Open Source Software

What is free software?

- “Free software” means software that respects users' freedom and community.
- **the users have the freedom to run, copy, distribute, study, change and improve the software**.
- “free software” is a matter of liberty, not price.
- To understand the concept, you should think of “free” as in “free speech,” not as in “free beer”.
- Sometimes call it “libre software,” borrowing the French or Spanish word for “free” as in freedom, to show we do not mean the software is gratis.

The four essential freedoms

A program is free software if the program's users have the four essential freedoms:

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

A program is free software if it gives users adequately all of these freedoms. Otherwise, it is nonfree.

Freedom 0:
Run the program as you wish

- The freedom to run the program means the freedom for any kind of person or organization to use it on any kind of computer system, for any kind of overall job and purpose, without being required to communicate about it with the developer or any other specific entity.
- It is the *user's* purpose that matters, not the *developer's* purpose; you as a user are free to run the program for your purposes, and if you distribute it to someone else, she is then free to run it for her purposes, but you are not entitled to impose your purposes on her.

Freedom 0:
Run the program as you wish…

- The freedom to run the program as you wish means that you are not forbidden or stopped from making it run. This has nothing to do with what functionality the program has, whether it is technically capable of functioning in any given environment, or whether it is useful for any particular computing activity.
- For example, if the code arbitrarily rejects certain meaningful inputs—or even fails unconditionally—that may make the program less useful, perhaps even totally useless, but it does not deny users the freedom to run the program, so it does not conflict with freedom 0. If the program is free, the users can overcome the loss of usefulness, because freedoms 1 and 3 permit users and communities to make and distribute modified versions without the arbitrary nuisance code.

Freedom 1:
Study the source code and make changes

- Freedom 1 includes the freedom to use your changed version in place of the original. If the program is delivered in a product designed to run someone else's modified versions but refuse to run yours — a practice known as “tivoization” or “lockdown”, or (in its practitioners' perverse terminology) as “secure boot” — freedom 1 becomes an empty pretense rather than a practical reality. These binaries are not free software even if the source code they are compiled from is free.
- One important way to modify a program is by merging in available free subroutines and modules. If the program's license says that you cannot merge in a suitably licensed existing module — for instance, if it requires you to be the copyright holder of any code you add — then the license is too restrictive to qualify as free.
- Whether a change constitutes an improvement is a subjective matter. If your right to modify a program is limited, in substance, to changes that someone else considers an improvement, that program is not free.

Freedom 2 and 3:
To redistribute if you wish: basic requirements

- Freedom to distribute (freedoms 2 and 3) means you are free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission to do so.
- You should also have the freedom to make modifications and use them privately in your own work or play, without even mentioning that they exist. If you do publish your changes, you should not be required to notify anyone in particular, or in any particular way.
- **Freedom 3 includes the freedom to release your modified versions as free software.** A free license may also permit other ways of releasing them; in other words, it does not have to be a copyleft license. However, a license that requires modified versions to be nonfree does not qualify as a free license.

Freedom 2 and 3:
To redistribute if you wish: basic requirements…

- The freedom to redistribute copies must include binary or executable forms of the program, as well as source code, for both modified and unmodified versions. (Distributing programs in runnable form is necessary for conveniently installable free operating systems.) It is OK if there is no way to produce a binary or executable form for a certain program (since some languages don't support that feature), but you must have the freedom to redistribute such forms should you find or develop a way to make them.

Rules about packaging and distribution details

- It is acceptable for the license to require that you change the name of the modified **version, remove a logo,** or **identify your modifications as yours**.
- As long as these requirements are not so burdensome that they effectively hamper you from releasing your changes, they are acceptable; you're already making other changes to the program, so you won't have trouble making a few more.

Copyleft or restrictive

- Copyleft is a general method for making a program (or other work) free (in the sense of freedom, not “zero price”), and requiring all modified and extended versions of the program to be free as well.
- Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it.
- Copyleft guarantees that every user has freedom.
- Copyleft also provides an incentive for other programmers to add to free software. Important free programs such as the GNU C++ compiler exist only because of this.

Noncopylefted free software

- Noncopylefted free software comes from the author with permission to redistribute and modify, and also to add additional restrictions to it.
- If a program is free but not copylefted, then some copies or modified versions may not be free at all. A software company can compile the program, with or without modifications, and distribute the executable file as a proprietary software product.
- Example: The X Window System.
  - The X Consortium released X11 with distribution terms that made it noncopylefted free software, and subsequent developers have mostly followed the same practice.
  - A copy which has those distribution terms is free software.
  - However, there are nonfree versions as well, and there are (or at least were) popular workstations and PC graphics boards for which nonfree versions are the only ones that work.
  - If you are using this hardware, X11 is not free software for you. The developers of X11 even made X11 nonfree for a while; they were able to do this because others had contributed their code under the same noncopyleft license.

Copyright

- Copyright refers to the legal right of the owner of intellectual property.
- In simpler terms, copyright is the right to copy. This means that the original creators of products and anyone they give authorization to are the only ones with the exclusive right to reproduce the work.
- Copyright law gives creators of original material the exclusive right to further use and duplicate that material for a given amount of time, at which point the copyrighted item becomes public domain.

Permissive software license

- A **permissive software license**, sometimes also called **BSD-like** or **BSD-style** license, is a free-software license with only minimal restrictions on how the software can be used, modified, and redistributed, usually including a warranty disclaimer. Examples include the GNU All-permissive License, MIT License, BSD licenses, Apple Public Source License and Apache license. As of 2016, the most popular free-software license is the permissive MIT license.

- **Lax permissive licensed software**

  Lax permissive licenses include the X11 license and the two BSD licenses. These licenses permit almost any use of the code, including distributing proprietary binaries with or without changing the source code.

Proprietary Software

- Only software owner have full legal access to the source code.
- Trusted partners can be granted inspection rights if they sign a non-disclosure agreement (NDA) ⊔ closed source.
- Software owners may or may not be the authors of the code.
- End users must accepts a license restricting their rights
- Such licenses generally:
  - Restrict user's rights of re-distribution
  - Prohibit trying to reconstruct source code or use inside another product
  - Indemnify the product from damages sue to either malfunction or misuse
- The difference with proprietary and free software has nothing to do with price
- The license differences have to do with redistribution, modification, reuse of code, etc.

Freeware

- *“Not to be confused with Free software or Free and open-source software”*
- **Freeware** is software, most often proprietary, that is distributed at no monetary cost to the end user.
- There is no agreed-upon set of rights, license, or EULA(**end-user license agreement**) that defines *freeware* unambiguously; every publisher defines its own rules for the freeware it offers.
- For instance, modification, redistribution by third parties, and reverse engineering are permitted by some publishers but prohibited by others.
- The source code for freeware is typically not made available. Freeware may be intended to benefit its producer , for example, encouraging sales of a more capable version.

Shareware

- Shareware software is a software that are *freely distributed to users on trial basis*. There is a *time limit* inbuilt in the software(for example- free for 30 days or 2 months). As the time limit gets over, it will be deactivated. To use it after time limit, you have to pay for the software.
- Users prefer shareware because of following reasons –
  - Available free of cost
  - helps to know about the product before buying it
  - Some examples of freeware software are –
  - Adobe acrobat 8 professional
  - PHP Debugger 2.1.3.3
  - Winzip
  - Getright

Shareware…

- **Shareware are of following types:**
- **Adware** – Contains ads to generate revenue for the developers
- **Donationware** – payment is optional
- **Nagware** – reminds user to purchase the license or the software
- **Freemium** – free for non-premium but of cost for premium features
- **Demoware** – demonstration version. It is further classified as crippleware and trialware.
  - **Crippleware** – Some features are disabled under time-limit
  - **Trialware** – all features are available under time-limit

Private software

- Private or custom software is software developed for one user (typically an organization or company). That user keeps it and uses it, and does not release it to the public either as source code or as binaries.
- A private program is free software (in a somewhat trivial sense) if its sole user has the four freedoms. In particular, if the user has full rights to the private program, the program is free. However, if the user distributes copies to others and does not provide the four freedoms with those copies, those copies are not free software.
- In general we do not believe it is wrong to develop a program and not release it. There are occasions when a program is so important that one might argue that withholding it from the public is doing wrong to humanity. However, such cases are rare. Most programs are not that important, and declining to release them is not particularly wrong. Thus, there is no conflict between the development of private or custom software and the principles of the free software movement.
- Nearly all employment for programmers is in development of custom software; therefore most programming jobs are, or could be, done in a way compatible with the free software movement.

Commercial software

- “Commercial” and “proprietary” are not the same! Commercial software is software developed by a business as part of its business. Most commercial software is proprietary, but there is commercial free software, and there is noncommercial nonfree software.
- For example, GNU Ada is developed by a company. It is always distributed under the terms of the GNU GPL, and every copy is free software; but its developers sell support contracts.

Open source software

- Open source software is software with source code that is publicly available under a license that gives users the right to study, change, and distribute the software as they wish.
- The term was coined in 1998 when a group of individuals pushed for title that was less ambiguous than free software.
- They changed the emphasis from freedom to security, cost savings, transparency, and other pragmatic benefits.
- The term is more palatable for the corporate world, even though it refers largely to the same software.
- While the GPL isn’t the only supported license, it remains one of the most prominent.
- Like free software, open source software can be distributed for free, but it doesn’t have to be.

The Open Source Initiative (OSI)

- The non-profit organization that supports the development of open source software, asserts that any open source software must adhere to the following criteria:
  1. Free Redistribution
  2. Source Code
  3. Derived Works
  4. Integrity of The Author's Source Code
  5. No Discrimination Against Persons or Groups
  6. No Discrimination Against Fields of Endeavor
  7. Distribution of License
  8. License Must Not Be Specific to a Product
  9. License Must Not Restrict Other Software
  10. License Must Be Technology-Neutral

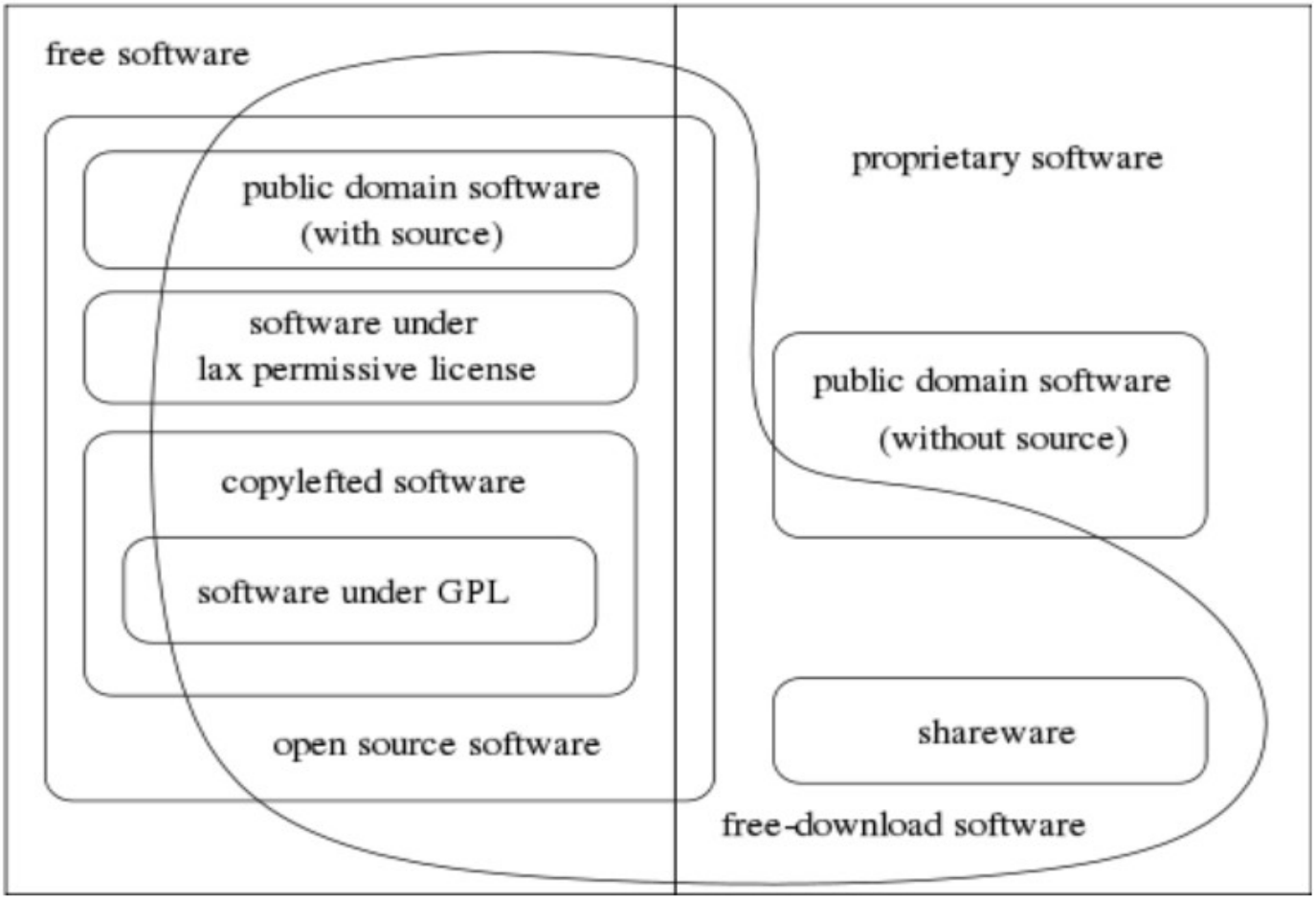
Free Software Vs Open Source Software

- **Free software movement was aimed at advocating the availability of free source codes and removal of copyrights.** Also, whatever codes are derived from a free software license should also be released as free software. This posed a major problem in the commercialization of products.
- OSS which was initially introduced to remove the confusion caused by the ”free” term took up the cause of removing the restrictions placed on commercialization.
- Codes under open source license can be used and modified, like free software, but the modified by-products do not have to be under the open source license. This helped in the commercialization of products produced using open source software. But since that was against what free software stood for, it separated the two into two different concepts.
- Thus free software was a movement that emphasized on the moral aspect of the availability of free codes and promoted intellectual freedom. But OSS was a practical solution to allow the business use of free codes. So, in a way, open source codes can be used for free software but the reverse causes a conflict.

Free and open-source software (FOSS)

- "Free and open-source software" (FOSS) is an umbrella term for software that is simultaneously considered both Free software and open-source software.
- FOSS (free and open-source software) allows the user to inspect the source code and provides a high level of control of the software's functions compared to proprietary software.
- Although there is almost a complete overlap between free-software licenses and open-source-software licenses, there is a strong philosophical disagreement between the advocates of these two positions.
- The terminology of FOSS or "Free and Open-source software" was created to be a **neutral on these philosophical disagreements between the FSF and OSI** and have a single unified term that could refer to both concepts.
- There are a number of related terms and abbreviations for free and open-source software (FOSS or F/OSS), or free/libre and open-source software (FLOSS or F/LOSS—FLOSS is the FSF-preferred term).

Categories of free and nonfree software



History

- In the 1950 and 1960
  - Almost all software was produced by academics and corporate researchers working in collaboration, often shared as public-domain software.
  - It was generally distributed under the principles of openness and cooperation, and was not seen as a commodity in itself.
  - Source code, the human-readable form of software, was generally distributed with the software machine code because users frequently modified the software themselves, because it would not run on different hardware or OS without modification, and also to fix bugs or add new functions.
- In 1969 the Advanced Research Projects Agency Network (ARPANET), a transcontinental, high-speed computer network was constructed. The network (later succeeded by the Internet) simplified the exchange of software code.
- In the early 1970s AT&T distributed early versions of Unix at no cost to government and academic researchers, but these versions did not come with permission to redistribute or to distribute modified versions, and were thus not free software in the modern meaning of the phrase.

History…

- In 1983, **Richard Stallman** launched the GNU Project to write a complete operating system free from constraints on use of its source code.
- A longer version called the GNU Manifesto was published in March 1985. It has been translated into several other languages.
- The **Free Software Foundation** was founded in October 1985, initially to raise funds to help develop GNU.
- Then Linux, a Unix-like kernel, was developed by **Linus Torvalds** in 1991 and made free software in 1992.
- The GNU project's lack of a kernel meant that no complete free-software operating systems existed. The development of Torvalds' kernel closed that last gap. The combination of the almost-finished GNU operating system and the Linux kernel made the first complete free-software operating system.

History…

- In the mid to late 90s,
  - When many web-based companies were starting up, free software became a popular choice for web servers.
  - **Apache HTTP Server(1996)** became the most-used web-server software – a title that still holds as of 2012.
  - Systems based on a common "stack" of software with the Linux kernel at the base, Apache providing web services, the **MySQL(1995)** database engine for data storage, and the **PHP(1995)** programming language for providing dynamic pages, came to be known as **LAMP** systems.

What is Governance model?

- Any project needs organization to achieve its purpose
- How decisions get made and who makes them requires careful thought
- A project is still an open source one, whether or not:
  - Anyone can contribute or only a selected few
  - Decisions are made democratically by an authority
  - Plans and discussions are made public before releases
- How this is done determines the **Governance Model** for a project

What is Governance model?

- Open source projects usually operate according to rules, customs, and processes that determine which contributors have the authority to perform certain tasks.
- Understanding those rules can increase your chances of contributing successfully and positively to a project.
- In open source software projects, the rules and customs that define who gets to do what (and how they are supposed to do it) is called a project’s "governance model.“
- Understanding a project's governance model can help you make a successful, positive contribution to a project.

Governance model:"**Do-ocracy**"

- Open source projects adopting the do-ocracy governance model tend to forgo formal and elaborate governance conventions and instead insist that decisions are made by those who do the work.
- Members of a do-ocracy gain authority by making the most consistent contributions.
- Peer review remains common under this model, but individual contributors tend to retain de facto decision-making power over project components on which they have worked most closely.
- Joining them can be difficult and intimidating for newcomers, as would-be contributors might not immediately know how to participate or seek approval for their contributions.

Governance model: **Founder-leader**

- The individual or group who started the project also administers the project, establishes its vision, controls all permissions to merge code into it, and assumes the right to speak for it in public.
- Some projects refer to their founder-leaders as *Benevolent dictators for life*.
- Lines of power and authority are typically quite clear. They radiate from founder-leaders, who are the final decision-makers for all project matters.
- limitations become apparent as a project grows to a certain size.

Governance model: **Self-appointing council or board**

- Members of an open source project may appoint a number of leadership groups to govern various aspects of a project.
- These groups construct their own decision-making conventions and succession procedures.
- This model is useful in cases where a project does not have a sponsoring foundation and establishing electoral mechanisms is prohibitively difficult.
- Limitation become apparent when self-appointing governing groups grow insular and unrepresentative of the entire project community—as member-selection processes tend to spawn self-reinforcing leadership cultures.

Governance model: **Electoral**

- Conduct governance through electoral processes.
- They may hold elections for various roles, or hold votes to ratify or update project policies and procedures.
- Communities establish and document electoral procedures to which they all agree, then enact those procedures as a regular matter of decision-making.
- But elections also have drawbacks. They can become contentious, distracting, and time-consuming for all project members—whether or not those members are running.

Governance model : Single-vendor or Corporate-backed

- The governing organization usually does not accept contributions from anyone outside it.
- Individual companies or industry consortia may choose to distribute software under the terms of an open source license as a way of reaching potential developers and users—even if they do not accept project contributions from those audiences.
- They might do this to accelerate adoption of their work, spur development activity atop a software platform, support a plugin ecosystem, or avoid the overhead required for cultivating an external developer community.
- This model becomes problematic in cases where a project claims to have an open community but is in fact wholly owned by a company or consortium.

Governance model : **Foundation-backed**

- To exert greater control over resources and project code, some open source projects choose to be managed by an incorporated NGO (non-government organization), such as a charitable nonprofit or trade association.
- This approach allows the project to take ownership of resources like servers, trademarks, patents, and insurance policies.
- Extensive funding and legal requirements normally limit this model to larger open source projects.
- High overhead—not strictly financial, but particularly in terms of contributor time, which can be substantial—is a significant drawback of the foundation-backed governance model.

10 things you should know about open source before you use it

1. It's not just for Linux
2. It's not always free
3. It may or may not have support
4. You have full access to the source code
5. Open source is not just for programmers
6. You aren't breaking any laws by adopting open source
7. You don't have to be an expert to use it
8. Most open source software is as reliable as its proprietary counterpart
9. Freeware and shareware are not the same as open source
10. You're probably already using it

https://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-open-source-before-you-use-it/

Advantage of open source software

1. Community
2. The power of the crowd
3. Transparency
4. Reliability
5. Better security
6. Merit-based
7. Faster time to market
8. Cost effective
9. Freedom from lock-in
10. Becoming the norm
11. long-term viability

https://opensource.com/article/17/8/enterprise-open-source-advantages

Myths about Open Source Software

1. IF SOFTWARE COSTS NOTHING, IT'S NO GOOD
2. OSS IS INFERIOR TO PROPRIETARY SOFTWARE
3. OSS IS PIRACY (OR AT LEAST ENCOURAGES IT)
4. OSS HAS NO SUPPORT
5. OSS IS ONLY FOR DEVELOPERS
6. USING OSS MEANS WORKING FROM THE COMMAND LINE
7. OSS IS ONLY GOOD FOR SMALL PROJECTS
8. OSS IS UNABLE TO DEVELOP GAMES
9. HAVING THE CODE FREELY AVAILABLE MAKES IT LESS SECURE THAN PROPRIETARY CODE
10. OSS IS UNABLE TO INNOVATE
11. SINCE THE LICENSE PLACES RESTRICTIONS ON THE USERS, OSS ISN'T REALLY FREE
12. OSS IS ALL ABOUT PRICE

https://www.datamation.com/osrc/article.php/3771441/Twelve-Myths-about-Free-and-Open-Source-Software.htm

References

1. **Free Software Foundation.**
2. **www.geeksforgeeks.org**
3. **wikipedia.org**
4. **coursera.org**
5. **https://opensource.org/**
6. **dzone.com**
7. **redhat.com**
8. **opensource.com**