔵				🔴		🔴
The Unlicense	🟢	🟢	🟢		🟢						🔴		🔴
Universal Permissive License v1.0	🟢	🟢	🟢	🟢	🟢		🔵				🔴		🔴
Vim License	🟢	🟢	🟢		🟢	🔵	🔵		🔵	🔵			
Do What The F*ck You Want To Public License	🟢	🟢	🟢		🟢								
zlib License	🟢	🟢	🟢		🟢		🔵			🔵	🔴		🔴

Legend

Open source licenses grant to the public 🟢 **permissions** to do things with licensed works which copyright or other "intellectual property" laws might otherwise disallow.

Most open source licenses' grants of permissions are subject to compliance with 🔵 **conditions**.

Most open source licenses also have 🔴 **limitations** that usually disclaim warranty and liability, and sometimes expressly exclude patents or trademarks from licenses' grants.

Commercial use	
	🟢 The licensed material and derivatives may be used for commercial purposes.
Distribution	
	🟢 The licensed material may be distributed.
Modification	
	🟢 The licensed material may be modified.
Patent use	
	🟢 This license provides an express grant of patent rights from contributors.
	🔴 This license explicitly states that it does NOT grant any rights in the patents of contributors.
Private use	
	🟢 The licensed material may be used and modified in private.
Disclose source	
	🔵 Source code must be made available when the licensed material is distributed.
License and copyright notice	
	🔵 A copy of the license and copyright notice must be included with the licensed material.
License and copyright notice for source	
	🔵 A copy of the license and copyright notice must be included with the licensed material in source form, but is not required for binaries.
Network use is distribution	
	🔵 Users who interact with the licensed material via network are given the right to receive a copy of the source code.
Same license	
	🔵 Modifications must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used.
Same license (file)	
	🔵 Modifications of existing files must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used.
Same license (library)	
	🔵 Modifications must be released under the same license when distributing the licensed material. In some cases a similar or related license may be used, or this condition may not apply to works that use the licensed material as a library.
State changes	
	🔵 Changes made to the licensed material must be documented.
Liability	
	🔴 This license includes a limitation of liability.
Trademark use	
	🔴 This license explicitly states that it does NOT grant trademark rights, even though licenses without such a statement probably do not grant any implicit trademark rights.
Warranty	
	🔴 This license explicitly states that it does NOT provide any warranty.

✎ Activity: Open-source licenses in your community

Try to identify 2-3 open-source software packages within your community that use different licenses.

Which licenses do they use? Or does a single license strongly dominate your community? Are they permissive or copyleft? In what other ways do they differ?

(Hint: the https://choosealicense.com/appendix/ (https://choosealicense.com/appendix/) page might be helpful.)

Key Points

- There are many OSI-approved licenses already available covering most needs. Some publications or other venues require OSI-approved licenses.
- The variants of the GNU GPL license are among the most popular copyleft licenses, while Apache, BSD, and MIT are among the most popular permissive licenses.
- ChooseALicense.com (https://choosealicense.com) has analyses of more than 40 open-source licenses along 13 different characteristics.

Documenting Your Choice of License

Overview

Teaching: 10 min
Exercises: 10 min
Questions

- What are the two basic strategies for documenting your choice of license?
- What information should you include in each file in your software?

Objectives

- Understand the importance of marking your software with your chosen license and copyright information.

So you've chosen a license for your software. Now you need to ensure that people are aware of it! This is particularly important for open-source software because you won't have the interaction of someone having to sign and return or otherwise indicate their acceptance of the terms that you would have with a proprietary license.

Two strategies for documenting your license

There are, in essence, two strategies for indicating your choice of license. The first is to put it in a file at the repository level. The second is to put it inside the individual files. The centralized approach has the advantage of simplicity and maintainability. However, if an individual file is separated from the distribution or repository, the recipient won't see the copyright and license information if the notice only appears in a central file.

The Software Freedom Law Center (https://softwarefreedom.org/)'s (SFLC's) whitepaper on Managing copyright information within a free software project (https://softwarefreedom.org/resources/2012/ManagingCopyrightInformation.html) suggests that the best practice is to do *both*.

Discussion

Have you ever received a file by itself, outside of the context of a version control repository or complete distribution of the package, for example as a potential solution to a problem or a bug? Was the origin of the file and its copyright and licensing evident to you? Or perhaps the person who gave it to you told you about the license and copyright terms?

Did that file (or parts of it) end up in another software package that you were working on at the time?

Centralized license and copyright information

You should place the complete copyright information together with the text of the license you've chosen in a prominent location in the main directory of your repository. In the past `COPYING` used to be a popular recommendation for this file, but `LICENSE` seems like a more obvious choice and is probably more commonly used these days.

If your package is more complicated, with multiple licenses, they can often naturally be grouped into subdirectories with consistent licensing and each subdirectory can include an appropriate `LICENSE` file. If the licensing structure is sufficiently complex, it may be worth placing a "roadmap" to the various licenses applying to different parts of the code in the top directory.

Tracking authorship and copyright information

Every person who makes a non-trivial contribution to a software package has a copyright interest in that package. (There's no legal definition for what constitutes a non-trivial contribution. The package maintainers need to determine that on a case by case basis. Fixing a typo, or even perhaps a simple bug fix may not be considered substantive. But a complex bug fix or implementing new functionality probably would be.) Such a list can get rather long and could change frequently as new contributors join. (Though if many of the contributors are performing work for hire and their employers actually own the copyright, the list of rightsholders may not be so long after all.) But it is important to maintain this information to the best of your ability to ensure that contributors get the credit they deserve and contributors can be identified if legal issues arise.

If used carefully, version control systems provide a good means to track authorship. But you need to use the version control tools in such a way that maintains the file histories as files are moved, renamed, etc. In other words, instead of changing the name of a file by `git rm` ing the old name and `git add` ing the new name, use `git mv` so that the history (of commits and the authorship of those commits) follows the file through the name change.

But this authoritative information about authorship is only available in the version control repository, using your version control tool. If the package is bundled up and distributed as a tarball, or in some other form outside of the repository, this information may be inaccessible to the recipient. The same is true for individual files which might be distributed outside of the package for various reasons.

So the recommendation is to construct a copyright notice for the entire repository (as opposed to for each individual file), and try to do a reasonable job of keeping it up to date. The most likely place for the copyright notice to live is in your `LICENSE` file because the license normally includes a copyright notice. But another option, if you prefer, could be a separate `COPYRIGHT` or `AUTHORS` file. Note, however, that the copyright holders are not necessarily the authors of the code, depending on whether the authors or their employers are the rightsholders.

File-scope license and copyright information

As a point of reference, the recommendation of the FSF is to include the following in the header (beginning) of *every* file in the package:

1. one sentence naming and briefly describing the program,
2. the copyright notice of the authors,
3. the name(s) of the license(s) under which the software is available,
4. a brief warranty disclaimer, and
5. a URL pointing to the full copy of the license.

Others recommend including the full text of the license rather than just the name and a URL. This is a lot of information to insert into every file and a lot of information to maintain. Note that the contributors to each individual file are likely to be different, so in principle, each file could have a *different* copyright notice, each of which would need to be maintained. All of which seems a little overboard for most purposes.

The SFLC's suggestion is to boil the per-file header down to the essentials. You want enough information that if the file was distributed separately from the rest of the repository, the recipient could identify the origins of the file and know where to look for the remaining details. Something along the following lines:

Code

```
Copyright 2012 The Foo Project Developers. See the LICENSE file at the top-level directory of this distribution and at http://www.example.co
m/foo/LICENSE.

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 distributi
on and at http://www.example.com/foo/LICENSE. No part of Foo Project, including this file, may be copied, modified, propagated, or distribut
ed except according to the terms contained in the LICENSE file.
```

Consider writing scripts to help you insert and maintain the file-scope copyright and license headers you decide upon.

Discussion

Is there any software that you work with directly, or in your community, which you know has a license associated with it but is not marked in at least one of the two ways we've discussed here (centralized and file-scope)?

Badges

Badges at the top of `README.md` files are a popular way to summarize a variety of information about the software package. Such badges often include testing status and other dynamic information. Licenses are pretty static, so it may be more and so more fun than functional, but badges are available to reflect many popular licenses.

The badge generation site https://shields.io (https://shields.io) can automatically render a badge for any license that GitHub recognizes by simply referencing the repository as follows:

Code

```
![GitHub](https://img.shields.io/github/license/:user/:repo)
```

for example, `![GitHub](https://img.shields.io/github/license/hpc-simtools/ips-framework)` which renders as

license BSD-3-Clause

The site also provides many license badges which can be selected explicitly, such as the badge for this lesson: `[![License: CC BY 4.0](https://img.shields.io/badge/License-CC_BY_4.0-lightgrey.svg)](https://creativecommons.org/licenses/by/4.0/)` which renders as

License CC BY 4.0

(https://creativecommons.org/licenses/by/4.0/)

While the shields.io (https://shields.io) site lists many licenses directly, a developer named Lukas Himsel (https://gist.github.com/lukas-h) has posted a Gist (https://gist.github.com/lukas-h/2a5d00690736b4c3a7ba) which provides badges for even more licenses.

Key Points

- License and copyright information can be documented in a centralized manner (at the repository level) and within individual files.
- Individual files should include enough information to identify that they are copyrighted and licensed and point the recipient to the details.

Collaboration and Licensing

Overview

Teaching: 17 min
Exercises: 15 min
Questions

- What are the concerns with accepting code from collaborators?
- What mechanisms are there to ensure collaborators agree to license terms?
- What concerns are there with using code from online forums?
- Why are LLMs challenging for copyright and licensing?

Objectives

- Understand the challenges surrounding code contributed from outside the project.

If you are lucky, you have an open-source project with an active user base and a few developers to help field issues. As a project grows, it's possible you will get pull requests from people you've never met. Even if the code looks great, it's possible the has author inadvertently has caused a licensing issue.

A somewhat related topic is how to handle code snippets from external sources like web forums or large language models.

Contributors and Licenses

There are two cases where a contributor can cause problems with licensing your code.

First, if you have a copyleft license, contributors from industry may be reluctant to contribute over fear of accidentally violating the terms of the GPL license. Since GPL requires derivative works to be copyleft, if the contributor were to incorporate some of the code into a larger project, they would be obligated to release the entire code base under the license terms.

Alternatively, if you have a restrictive license, it's possible a contribution may not be subject to copyright at all! Any work from a government employee, say a researcher at a federal agency, is not subject to copyright and is in public domain.

It is important to communicate your license terms to all collaborators and decide on a license early in a project's life cycle. Changing a license is possible, but may require the explicit approval of all contributors; for larger, older projects the prospect is daunting.

Part of being proactive is developing and publishing your `CONTRIBUTING` guidelines. You can also choose to use a **Contributor License Agreement** (CLA), which is a legal document that new contributors must sign prior to merging their code. While protective of the software project, CLAs may limit inclusivity by acting as a barrier to first-time contributors. They can also create a power imbalance between maintainers and contributors. Practically, CLAs often require review and approval by the legal department of the contributor.

A **Developer Certificate of Origin** (DCO) is a lighter-weight agreement that allows contributors to confirm the code they commit is suitable for the project license. They can be integrated into a pull request or commit message instead of a separate legal document. In either case, the policies for contributors should be clear and easy to find and verify if legal issues do arise.

Activity

What are the contributor guidelines for some open source code you use?

Pick one dependency or utility you regularly use. Does the project have a CONTRIBUTING file? CLA? DCO? Does the license mention contributors?

Code from Internet Forums

Question and answer forums like Stack Overflow provide a valuable avenue for developers to connect with knowledgeable users covering a range of topics from installation, language usage, and debugging. It is satisfying to find the exact answer to your problem so you can get back to work. But as with anything else you see on the internet, the material is subject to copyright. It is worth thinking about how you can incorporate solutions found online into your work while respecting their copyrights. Material on Stack Overflow, for example, is published under a CC-BY-SA license (Creative Commons Attribution Sharealike (https://creativecommons.org/licenses/by-sa/4.0/)).

Generally, if you're using the Stack Overflow material as guidance or documentation, and adapting it to your particular situation without using text (code) verbatim from the Stack Overflow posting, it should be relatively straightforward because you're not making direct use of the copyrighted material. You don't have to disclose where the original idea came from, but for your own benefit it is a good idea to include the URL in a comment. Its also nice to give credit where credit is due. An example of this kind of use might be if you are searching for how to plot a scatter plot with transparency given by another column in your dataframe. An answer may suggest matplotlib, seaborn, or another plotting library. It may provide example code or command line instructions to make such a plot. But you adapt it, with your variables and filenames, and other details. And in the end, there's probably little from the original post appearing in your code – maybe routine or command names and a few key options.

On the other hand, if you are searching for, say, a binary search written in python, or the answer contains a function snippet for performing the task you need? Fair use does not use length as a factor, if you directly copy and paste code and you want to distribute that code, it would then fall under the license applicable to the forum posting. As mentioned, for Stack Overflow, for example, that's CC-BY-SA.

Pop Quiz

If you use CC-BY-SA work in your project, what kind of license should you use?

Solution Though they are considered appropriate for software *documentation*, the Creative Commons recommend against using their licenses for software per se (see Can I apply a Creative Commons license to software? (https://creativecommons.org/faq/#can-i-apply-a-creative-commons-license-to-software)). But they do define a concept of "compatibility" between some CC licenses and software licenses precisely for situations like this (see Compatible Licenses (https://creativecommons.org/share-your-work/licensing-considerations/compatible-licenses/)). According to this page, CC-BY-SA 4.0 (not prior versions) is compatible with the Free Art License (http://artlibre.org/licence/lal/en/) or the GPLv3 (https://www.gnu.org/copyleft/gpl.html) licenses.

Generative AI, Large Language Models, and Code Assistants

A recent development for software engineers is the rise of large language models (LLMs) capable of producing code from English descriptions. The utilities are integrated in many search engines, integrated development environments (IDEs), or as standalone assistants. While performance and utility can vary wildly, LLMs can increase developer productivity by removing some of the tedious jobs. Just be wary, correct-seeming code can be worse than something clearly wrong!

Just as the technological limits of LLMs are still being discovered, the legal aspect of AI, copyright, and licensing are actively being determined in court. There are three phases where licensing and copyright concerns appear in utilizing LLMs:

1. Does the training process respect the licenses of the code used for training?
2. If you want to refine an LLM, what license is the model distributed with?
3. If I use code generated by an LLM, what attribution does it need and will it affect my license?

Ingested Code

Unless you are training your own LLM, this is more of an interesting case study in copyright than a day-to-day concern. Since the implementation details of many LLMs are proprietary, you may not know what code a model was trained on, what license the code used, or if permission was attained to use the software for training.

Legal challenges against AI companies have been brought up by artists and authors, who allege the generation of verbatim passages or replication of artistic style indicates that the training data included copyrighted material against the creators' wishes. Code generation hasn't been included in these lawsuits so far, but rulings on other domains could affect how LLMs are trained. If you are concerned about the origin of code used to train an LLM, look for LLMs that provide information on the training set and use a training set aligned with your license and values.

Refining an LLM

In a technical sense, refining a published network is fairly straight forward! Just like any other piece of software, you can follow the license distributed with the material in creating new works derived from the original network. However, the weights of the foundation model could "contain" copyrighted material from data on which they were trained. It's possible that during refinement, the network will retain the copyright material in a form that can be recovered. Litigation will be needed to sort out fair use, but keep in mind that refined networks may contain the foundation model largely unchanged.

Using Code from LLMs

You may have already used code from an LLM, either to play with a new technology or even in production to save time on development. Continuing with the theme of this section, we don't know all the answers until legislation and lawsuits settle.

Consider the following scenario: you use an LLM to generate a function that is later discovered to be verbatim from a copyrighted code base, violating the license. Who's liable for damages? You? The company that trained the network? The company you pay to use the LLM? Legally, we don't yet have answers to this, but getting the code from an LLM may not be considered a defense against the fact that your code infringes on someone else's copyright.

Some tools have been developed to scan LLM output to flag large snippets that are present in other sources. Many tech companies don't allow code generation from outside products due to privacy concerns, as well as licensing issues. As an aside, if you're interested in a job in industry, you might want to make sure your coding skills are solid *without* help from an LLM.

The US government recently declared that AI-generated work can not be copyrighted if it's produced without human intervention beyond prompt engineering. This is likely to apply to a work as a whole instead of snippets, e.g. if your project has a few generated functions it could still have a copyright. If instead you instruct an LLM to "make a game" (and it's able to do so), that could would not be copyrighted. As an example, "Zaraya of the Dawn" (https://www.copyright.gov/docs/zarya-of-the-dawn.pdf) is a comic book where the images were produced by Midjourney. While the text and layout were human-generated and therefore subject to copyright, the US Copyright Office found the images could not be copyrighted. Extending this to software, if you have AI produce all of your UI, that portion of your code might not be copyrightable.

If you are developing code you intend to monetize, the safest advice currently would be to avoid LLM-generated code altogether. Were it brought to light in discovery, AI-generated code could open the door to damages over the entire code base. Otherwise, treat LLM output like code from an internet forum, you can use it for information and to point you towards the code you need, but don't copy its entire output. If you do copy and paste code, you may also want to mark in the source code what is LLM-derived. Doing this consistently could safeguard parts of your project that may resemble proprietary code by chance.

Key Points

- Collaborators may be restricted in their ability to contribute to open source projects (e.g. industrial partners) or unable to copyright their work (government employees).
- You can include a Contributor License Agreement (CLA) to ensure collaborators agree to license terms prior to committing code.
- Stackoverflow content is licensed as CC BY-SA, which is incompatible with permissive or proprietary licenses.
- License and copyright around LLM-generated content is actively being litigated.