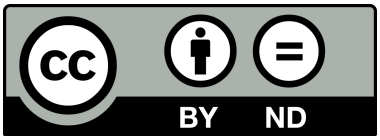
# All of GNU in One Book!

Featuring Richard Stallman
& the Free Software
Foundation!

Curated & Compiled by:
Salem Warde

DISCLAIMER: This book is NOT yet endorsed/sponsored by the Richard Stallman or the Free Software Foundation!





# Chapter I: What is Free Software?

"Free software" means software that respects users' freedom and community. Roughly, it means that **the users have the freedom to run, copy, distribute, study, change and improve the software**. Thus, "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." We 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.
You may have paid money to get copies of a free program, or you may have obtained copies at no charge. But regardless of how you got your copies, you always have the freedom to copy and change the software, even to sell copies[1].

We campaign for these freedoms because everyone deserves them. With these freedoms, the users (both individually and collectively) control the program and what it does for them. When users don't control the program, we call it a "nonfree" or "proprietary" program. The nonfree program controls the users, and the developer controls the program; this makes the program an instrument of unjust power[2].

"Open source" is something different: it has a very different philosophy based on different values. Its practical definition is different too, but nearly all open source programs are in fact free. We explain the difference in Why "Open Source" misses the point of Free Software[3].

## The Free Software Definition

The free software definition presents the criteria for whether a particular software program qualifies as free software. From time to time we revise this definition, to clarify it or to resolve questions about subtle issues. See the History section[4] below for a list of changes that affect the definition of free software.

### The four essential freedoms

A program is free software if the program's users have the four essential freedoms: [1][5]

- The freedom to run the program as you wish, for any purpose (freedom 0).
- 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.
- The freedom to redistribute copies so you can help others (freedom 2).
- 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. While we can distinguish various nonfree distribution schemes in terms of how far they fall short of being free, we consider them all equally unethical.

In any given scenario, these freedoms must apply to whatever code we plan to make use of, or lead others to make use of. For instance, consider a program A which automatically launches a program B to handle some cases. If we plan to distribute A as it stands, that implies users will need B, so we need to judge whether both A and B are free. However, if we plan to modify A so that it doesn't use B, only A needs to be free; B is not pertinent to that plan.

### Free software *can* be commercial

"Free software" does not mean "noncommercial." On the contrary, a free program must be available for commercial use, commercial development, and commercial distribution. This policy is of fundamental importance—without this, free software could not achieve its aims.

We want to invite everyone to use the GNU system, including businesses and their workers. That requires allowing commercial use. We hope that free replacement programs will supplant comparable proprietary programs, but they can't do that if businesses are forbidden to use them. We want commercial products that contain software to include the GNU system, and that would constitute commercial distribution for a price. Commercial development of free software is no longer unusual; such free commercial software is very important. Paid, professional support for free software fills an important need.

Thus, to exclude commercial use, commercial development or commercial distribution would hobble the free software community and obstruct its path to success. We must conclude that a program licensed with such restrictions does not qualify as free software.

A free program must offer the four freedoms to any would-be user that obtains a copy of the software, who has complied thus far with the conditions of the free license covering the software in any previous distribution of it. Putting some of the freedoms off limits to some users, or requiring that users pay, in money or in kind, to exercise them, is tantamount to not granting the freedoms in question, and thus renders the program nonfree.

## Clarifying the Boundary Between Free and Nonfree

In the rest of this article we explain more precisely how far the various freedoms need to extend, on various issues, in order for a program to be free.

### The freedom to 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. In this freedom, 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 other people, they are then free to run it for their purposes, but you are not entitled to impose your purposes on them.

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.

"As you wish" includes, optionally, "not at all" if that is what you wish. So there is no need for a separate "freedom not to run a program."

**The freedom to study the source code and make changes**

In order for freedoms 1 and 3 (the freedom to make changes and the freedom to publish the changed versions) to be meaningful, you need to have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software. Obfuscated "source code" is not real source code and does not count as source code.

Source code is defined as the preferred form of the program for making changes in. Thus, whatever form a developer changes to develop the program is the source code of that developer's version.

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.

One special case of freedom 1 is to delete the program's code so it returns after doing nothing, or make it invoke some other program. Thus, freedom 1 includes the “freedom to delete the program.”

### The freedom 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[6]. 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[7] license. However, a license that requires modified versions to be nonfree does not qualify as a free license.

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.

### Copyleft

Certain kinds of rules about the manner of distributing free software are acceptable, when they don't conflict with the central freedoms. For example, copyleft (very simply stated) is the rule that when redistributing the program, you cannot add restrictions to deny other people the central freedoms. This rule does not conflict with the central freedoms; rather it protects them.

In the GNU project, we use copyleft to protect the four freedoms legally for everyone. We believe there are important reasons why it is better to use copyleft[8]. However, noncopylefted free software[9] is ethical

too. See Categories of Free Software[10] for a description of how “free software,” “copylefted software” and other categories of software relate to each other.

### Rules about packaging and distribution details

Rules about how to package a modified version are acceptable, if they don't substantively limit your freedom to release modified versions, or your freedom to make and use modified versions privately. Thus, 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.

Rules that “if you make your version available in this way, you must make it available in that way also” can be acceptable too, on the same condition. An example of such an acceptable rule is one saying that if you have distributed a modified version and a previous developer asks for a copy of it, you must send one. (Note that such a rule still leaves you the choice of whether to distribute your version at all.) Rules that require release of source code to the users for versions that you put into public use are also acceptable.

A special issue arises when a license requires changing the name by which the program will be invoked from other programs. That effectively hampers you from releasing your changed version so that it can replace the original when invoked by those other programs. This sort of requirement is acceptable only if there's a suitable aliasing facility that allows you to specify the original program's name as an alias for the modified version.

### Export regulations

Sometimes government export control regulations and trade sanctions can constrain your freedom to distribute copies of programs internationally. Software developers do not have the power to eliminate or override these restrictions, but what they can and must do is refuse to impose them as conditions of use of the program. In this way, the restrictions will not affect activities and people outside the jurisdictions of these governments. Thus, free software licenses must not require obedience to any nontrivial export regulations as a condition of exercising any of the essential freedoms.

Merely mentioning the existence of export regulations, without making them a condition of the license itself, is acceptable since it does not restrict users. If an export regulation is actually trivial for free software, then requiring it as a condition is not an actual problem; however, it is a potential problem, since a later change in export law could make the requirement nontrivial and thus render the software nonfree.

### Legal considerations

In order for these freedoms to be real, they must be permanent and irrevocable as long as you do nothing wrong; if the developer of the software has the power to revoke the license, or retroactively add restrictions to its terms, without your doing anything wrong to give cause, the software is not free.

A free license may not require compliance with the license of a nonfree program. Thus, for instance, if a license requires you to comply with the licenses of "all the programs you use," in the case of a user that runs nonfree programs this would require compliance with the licenses of those nonfree programs; that makes the license nonfree.

It is acceptable for a free license to specify which jurisdiction's law applies, or where litigation must be done, or both.

### Contract-based licenses

Most free software licenses are based on copyright, and there are limits on what kinds of requirements can be imposed through copyright. If a copyright-based license respects freedom in the ways described above, it is unlikely to have some other sort of problem that we never anticipated (though this does happen occasionally). However, some free software licenses are based on contracts, and contracts can impose a much larger range of possible restrictions. That means there are many possible ways such a license could be unacceptably restrictive and nonfree.

We can't possibly list all the ways that might happen. If a contract-based license restricts the user in an unusual way that copyright-based licenses cannot, and which isn't mentioned here as legitimate, we will have to think about it, and we will probably conclude it is nonfree.

## The Free Software Definition in Practice

### How we interpret these criteria

Note that criteria such as those stated in this free software definition require careful thought for their interpretation. To decide whether a specific software license qualifies as a free software license, we judge it based on these criteria to determine whether it fits their spirit as well as the precise words. If a license includes unconscionable restrictions, we reject it, even if we did not anticipate the issue in these criteria. Sometimes a license requirement raises an issue that calls for extensive thought, including discussions with a lawyer, before we can decide if the requirement is acceptable. When we reach a conclusion about a new issue, we often update these criteria to make it easier to see why certain licenses do or don't qualify.

### Get help with free licenses

If you are interested in whether a specific license qualifies as a free software license, see our list of licenses[11]. If the license you are concerned with is not listed there, you can ask us about it by sending us email at <licensing@gnu.org>.

If you are contemplating writing a new license, please contact the Free Software Foundation first by writing to that address. The proliferation of different free software licenses means increased work for users in understanding the licenses; we may be able to help you find an existing free software license that meets your needs.

If that isn't possible, if you really need a new license, with our help you can ensure that the license really is a free software license and avoid various practical problems.

### Use the right words when talking about free software

When talking about free software, it is best to avoid using terms like "give away" or "for free," because those terms imply that the issue is about price, not freedom. Some common terms such as "piracy" embody opinions we hope you won't endorse. See Confusing Words and Phrases that are Worth Avoiding[12] for a discussion of these terms. We also have a list of proper translations of "free software"[13] into various languages.

Another group uses the term "open source" to mean something close (but not identical) to "free software." We prefer the term "free software" because, once you have heard that it refers to freedom rather than price, it calls to mind freedom. The word "open" never refers to freedom.

## Beyond Software

Software manuals must be free[14], for the same reasons that software must be free, and because the manuals are in effect part of the software.

The same arguments also make sense for other kinds of works of practical use—that is to say, works that embody useful knowledge, such as educational works and reference works. Wikipedia[15] is the best-known example.

Any kind of work *can* be free, and the definition of free software has been extended to a definition of free cultural works[16] applicable to any kind of works.

## History

From time to time we revise this Free Software Definition. Here is the list of substantive changes, along with links to show exactly what was changed.

- Version 1.169: Explain more clearly why the four freedoms must apply to commercial activity. Explain why the four freedoms imply the freedom not to run the program and the freedom to delete it, so there is no need to state those as separate requirements.
- Version 1.165: Clarify that arbitrary annoyances in the code do not negate freedom 0, and that freedoms 1 and 3 enable users to remove them.
- Version 1.153: Clarify that freedom to run the program means nothing stops you from making it run.
- Version 1.141: Clarify which code needs to be free.
- Version 1.135: Say each time that freedom 0 is the freedom to run the program as you wish.
- Version 1.134: Freedom 0 is not a matter of the program's functionality.
- Version 1.131: A free license may not require compliance with a nonfree license of another program.
- Version 1.129: State explicitly that choice of law and choice of forum specifications are allowed. (This was always our policy.)
- Version 1.122: An export control requirement is a real problem if the requirement is nontrivial; otherwise it is only a potential problem.
- Version 1.118: Clarification: the issue is limits on your right to modify, not on what modifications you have made. And modifications are not limited to “improvements”
- Version 1.111: Clarify 1.77 by saying that only retroactive *restrictions* are unacceptable. The copyright holders can always grant additional *permission* for use of the work by releasing the work in another way in parallel.
- Version 1.105: Reflect, in the brief statement of freedom 1, the point (already stated in version 1.80) that it includes really using your modified version for your computing.
- Version 1.92: Clarify that obfuscated code does not qualify as source code.
- Version 1.90: Clarify that freedom 3 means the right to distribute copies of your own modified or improved version, not a right to participate in someone else's development project.
- Version 1.89: Freedom 3 includes the right to release modified versions as free software.
- Version 1.80: Freedom 1 must be practical, not just theoretical; i.e., no tivoization.
- Version 1.77: Clarify that all retroactive changes to the license are unacceptable, even if it's not described as a complete replacement.
- Version 1.74: Four clarifications of points not explicit enough, or stated in some places but not reflected everywhere:
    - “Improvements” does not mean the license can substantively limit what kinds of modified versions you can release. Freedom 3 includes distributing modified versions, not just changes.
    - The right to merge in existing modules refers to those that are suitably licensed.
    - Explicitly state the conclusion of the point about export controls.

    - Imposing a license change constitutes revoking the old license.
- Version 1.57: Add “Beyond Software” section.
- Version 1.46: Clarify whose purpose is significant in the freedom to run the program for any purpose.
- Version 1.41: Clarify wording about contract-based licenses.
- Version 1.40: Explain that a free license must allow to you use other available free software to create your modifications.
- Version 1.39: Note that it is acceptable for a license to require you to provide source for versions of the software you put into public use.
- Version 1.31: Note that it is acceptable for a license to require you to identify yourself as the author of modifications. Other minor clarifications throughout the text.
- Version 1.23: Address potential problems related to contract-based licenses.
- Version 1.16: Explain why distribution of binaries is important.
- Version 1.11: Note that a free license may require you to send a copy of versions you distribute to previous developers on request.

There are gaps in the version numbers shown above because there are other changes in this page that do not affect the definition or its interpretations. For instance, the list does not include changes in asides, formatting, spelling, punctuation, or other parts of the page. You can review the complete list of changes to the page through the cvsweb interface[17].

## Footnote

1. The reason they are numbered 0, 1, 2 and 3 is historical. Around 1990 there were three freedoms, numbered 1, 2 and 3. Then we realized that the freedom to run the program needed to be mentioned explicitly. It was clearly more basic than the other three, so it properly should precede them. Rather than renumber the others, we made it freedom 0.

- END OF CHAPTER

# Chapter II: Free Software Is Even More Important Now

*by Richard Stallman*

Since 1983, the Free Software Movement has campaigned for computer users' freedom—for users to control the software they use, rather than vice versa. When a program respects users' freedom and community, we call it “free software.”

We also sometimes call it “libre software” to emphasize that we're talking about liberty, not price. Some proprietary (nonfree) programs, such as Photoshop, are very expensive; others, such as the Uber app, are available gratis—but that's a minor detail. Either way, they give the program's developer power over the users, power that no one should have.

Those two nonfree programs have something else in common: they are both *malware*. That is, both have functionalities designed to mistreat the user. Proprietary software nowadays is often malware because the developers' power corrupts them[18]. That directory lists around 650 different malicious functionalities (as of March 2025), but it is surely just the tip of the iceberg.

With free software, the users control the program, both individually and collectively. So they control what their computers do (assuming those computers are loyal[19] and do what the users' programs tell them to do).

With proprietary software, the program controls the users, and some other entity (the developer or “owner”) controls the program. So the proprietary program gives its developer power over its users. That is unjust in itself; moreover, it tempts the developer to mistreat the users in other ways.

Even when proprietary software isn't downright malicious, its developers have an incentive to make it addictive, controlling and manipulative[20]. You can say, as does the author of that article, that the developers have an ethical obligation not to do that, but generally they follow their interests. If you want this not to happen, make sure the program is controlled by its users.

Freedom means having control over your own life. If you use a program to carry out activities in your life, your freedom depends on your having control over the program. You deserve to have control over the programs you use, and all the more so when you use them for something important in your life.

Users' control over the program requires four essential freedoms[21].

(0) The freedom to run the program as you wish, for whatever purpose.

(1) The freedom to study the program's "source code," and change it, so the program does your computing as you wish. Programs are written by programmers in a programming language—like English combined with algebra—and that form of the program is the "source code." Anyone who knows programming, and has the program in source code form, can read the source code, understand its functioning, and change it too. When all you get is the executable form, a series of numbers that are efficient for the computer to run but extremely hard for a human being to understand, understanding and changing the program in that form are forbiddingly hard.

(2) The freedom to make and distribute exact copies when you wish. (It is not an obligation; doing this is your choice. If the program is free, that doesn't mean someone has an obligation to offer you a copy, or that you have an obligation to offer him a copy. Distributing a program to users without freedom mistreats them; however, choosing not to distribute the program—using it privately—does not mistreat anyone.)

(3) The freedom to make and distribute copies of your modified versions, when you wish.

The first two freedoms mean each user can exercise individual control over the program. With the other two freedoms, any group of users can together exercise *collective control* over the program. With all four freedoms, the users fully control the program. If any of them is missing or inadequate, the program is proprietary (nonfree), and unjust.

Other kinds of works are also used for practical activities, including recipes for cooking, educational works such as textbooks, reference works such as dictionaries and encyclopedias, fonts for displaying paragraphs of text, circuit diagrams for hardware for people to build, and patterns for making useful (not merely decorative) objects with a 3D printer. Since these are not software, the free software movement strictly speaking doesn't cover them; but the same reasoning applies and leads to the same conclusion: these works should carry the four freedoms.

A free program allows you to tinker with it to make it do what you want (or cease to do something you dislike). Tinkering with software may sound ridiculous if you are accustomed to proprietary software as a sealed box, but in the Free World it's a common thing to do, and a good way to learn programming. Even the traditional American pastime of tinkering with cars is obstructed because cars now contain nonfree software.

## The Injustice of Proprietariness

If the users don't control the program, the program controls the users. With proprietary software, there is always some entity, the developer or "owner" of the program, that controls the program—and through it, exercises power over its users. A nonfree program is a yoke, an instrument of unjust power.

In outrageous cases (though this outrage has become quite usual) proprietary programs are designed to spy on the users, restrict them, censor them, and abuse them. For instance, the operating system of Apple iThings[22] does all of these, and so does Windows on mobile devices with ARM chips. Windows, mobile phone firmware, and Google Chrome for Windows include a universal back door that allows some company to change the program remotely without asking permission. The Amazon Kindle has a back door that can erase books.

The use of nonfree software in the "internet of things" would turn it into the "internet of telemarketers"[23] as well as the "internet of snoopers."

With the goal of ending the injustice of nonfree software, the free software movement develops free programs so users can free themselves. We began in 1984 by developing the free operating system GNU[24]. Today, millions of computers run GNU, mainly in the GNU/Linux combination[25].

Distributing a program to users without freedom mistreats those users; however, choosing not to distribute the program does not mistreat anyone. If you write a program and use it privately, that does no wrong to others. (You do miss an opportunity to do good, but that's not the same as doing wrong.) Thus, when we say all software must be free, we mean that every copy must come with the four freedoms, but we don't mean that someone has an obligation to offer you a copy.

## Nonfree Software and SaaSS

Nonfree software was the first way for companies to take control of people's computing. Nowadays, there is another way, called Service as a Software Substitute, or SaaSS. That means letting someone else's server do your own computing tasks.

SaaSS doesn't mean the programs on the server are nonfree (though they often are). Rather, using SaaSS causes the same injustices as using a nonfree program: they are two paths to the same bad place. Take the example of a SaaSS translation service: The user sends text to the server, and the server translates it (from English to Spanish, say) and sends the translation back to the user. Now the job of translating is under the control of the server operator rather than the user.

If you use SaaSS, the server operator controls your computing. It requires entrusting all the pertinent data to the server operator, which will be forced to show it to the state as well—who does that server really serve, after all?[26]

## Primary And Secondary Injustices

When you use proprietary programs or SaaSS, first of all you do wrong to yourself, because it gives some entity unjust power over you. For your own sake, you should escape. It also wrongs others if you

make a promise not to share. It is evil to keep such a promise, and a lesser evil to break it; to be truly upright, you should not make the promise at all.

There are cases where using nonfree software puts pressure directly on others to do likewise. Skype is a clear example: when one person uses the nonfree Skype client software, it requires another person to use that software too—thus both surrender their freedom. (Google Hangouts have the same problem.) It is wrong even to suggest using such programs. We should refuse to use them even briefly, even on someone else's computer.

Another harm of using nonfree programs and SaaSS is that it rewards the perpetrator, encouraging further development of that program or "service," leading in turn to even more people falling under the company's thumb.

All the forms of indirect harm are magnified when the user is a public entity or a school.

## Free Software and the State

Public agencies exist for the people, not for themselves. When they do computing, they do it for the people. They have a duty to maintain full control over that computing so that they can assure it is done properly for the people. (This constitutes the computational sovereignty of the state.) They must never allow control over the state's computing to fall into private hands.

To maintain control of the people's computing, public agencies must not do it with proprietary software (software under the control of an entity other than the state). And they must not entrust it to a service programmed and run by an entity other than the state, since this would be SaaSS.

Proprietary software has no security at all in one crucial case—against its developer. And the developer may help others attack. Microsoft shows Windows bugs to the NSA[27] (the US government digital spying agency) before fixing them. We do not know whether Apple does likewise, but it is under the same government pressure as Microsoft. If the government of any other country uses such software, it endangers national security. Do you want the NSA to break into your government's computers? See our suggested policies for governments to promote free software[28].

## Free Software and Education

Schools (and this includes all educational activities) influence the future of society through what they teach. They should teach exclusively free software, so as to use their influence for the good. To teach a proprietary program is to implant dependence, which goes against the mission of education. By training in use of free software, schools will direct society's future towards freedom, and help talented programmers master the craft.

They will also teach students the habit of cooperating, helping other people. Each class should have this rule: “Students, this class is a place where we share our knowledge. If you bring software to class, you may not keep it for yourself. Rather, you must share copies with the rest of the class—including the program's source code, in case someone else wants to learn. Therefore, bringing proprietary software to class is not permitted except to reverse engineer it.”

Proprietary developers would have us punish students who are good enough at heart to share software and thwart those curious enough to want to change it. This means a bad education. See more discussion about the use of free software in schools[29].

## Free Software: More Than “Advantages”

I'm often asked to describe the “advantages” of free software. But the word “advantages” is too weak when it comes to freedom. Life without freedom is oppression, and that applies to computing as well as every other activity in our lives. We must refuse to give the developers of the programs or computing services control over the computing we do. This is the right thing to do, for selfish reasons; but not solely for selfish reasons.

Freedom includes the freedom to cooperate with others. Denying people that freedom means keeping them divided, which is the start of a scheme to oppress them. In the free software community, we are very much aware of the importance of the freedom to cooperate because our work consists of organized cooperation. If your friend comes to visit and sees you use a program, she might ask for a copy. A program which stops you from redistributing it, or says you're “not supposed to,” is antisocial.

In computing, cooperation includes redistributing exact copies of a program to other users. It also includes distributing your changed versions to them. Free software encourages these forms of cooperation, while proprietary software forbids them. It forbids redistribution of copies, and by denying users the source code, it blocks them from making changes. SaaSS has the same effects: if your computing is done over the web in someone else's server, by someone else's copy of a program, you can't see it or touch the software that does your computing, so you can't redistribute it or change it.

## Conclusion

We deserve to have control of our own computing. How can we win this control?

- By rejecting nonfree software on the computers we own or regularly use, and rejecting SaaSS.
- By developing free software[30] (for those of us who are programmers.)
- By refusing to develop or promote nonfree software or SaaSS.
- By spreading these ideas to others[31].
- By saying no and stating our reasons[32] when we are invited to run a nonfree program.

We and thousands of users have done this since 1984, which is how we now have the free GNU/Linux operating system that anyone—programmer or not—can use. Join our cause, as a programmer or an activist. Let's make all computer users free.

- END OF CHAPTER

# Chapter III: Selling Free Software

Many people believe that the spirit of the GNU Project is that you should not charge money for distributing copies of software, or that you should charge as little as possible—just enough to cover the cost. This is a misunderstanding.

Actually, we encourage people who redistribute free software[33] to charge as much as they wish or can. If a license does not permit users to make copies and sell them, it is a nonfree license. If this seems surprising to you, please read on.

The word "free" has two legitimate general meanings; it can refer either to freedom or to price. When we speak of "free software," we're talking about freedom, not price. (Think of "free speech," not "free beer.") Specifically, it means that a user is free to run the program, study and change the program, and redistribute the program with or without changes.

Free programs are sometimes distributed gratis, and sometimes for a substantial price. Often the same program is available in both ways from different places. The program is free regardless of the price, because users have freedom in using it.

Nonfree programs[34] are usually sold for a high price, but sometimes a store will give you a copy at no charge. That doesn't make it free software, though. Price or no price, the program is nonfree because its users are denied freedom.

Since free software is not a matter of price, a low price doesn't make the software free, or even closer to free. So if you are redistributing copies of free software, you might as well charge a substantial fee and *make some money*. Redistributing free software is a good and legitimate activity; if you do it, you might as well make a profit from it.

Free software is a community project, and everyone who depends on it ought to look for ways to contribute to building the community. For a distributor, the way to do this is to give a part of the profit to free software development projects or to the Free Software Foundation[35]. This way you can advance the world of free software.

## Distributing free software is an opportunity to raise funds for development. Don't waste it!

In order to contribute funds, you need to have some extra. If you charge too low a fee, you won't have anything to spare to support development.

## Will a higher distribution price hurt some users?

People sometimes worry that a high distribution fee will put free software out of range for users who don't have a lot of money. With proprietary software, a high price does exactly that—but free software is different.

The difference is that free software naturally tends to spread around, and there are many ways to get it.

Software hoarders try their damnedest to stop you from running a proprietary program without paying the standard price. If this price is high, that does make it hard for some users to use the program.

With free software, users don't *have* to pay the distribution fee in order to use the software. They can copy the program from a friend who has a copy, or with the help of a friend who has network access. Or several users can join together, split the price of one CD-ROM, then each in turn can install the software. A high CD-ROM price is not a major obstacle when the software is free.

## Will a higher distribution price discourage use of free software?

Another common concern is for the popularity of free software. People think that a high price for distribution would reduce the number of users, or that a low price is likely to encourage users.

This is true for proprietary software—but free software is different. With so many ways to get copies, the price of distribution service has less effect on popularity.

In the long run, how many people use free software is determined mainly by *how much free software can do*, and how easy it is to use. Many users do not make freedom their priority; they may continue to use proprietary software if free software can't do all the jobs they want done. Thus, if we want to increase the number of users in the long run, we should above all *develop more free software*.

The most direct way to do this is by writing needed free software[36] or manuals[37] yourself. But if you do distribution rather than writing, the best way you can help is by raising funds for others to write them.

## The term "selling software" can be confusing too

Strictly speaking, "selling" means trading goods for money. Selling a copy of a free program is legitimate, and we encourage it.

However, when people think of "selling software[38]," they usually imagine doing it the way most companies do it: making the software proprietary rather than free.

So unless you're going to draw distinctions carefully, the way this article does, we suggest it is better to avoid using the term “selling software” and choose some other wording instead. For example, you could say “distributing free software for a fee”—that is unambiguous.

## High or low fees, and the GNU GPL

Except for one special situation, the GNU General Public License[39] (GNU GPL) has no requirements about how much you can charge for distributing a copy of free software. You can charge nothing, a penny, a dollar, or a billion dollars. It's up to you, and the marketplace, so don't complain to us if nobody wants to pay a billion dollars for a copy.

The one exception is in the case where binaries are distributed without the corresponding complete source code. Those who do this are required by the GNU GPL to provide source code on subsequent request. Without a limit on the fee for the source code, they would be able set a fee too large for anyone to pay—such as a billion dollars—and thus pretend to release source code while in truth concealing it. So in this case we have to limit the fee[40] for source in order to ensure the user's freedom. In ordinary situations, however, there is no such justification for limiting distribution fees, so we do not limit them.

Sometimes companies whose activities cross the line stated in the GNU GPL plead for permission, saying that they “won't charge money for the GNU software” or such like. That won't get them anywhere with us. Free software is about freedom, and enforcing the GPL is defending freedom. When we defend users' freedom, we are not distracted by side issues such as how much of a distribution fee is charged. Freedom is the issue, the whole issue, and the only issue.

- END OF CHAPTER

# Chapter IV: Why programs must not limit the freedom to run them

*by Richard Stallman*

Free software means software controlled by its users, rather than the reverse. Specifically, it means the software comes with four essential freedoms that software users deserve. At the head of the list is freedom 0, the freedom to run the program as you wish, in order to do what you wish.

Some developers propose to place usage restrictions in software licenses to ban using the program for certain purposes, but that would be a disastrous path. This article explains why freedom 0 must not be limited. Conditions to limit the use of a program would achieve little of their aims, but could wreck the free software community.

First of all, let's be clear what freedom 0 means. It means that the distribution of the software does not restrict how you use it. This doesn't make you exempt from laws. For instance, fraud is a crime in the US—a law which I think is right and proper. Whatever the free software license says, using a free program to carry out your fraud won't shield you from prosecution.

A license condition against fraud would be superfluous in a country where fraud is a crime. But why not a condition against using it for torture, a practice that states frequently condone when carried out by the “security forces”?

A condition against torture would not work, because enforcement of any free software license is done through the state. A state that wants to carry out torture will ignore the license. When victims of US torture try suing the US government, courts dismiss the cases on the grounds that their treatment is a national security secret. If a software developer tried to sue the US government for using a program for torture against the conditions of its license, that suit would be dismissed too. In general, states are clever at making legal excuses for whatever terrible things they want to do. Businesses with powerful lobbies can do it too.

What if the condition were against some specialized private activity? For instance, PETA proposed a license that would forbid use of the software to cause pain to animals with a spinal column. Or there might be a condition against using a certain program to make or publish drawings of Mohammad. Or against its use in experiments with embryonic stem cells. Or against using it to make unauthorized copies of musical recordings.

It is not clear these would be enforcible. Free software licenses are based on copyright law, and trying to impose usage conditions that way is stretching what copyright law permits, stretching it in a dangerous way. Would you like books to carry license conditions about how you can use the information in them?

What if such conditions are legally enforcible—would that be good?

The fact is, people have very different ethical ideas about the activities that might be done using software. I happen to think those four unusual activities are legitimate and should not be forbidden. In particular I support the use of software for medical experiments on animals, and for processing meat. I defend the human rights of animal right activists but I don't agree with them; I would not want PETA to get its way in restricting the use of software.

Since I am not a pacifist, I would also disagree with a “no military use” provision. I condemn wars of aggression but I don't condemn fighting back. In fact, I have supported efforts to convince various armies to switch to free software, since they can check it for back doors and surveillance features that could imperil national security.

Since I am not against business in general, I would oppose a restriction against commercial use. A system that we could use only for recreation, hobbies and school is off limits to much of what we do with computers.

I've stated above some parts of my views about certain political issues unrelated to the issue of free software—about which of those activities are or aren't unjust. Your views about them might differ, and that's precisely the point. If we accepted programs with usage restrictions as part of a free operating system such as GNU, people would come up with lots of different usage restrictions. There would be programs banned for use in meat processing, programs banned only for pigs, programs banned only for cows, and programs limited to kosher foods. Someone who hates spinach might license a program to allow use for processing any vegetable except spinach, while a Popeye fan's program might allow only use for spinach. There would be music programs allowed only for rap music, and others allowed only for classical music.

The result would be a system that you could not count on for any purpose. For each task you wish to do, you'd have to check lots of licenses to see which parts of your system are off limits for that task. Not only for the components you explicitly use, but also for the hundreds of components that they link with, invoke, or communicate with.

How would users respond to that? I think most of them would use proprietary systems. Allowing usage restrictions in free software would mainly push users towards nonfree software. Trying to stop users from doing something through usage restrictions in free software is as ineffective as pushing on an object through a long, straight, soft piece of cooked spaghetti. As one wag put it, this is “someone

with a very small hammer seeing every problem as a nail, and not even acknowledging that the nail is far too big for the hammer."

It is worse than ineffective; it is wrong too, because software developers should not exercise such power over what users do. Imagine selling pens with conditions about what you can write with them; that would be noisome, and we should not stand for it. Likewise for general software. If you make something that is generally useful, like a pen, people will use it to write all sorts of things, even horrible things such as orders to torture a dissident; but you must not have the power to control people's activities through their pens. It is the same for a text editor, compiler or kernel.

You do have an opportunity to determine what your software can be used for: when you decide what functionality to implement. You can write programs that lend themselves mainly to uses you think are positive, and you have no obligation to write any features that might lend themselves particularly to activities you disapprove of.

The conclusion is clear: a program must not restrict what jobs its users do with it. Freedom 0 must be complete. We need to stop torture, but we can't do it through software licenses. The proper job of software licenses is to establish and protect users' freedom.

- END OF CHAPTER

# Chapter V: Your Freedom Needs Free Software

*by Richard Stallman*

*Many of us know that governments can threaten the human rights of software users through censorship and surveillance of the Internet. Many do not realize that the software they run on their home or work computers can be an even worse threat. Thinking of software as "just a tool," they suppose that it obeys them, when in fact it often obeys others instead.*

The software running in most computers is nonfree, proprietary software: controlled by software companies, not by its users. Users can't check what these programs do, nor prevent them from doing what they don't want. Most people accept this because they have seen no other way, but it is simply wrong to give developers power over the users' computer.

This unjust power, as usual, tempts its wielders to further misdeeds. If a computer talks to a network, and you don't control the software in it, it can easily spy on you. Microsoft Windows spies on users; for instance, it reports what words a user searches for in her own files, and what other programs are installed. RealPlayer spies too; it reports what the user plays. Cell phones are full of nonfree software, which spies. Cell phones send out localizing signals even when "off," many can send out your precise GPS location whether you wish or not, and some models can be switched on remotely as listening devices. Users can't fix these malicious features because they don't have control.

Some proprietary software is designed to restrict and attack its users. Windows Vista[41] was a big advance in this field; the reason it required replacement of old hardware is that the new models were designed to support unbreakable restrictions. Microsoft thus required users to pay for shiny new shackles. Vista was also designed to permit forced updating by corporate authority. Hence the Bad Vista campaign, which urged Windows users not to "upgrade" to Vista. For later Windows versions, which are even more malicious[42], we now have Upgrade from Windows[43]. Mac OS also contains features designed to restrict its users[44].

In 1999, Microsoft installed back doors for the US government's use. Researchers disagree on what those back doors can do[45], but Microsoft has made that question less important by its forced software installation, which is even worse. If the 1999 code won't let NSA spy on you, Microsoft can force another change which will do so. Other proprietary programs may or may not have back doors, but since we cannot check them, we cannot trust them.

The only way to assure that your software is working for you is to insist on free/libre software. This means users get the source code, are free to study and change it, and are free to redistribute it with or without changes. The GNU/Linux system[46], developed specifically for users' freedom[47], includes office

applications, multimedia, games, and everything you really need to run a computer. See our list of totally free/libre versions of GNU/Linux[48].

A special problem occurs when activists for social change use proprietary software, because its developers, who control it, may be companies they wish to protest—or that work hand in glove with the states whose policies they oppose. Control of our software by a proprietary software company, whether it be Microsoft, Apple, Adobe or Skype, means control of what we can say, and to whom. This threatens our freedom in all areas of life.

There is also danger in using a company's server to do your word processing or email—and not just if you are in China, as US lawyer Michael Springmann discovered. In 2003, AOL not only handed over to the police his confidential discussions with clients, it also made his email and his address list disappear, and didn't admit this was intentional until one of its staff made a slip. Springmann gave up on getting his data back.

The US is not the only state that doesn't respect human rights, so keep your data on your own computer, and your backups under your own custody—and run your computer with free/libre software.

- END OF CHAPTER

# Chapter VI: Why Software Should Not Have Owners

*by Richard Stallman*

Digital information technology contributes to the world by making it easier to copy and modify information. Computers promise to make this easier for all of us.

Not everyone wants it to be easier. The system of copyright gives software programs "owners," most of whom aim to withhold software's potential benefit from the rest of the public. They would like to be the only ones who can copy and modify the software that we use.

The copyright system grew up with printing—a technology for mass-production copying. Copyright fit in well with this technology because it restricted only the mass producers of copies. It did not take freedom away from readers of books. An ordinary reader, who did not own a printing press, could copy books only with pen and ink, and few readers were sued for that.

Digital technology is more flexible than the printing press: when information has digital form, you can easily copy it to share it with others. This very flexibility makes a bad fit with a system like copyright. That's the reason for the increasingly nasty and draconian measures now used to enforce software copyright. Consider these four practices of the Software Publishers Association (SPA):

- Massive propaganda saying it is wrong to disobey the owners to help your friend.
- Solicitation for stool pigeons to inform on their coworkers and colleagues.
- Raids (with police help) on offices and schools, in which people are told they must prove they are innocent of illegal copying.
- Prosecution (by the US government, at the SPA's request) of people such as MIT's David LaMacchia, not for copying software (he is not accused of copying any), but merely for leaving copying facilities unguarded and failing to censor their use.[1]

All four practices resemble those used in the former Soviet Union, where every copying machine had a guard to prevent forbidden copying, and where individuals had to copy information secretly and pass it from hand to hand as samizdat. There is of course a difference: the motive for information control in the Soviet Union was political; in the US the motive is profit. But it is the actions that affect us, not the motive. Any attempt to block the sharing of information, no matter why, leads to the same methods and the same harshness.

Owners make several kinds of arguments for giving them the power to control how we use information:

- Name calling. Owners use smear words such as “piracy” and “theft,” as well as expert terminology such as “intellectual property” and “damage,” to suggest a certain line of thinking to the public—a simplistic analogy between programs and physical objects.
  Our ideas and intuitions about property for material objects are about whether it is right to *take an object away* from someone else. They don't directly apply to *making a copy* of something. But the owners ask us to apply them anyway.
- Exaggeration. Owners say that they suffer “harm” or “economic loss” when users copy programs themselves. But the copying has no direct effect on the owner, and it harms no one. The owner can lose only if the person who made the copy would otherwise have paid for one from the owner.
  A little thought shows that most such people would not have bought copies. Yet the owners compute their “losses” as if each and every one would have bought a copy. That is exaggeration—to put it kindly.
- The law. Owners often describe the current state of the law, and the harsh penalties they can threaten us with. Implicit in this approach is the suggestion that today's law reflects an unquestionable view of morality—yet at the same time, we are urged to regard these penalties as facts of nature that can't be blamed on anyone.
  This line of persuasion isn't designed to stand up to critical thinking; it's intended to reinforce a habitual mental pathway.
  It's elementary that laws don't decide right and wrong. Every American should know that, in the 1950s, it was against the law in many states for a black person to sit in the front of a bus; but only racists would say sitting there was wrong.
- Natural rights. Authors often claim a special connection with programs they have written, and go on to assert that, as a result, their desires and interests concerning the program simply outweigh those of anyone else—or even those of the whole rest of the world. (Typically companies, not authors, hold the copyrights on software, but we are expected to ignore this discrepancy.)
  To those who propose this as an ethical axiom—the author is more important than you—I can only say that I, a notable software author myself, call it bunk.
  But people in general are only likely to feel any sympathy with the natural rights claims for two reasons.
  One reason is an overstretched analogy with material objects. When I cook spaghetti, I do object if someone else eats it, because then I cannot eat it. His action hurts me exactly as much as it benefits him; only one of us can eat the spaghetti, so the question is, which one? The smallest distinction between us is enough to tip the ethical balance.
  But whether you run or change a program I wrote affects you directly and me only indirectly. Whether you give a copy to your friend affects you and your friend much more than it affects me. I shouldn't have the power to tell you not to do these things. No one should.
  The second reason is that people have been told that natural rights for authors is the accepted and unquestioned tradition of our society.

As a matter of history, the opposite is true. The idea of natural rights of authors was proposed and decisively rejected when the US Constitution was drawn up. That's why the Constitution only *permits* a system of copyright and does not *require* one; that's why it says that copyright must be temporary. It also states that the purpose of copyright is to promote progress—not to reward authors. Copyright does reward authors somewhat, and publishers more, but that is intended as a means of modifying their behavior.

The real established tradition of our society is that copyright cuts into the natural rights of the public—and that this can only be justified for the public's sake.

- Economics. The final argument made for having owners of software is that this leads to production of more software.

  Unlike the others, this argument at least takes a legitimate approach to the subject. It is based on a valid goal—satisfying the users of software. And it is empirically clear that people will produce more of something if they are well paid for doing so.

  But the economic argument has a flaw: it is based on the assumption that the difference is only a matter of how much money we have to pay. It assumes that *production of software* is what we want, whether the software has owners or not.

  People readily accept this assumption because it accords with our experiences with material objects. Consider a sandwich, for instance. You might well be able to get an equivalent sandwich either gratis or for a price. If so, the amount you pay is the only difference. Whether or not you have to buy it, the sandwich has the same taste, the same nutritional value, and in either case you can only eat it once. Whether you get the sandwich from an owner or not cannot directly affect anything but the amount of money you have afterwards.

  This is true for any kind of material object—whether or not it has an owner does not directly affect what it *is*, or what you can do with it if you acquire it.

  But if a program has an owner, this very much affects what it is, and what you can do with a copy if you buy one. The difference is not just a matter of money. The system of owners of software encourages software owners to produce something—but not what society really needs. And it causes intangible ethical pollution that affects us all.

What does society need? It needs information that is truly available to its citizens—for example, programs that people can read, fix, adapt, and improve, not just operate. But what software owners typically deliver is a black box that we can't study or change.

Society also needs freedom. When a program has an owner, the users lose freedom to control part of their own lives.

And, above all, society needs to encourage the spirit of voluntary cooperation in its citizens. When software owners tell us that helping our neighbors in a natural way is “piracy,” they pollute our society's civic spirit.

This is why we say that free software is a matter of freedom, not price.

The economic argument for owners is erroneous, but the economic issue is real. Some people write useful software for the pleasure of writing it or for admiration and love; but if we want more software than those people write, we need to raise funds.

Since the 1980s, free software developers have tried various methods of finding funds, with some success. There's no need to make anyone rich; a typical income is plenty of incentive to do many jobs that are less satisfying than programming.

For years, until a fellowship made it unnecessary, I made a living from custom enhancements of the free software I had written. Each enhancement was added to the standard released version and thus eventually became available to the general public. Clients paid me so that I would work on the enhancements they wanted, rather than on the features I would otherwise have considered highest priority.

Some free software developers make money by selling support services. In 1994, Cygnus Support, with around 50 employees, estimated that about 15 percent of its staff activity was free software development—a respectable percentage for a software company.

In the early 1990s, companies including Intel, Motorola, Texas Instruments and Analog Devices combined to fund the continued development of the GNU C compiler. Most GCC development is still done by paid developers. The GNU compiler for the Ada language was funded in the 90s by the US Air Force, and continued since then by a company formed specifically for the purpose.

The free software movement is still small, but the example of listener-supported radio in the US shows it's possible to support a large activity without forcing each user to pay.

As a computer user today, you may find yourself using a proprietary program. If your friend asks to make a copy, it would be wrong to refuse. Cooperation is more important than copyright. But underground, closet cooperation does not make for a good society. A person should aspire to live an upright life openly with pride, and this means saying no to proprietary software.

You deserve to be able to cooperate openly and freely with other people who use software. You deserve to be able to learn how the software works, and to teach your students with it. You deserve to be able to hire your favorite programmer to fix it when it breaks.

You deserve free software.

## Footnote

1. The charges were subsequently dismissed.

# Chapter VII: Tivoization

There is a paradoxical class of firmware, for which the source code is free software, because it carries a free software license[49], but specific hardware, for which these programs are designed, renders any binaries produced from that source code nonfree in practice. That is because that hardware requires the binary to be signed by the hardware manufacturer, either in order to run at all, or in order to make use of crucial hardware facilities, effectively forbidding users to run modified versions. We call these programs *tivoized blobs*.

While it is still physically possible to replace the released binary on the hardware that enforces signatures, it is useless to do so, since the hardware would refuse to run the modified version, or to do some special job such as decoding the DRM. Therefore, the freedom #1 (one of the four essential freedoms) is missing, and that binary is not free, even though the source code may carry a free software license. Indirectly, tivoization affects the other freedoms (to use and to distribute modified versions), because any modification of the firmware by yourself will result in broken hardware. The binary may qualify as open source[50], because the term "open source" is defined in terms of how the source is treated.

The publisher or the manufacturer may advertize this forced signature check as a "feature." Here is their argument: your computer won't boot (or will lack important features) if the hardware detects corrupted firmware, so tivoization protects you and your data. But we should wonder: whom does it protect, and from whom? Who is the owner of this lock? Who decides what is good or bad software for our own computing? If it is not us, then this computer is not loyal.

The tivoization is a not a security feature, it is a trap for our freedoms. It prevents users from upgrading their own hardware or firmware, and it suggests a false sense of security by giving the control of their computer only to some "trusted" firmware provider, compelling users to take the provider's word for their safety.

The firmware that drives the hardware at the lowest level also has the most control over it. It often contains back doors[51] and vulnerabilities[52] which only the "trusted" provider (trusted by the hardware) is allowed to fix.

Preventing unsigned or self-signed versions of the firmware to be run is a way for the manufacturer and publisher to keep the control over your computing, even more than if the source code itself were proprietary! It only serves the purpose of the publisher or manufacturer, and has no benefit to the software user or the hardware owner. On the other hand, supposing some models of hardware will run modified versions, there is no advantage for you in using the manufacturer's signed version instead of a self-signed variant.

Among the most important additions in the GNU General Public License version 3, in 2007, was to prohibit taking a GPLv3-covered program and distributing it under tivoization[53], because it denies users the freedom, in practice, to modify the program and then use the modified version.

As stated by the GNU Free System Distribution Guidelines[54], operating systems which provide such firmware are not free, whether the upstream source code is free or not.

- END OF CHAPTER

# Chapter VIII: When Free Software Isn't (Practically) Superior

*by Benjamin Mako Hill*

The Open Source Initiative's mission statement reads, “Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.”

For more than a decade now, the Free Software Foundation has argued against this “open source” characterization of the free software movement. Free software advocates have primarily argued against this framing because “open source” is an explicit effort to deemphasize our core message of freedom and obscure our movement's role in the success of the software we have built. We have argued that “open source” is bad, fundamentally, because it attempts to keep people from talking about software freedom. But there is another reason we should be wary of the open source framing. The fundamental open source argument, as quoted in the mission statement above, is often incorrect.

Although the Open Source Initiative suggests “the promise of open source is better quality, higher reliability, more flexibility,” this promise is not always realized. Although we do not often advertise the fact, any user of an early-stage free software project can explain that free software is not always as convenient, in purely practical terms, as its proprietary competitors. Free software is sometimes low quality. It is sometimes unreliable. It is sometimes inflexible. If people take the arguments in favor of open source seriously, they must explain why open source has not lived up to its “promise” and conclude that proprietary tools would be a better choice. There is no reason we should have to do either.

Richard Stallman speaks to this in his article on Why Open Source Misses the Point when he explains, “The idea of open source is that allowing users to change and redistribute the software will make it more powerful and reliable. But this is not guaranteed. Developers of proprietary software are not necessarily incompetent. Sometimes they produce a program that is powerful and reliable, even though it does not respect the users' freedom.”

For open source, poor-quality software is a problem to be explained away or a reason to eschew the software altogether. For free software, it is a problem to be worked through. For free software advocates, glitches and missing features are never a source of shame. Any piece of free software that respects users' freedom has a strong inherent advantage over a proprietary competitor that does not. Even if it has other issues, free software always has freedom.

Of course, every piece of free software must start somewhere. A brand-new piece of software, for example, is unlikely to be more featureful than an established proprietary tool. Projects begin with many bugs and improve over time. While open source advocates might argue that a project will grow into usefulness over time and with luck, free software projects represent important contributions on day one to a free software advocate. Every piece of software that gives users control over their technology is a step forward. Improved quality as a project matures is the icing on the cake.

A second, perhaps even more damning, fact is that the collaborative, distributed, peer-review development process at the heart of the definition of open source bears little resemblance to the practice of software development in the vast majority of projects under free (or "open source") licenses.

Several academic studies of free software hosting sites[55] SourceForge and Savannah[56] have shown what many free software developers who have put a codebase online already know first-hand. The vast majority of free software projects are not particularly collaborative. The median number of contributors to a free software project on SourceForge? One. A lone developer. SourceForge projects at the ninety-fifth percentile by participant size have only five contributors. More than half of these free software projects—and even most projects that have made several successful releases and been downloaded frequently, are the work of a single developer with little outside help.

By emphasizing the power of collaborative development and "distributed peer review," open source approaches seem to have very little to say about why one should use, or contribute to, the vast majority of free software projects. Because the purported benefits of collaboration cannot be realized when there is no collaboration, the vast majority of free development projects are at no technical advantage with respect to a proprietary competitor.

For free software advocates, these same projects are each seen as important successes. Because every piece of free software respects its users' freedom, advocates of software freedom argue that each piece of free software begins with an inherent ethical advantage over proprietary competitors—even a more featureful one. By emphasizing freedom over practical advantages, free software's advocacy is rooted in a technical reality in a way that open source is often not. When free software is better, we can celebrate this fact. When it is not, we need not treat it as a damning critique of free software advocacy or even as a compelling argument against the use of the software in question.

Open source advocates must defend their thesis that freely developed software should, or will with time, be better than proprietary software. Free software supporters can instead ask, "How can we make free software better?" In a free software framing, high quality software exists as a means to an end rather than an end itself. Free software developers should strive to create functional, flexible software that serves its users well. But doing so is not the only way to make steps toward solving what is both an easier and a much more profoundly important goal: respecting and protecting their freedom.

Of course, we do not need to reject arguments that collaboration can play an important role in creating high-quality software. In many of the most successful free software projects, it clearly has done exactly that. The benefits of collaboration become something to understand, support, and work towards, rather than something to take for granted in the face of evidence that refuses to conform to ideology.

- END OF CHAPTER

# Chapter IX: Applying the Free Software Criteria

*by Richard Stallman*

The four essential freedoms provide the criteria for whether a particular piece of code is free/libre (i.e., respects its users' freedom). How should we apply them to judge whether a software package, an operating system, a computer, or a web page is fit to recommend?

Whether a program is free affects first of all our decisions about our private activities: to maintain our freedom, we need to reject the programs that would take it away. However, it also affects what we should say to others and do with others.

A nonfree program is an injustice. To distribute a nonfree program, to recommend a nonfree program to other people, or more generally steer them into a course that leads to using nonfree software, means leading them to give up their freedom. To be sure, leading people to use nonfree software is not the same as installing nonfree software in their computers, but we should not lead people in the wrong direction.

At a deeper level, we must not present a nonfree program as a solution because that would grant it legitimacy. Nonfree software is a problem; to present it as a solution denies the existence of the problem[57].

This article explains how we apply the basic free software criteria to judging various kinds of things, so we can decide whether to recommend them or not.

## Software packages

For a software package to be free, all the code in it must be free. But not only the code. Since documentation files including manuals, README, change log, and so on are essential technical parts of a software package, they must be free as well[58].

A software package is typically used alongside many other packages, and interacts with some of them. Which kinds of interaction with nonfree programs are ethically acceptable?

We developed GNU so that there would be a free operating system, because in 1983 none existed. As we developed the initial components of GNU, in the 1980s, it was inevitable that each component depended on nonfree software. For instance, no C program could run without a nonfree C compiler until GCC was working, and none could run without Unix libc until glibc was working. Each component could run only on nonfree systems, because all systems were nonfree.

After we released a component that could run on some nonfree systems, users ported it to other nonfree systems; those ports were no worse, ethically, than the platform-specific code we needed to develop these components, so we incorporated their patches.

When the kernel, Linux, was freed in 1992, it filled the last gap in the GNU system. (Initially, in 1991, Linux had been distributed under a nonfree license.) The combination of GNU and Linux made a complete free operating system—GNU/Linux.

At that point, we could have deleted the support for nonfree platforms, but we decided not to. A nonfree system is an injustice, but it's not our fault a user runs one. Supporting a free program on that system does not compound the injustice. And it's useful, not only for users of those systems, but also for attracting more people to contribute to developing the free program.

However, a nonfree program that runs on top of a free program is a completely different issue, because it leads users to take a step away from freedom. In some cases we disallow this: for instance, GCC prohibits nonfree plug-ins[59]. When a program permits nonfree add-ons, it should at least not steer people towards using them. For instance, we choose LibreOffice over OpenOffice because OpenOffice suggests use of nonfree add-ons, while LibreOffice shuns them. We developed IceCat[60] initially to avoid proposing the nonfree add-ons suggested by Firefox.

In practice, if the IceCat package explains how to run IceCat on MacOS, that will not lead people to run MacOS. But if it talked about some nonfree add-on, that would encourage IceCat users to install the add-on. Therefore, the IceCat package, including manuals and web site, shouldn't talk about such things.

Sometimes a free program and a nonfree program interoperate but neither is based on the other. Our rule for such cases is that if the nonfree program is very well known, we should tell people how to use our free program with it; but if the proprietary program is obscure, we should not hint that it exists. Sometimes we support interoperation with the nonfree program if that is installed, but avoid telling users about the possibility of doing so.

We reject “enhancements” that would work only on a nonfree system. Those would encourage people to use the nonfree system instead of GNU, scoring an own-goal.

## GNU/Linux distros

After the liberation of Linux in 1992, people began developing GNU/Linux distributions (“distros”). Only a few distros are entirely free software[61].

The rules for a software package apply to a distro too: an ethical distro must contain only free software and steer users only towards free software. But what does it mean for a distro to "contain" a particular software package?

Some distros install programs from binary packages that are part of the distro; others build each program from upstream source, and literally *contain* only the recipes to download and build it. For issues of freedom, how a distro installs a given package is not significant; if it presents that package as an option, or its web site does, we say it "contains" that package.

The users of a free system have control over it, so they can install whatever they wish. Free distros provide general facilities with which users can install their own programs and their modified versions of free programs; they can also install nonfree programs. Providing these general facilities is not an ethical flaw in the distro, because the distro's developers are not responsible for what users get and install on their own initiative.

The developers become responsible for installation of nonfree software when they steer the users toward a nonfree program—for instance, by putting it in the distro's list of packages, or distributing it from their server, or presenting it as a solution rather than a problem. This is the point where most GNU/Linux distros have an ethical flaw.

People who install software packages on their own have a certain level of sophistication: if we tell them "Baby contains nonfree code, but Gbaby is free," we can expect them to take care to remember which is which. But distros are recommended to ordinary users who would forget such details. They would think, "What name did they say I should use? I think it was Baby."

Therefore, to recommend a distro to the general public, we insist that its name not be similar to a distro we reject, so our message recommending only the free distro can be reliably transmitted.

Another difference between a distro and a software package is how likely it is for nonfree code to be added. The developers of a program carefully check the code they add. If they have decided to make the program free, they are unlikely to add nonfree code. There have been exceptions, including the very harmful case of the "binary blobs" that were added to Linux, but they are a small fraction of the free programs that exist.

By contrast, a GNU/Linux distro typically contains thousands of packages, and the distro's developers may add hundreds of packages a year. Without a careful effort to avoid packages that contain some nonfree software, some will surely creep in. Since the free distros are few in number, we ask the developers of each free distro to make a commitment to keep the distro free software by removing any nonfree code or malware, as a condition for listing that distro. See the GNU free system distribution guidelines.

We don't ask for such promises for free software packages: it's not feasible, and fortunately not necessary. To get promises from the developers of 30,000 free programs to keep them free would avoid a few problems, at the cost of much work for the FSF staff; in addition, most of those developers have no relationship with the GNU Project and might have no interest in making us any promises. So we deal with the rare cases that change from free to nonfree, when we find out about them.

## Peripherals

A computer peripheral needs software in the computer—perhaps a driver, perhaps firmware to be loaded by the system into the peripheral to make it run. Thus, a peripheral is acceptable to use and recommend if it can be used from a computer that has no nonfree software installed—the peripheral's driver, and any firmware that the system needs to load into it, are free.

It is simple to check this: connect the peripheral to a computer running a totally free GNU/Linux distro and see if it works. But most users would like to know *before* they buy the peripheral, so we list information about many peripherals in h-node.org, a hardware database for fully free operating systems.

## Computers

A computer contains software at various levels. On what criterion should we certify that a computer "Respects Your Freedom"?

Obviously the operating system and everything above it must be free. In the 90s, the startup software (BIOS, then) became replaceable, and since it runs on the CPU, it is the same sort of issue as the operating system. Thus, programs such as firmware and drivers that are installed in or with the system or the startup software must be free.

If a computer has hardware features that require nonfree drivers or firmware installed with the system, we may be able to endorse it. If it is usable without those features, and if we think most people won't be led to install the nonfree software to make them function, then we can endorse it. Otherwise, we can't. This will be a judgment call.

A computer can have modifiable preinstalled firmware and microcode at lower levels. It can also have code in true read-only memory. We decided to ignore these programs in our certification criteria today, because otherwise no computer could comply, and because firmware that is not normally changed is ethically equivalent to circuits. So our certification criteria cover only the code that runs on the computer's main processor and is not in true read-only memory. When and as free software becomes possible for other levels of processing, we will require free software at those levels too.

Since certifying a product is active promotion of it, we insist that the seller support us in return, by talking about free software rather than open source and referring to the combination of GNU and Linux as "GNU/Linux". We have no obligation to actively promote projects that won't recognize our work and support our movement.

See our certification criteria[62].

## Web pages

Nowadays many web pages contain complex JavaScript programs and won't work without them. This is a harmful practice since it hampers users' control over their computing. Furthermore, most of these programs are nonfree, an injustice. Often the JavaScript code spies on the user. JavaScript has morphed into a attack on users' freedom.[63]

To address this problem, we have developed LibreJS[64], an add-on for Firefox that blocks nontrivial nonfree JavaScript code. (There is no need to block the simple scripts that implement minor user interface hacks.) We ask sites to please free their JavaScript programs and mark their licenses for LibreJS to recognize.

Meanwhile, is it ethical to link to a web page that contains a nonfree JavaScript program? If we were totally unyielding, we would link only to free JavaScript code. However, many pages do work even when their JavaScript code is not run. Also, you will most often encounter nonfree JavaScript in other ways besides following our links; to avoid it, you must use LibreJS or disable JavaScript. So we have decided to go ahead and link to pages that work without nonfree JavaScript, while urging users to protect themselves from nonfree JavaScript in general.

However, if a page can't do its job without running the nonfree JavaScript code, linking to it undeniably asks people to run that nonfree code. On principle, we do not link to such pages.

## Conclusion

Applying the basic idea that *software should be free* to different situations leads to different practical policies. As new situations arise, the GNU Project and the Free Software Foundation will adapt our freedom criteria so as to lead computer users towards freedom, in practice and in principle. By recommending only freedom-respecting programs, distros, and hardware products, and stating your policy, you can give much-needed support to the free software movement.

- END OF CHAPTER

# Chapter X: Imperfection is not the same as oppression

*by Richard Stallman*

When a free program lacks capabilities that users want, that is unfortunate; we urge people to add what is missing. Some would go further and claim that a program is not even free software if it lacks certain functionality—that it denies freedom 0 (the freedom to run the program as you wish) to users or uses that it does not support. This argument is misguided because it is based on identifying capacity with freedom, and imperfection with oppression.

Each program inevitably has certain functionalities and lacks others that might be desirable. There are some jobs it can do, and others it can't do without further work. This is the nature of software.

The absence of key functionality can mean certain users find the program totally unusable. For instance, if you only understand graphical interfaces, a command line program may be impossible for you to use. If you can't see the screen, a program without a screen reader may be impossible for you to use. If you speak only Greek, a program with menus and messages in English may be impossible for you to use. If your programs are written in Ada, a C compiler is impossible for you to use. To overcome these barriers yourself is unreasonable to demand of you. Free software really ought to provide the functionality you need.

Free software really ought to provide it, but the lack of that feature does not make the program nonfree, because it is an imperfection, not oppression.

Making a program nonfree is an injustice committed by the developer that denies freedom to whoever uses it. The developer deserves condemnation for this. It is crucial to condemn that developer, because nobody else can undo the injustice as long as the developer continues to do it. We can, and do, try to rescue the victims by developing a free replacement, but we can't make the nonfree program free.

Developing a free program without adding a certain important feature is not doing wrong to anyone. Rather, it's doing some good but not all the good that people need. Nobody in particular deserves condemnation for not developing the missing feature, since any capable person could do it. It would be ungrateful, as well as self-defeating, to single out the free program's authors for blame for not having done some additional work.

What we can do is state that completing the job calls for doing some additional work. That is constructive because it helps us convince someone to do that work.

If you think a certain extension in a free program is important, please push for it in the way that respects our contributors. Don't criticize the people who contributed the useful code we have. Rather, look for a way to complete the job. You can urge the program's developers to turn their attention to the missing feature when they have time for more work. You can offer to help them. You can recruit people or raise funds to support the work.

- END OF CHAPTER

# Chapter XI: Android and Users' Freedom

*by Richard Stallman*

To what extent does Android respect the freedom of its users? For a computer user that values freedom, that is the most important question to ask about any software system.

In the free/libre software movement, we develop software that respects users' freedom, so we and you can escape from software that doesn't. By contrast, the idea of "open source" focuses on how to develop code; it is a different current of thought whose principal value is code quality rather than freedom. Thus, the concern here is not whether Android is "open," but whether it allows users to be free.

Android is an operating system primarily for mobile phones and other devices, which consists of Linux (Torvalds' kernel), some libraries, a Java platform and some applications. Linux aside, the software of Android versions 1 and 2 was mostly developed by Google; Google released it under the Apache 2.0 license, which is a lax free software license without copyleft.

The version of Linux included in Android is not entirely free software, since it contains nonfree "binary blobs" (just like Torvalds' version of Linux), some of which are really used in some Android devices. Android platforms use other nonfree firmware, too, and nonfree libraries. Aside from those, the source code of Android versions 1 and 2, as released by Google, is free software —but this code is insufficient to run the device. Some of the applications that generally come with Android are nonfree, too.

Android is very different from the GNU/Linux operating system[65] because it contains very little of GNU. Indeed, just about the only component in common between Android and GNU/Linux is Linux, the kernel. People who erroneously think "Linux" refers to the entire GNU/Linux combination get tied in knots by these facts, and make paradoxical statements such as "Android contains Linux, but it isn't Linux."(1) Absent this confusion, the situation is simple: Android contains Linux, but not GNU; thus, Android and GNU/Linux are mostly different, because all they have in common is Linux.

Within Android, Linux the kernel remains a separate program, with its source code under GNU GPL version 2[66]. To combine Linux with code under the Apache 2.0 license would be copyright infringement, since GPL version 2 and Apache 2.0 are incompatible[67]. Rumors that Google has somehow converted Linux to the Apache license are erroneous; Google has no power to change the license on the code of Linux, and did not try. If the authors of Linux allowed its use under GPL version 3, then that code could be combined with Apache-licensed code, and the combination could be released under GPL version 3. But Linux has not been released that way.

Google has complied with the requirements of the GNU General Public License for Linux, but the Apache license on the rest of Android does not require source release. Google said it would never publish the source code of Android 3.0 (aside from Linux). Android 3.1 source code was also withheld, making Android 3, apart from Linux, nonfree software pure and simple.

Google said it withheld the 3.0 source code because it was buggy, and that people should wait for the next release. That may be good advice for people who simply want to run the Android system, but the users should be the ones to decide this. Anyway, developers and tinkerers who want to include some of the changes in their own versions could use that code just fine.

Fortunately, Google later released the source code for Android 3.* when it released version 4 (also with source code). The problem above turned out to be a temporary aberration rather than a policy shift. However, what happens once may happen again.

In any case, most of the source code of various versions of Android has been released as free software. Does that mean that products using those Android versions respect users' freedom? No, for several reasons.

First of all, most of them contain nonfree Google applications for talking to services such as YouTube and Google Maps. These are officially not part of Android, but that doesn't make the product ok. Many of the free applications available for earlier versions of Android have been replaced by nonfree applications[68]; in 2013 Android devices appeared which provided no way to view photos except through a nonfree Google+ app[69]. In 2014 Google announced that Android versions for TVs, watches and cars would be largely nonfree.[70]

Most Android devices come with the nonfree Google Play software (formerly "Android Market"). This software invites users with a Google account to install nonfree apps. It also has a back door with which Google can forcibly install or deinstall apps. (This probably makes it a universal back door, though that is not proved.) Google Play is officially not part of Android, but that doesn't make it any less bad.

Google has moved many basic general facilities into the nonfree Google Play Services library[71]. If an app's own code is free software but it depends on Google Play Services, that app as a whole is effectively nonfree; it can't run on a free version of Android, such as Replicant.

If you value freedom, you don't want the nonfree apps that Google Play offers. To install free Android apps, you don't need Google Play, because you can get them from f-droid.org.

Android products also come with nonfree libraries. These are officially not part of Android, but since various Android functionalities depend on them, they are part of any real Android installation.

Even the programs that are officially part of Android may not correspond to the source code Google releases. Manufacturers may change this code, and often they don't release the source code for their versions. The GNU GPL requires them to distribute the code for their versions of Linux, assuming they comply. The rest of the code, under the lax Apache license, does not require them to release the source version that they really use.

One user discovered that many of the programs in the Android system that came with his phone were modified to send personal data to Motorola.[72] Some manufacturers add a hidden general surveillance package such as Carrier IQ.[73]

Replicant[74] is the free version of Android. The Replicant developers have replaced many nonfree libraries, for certain device models. The nonfree apps are excluded, but you certainly don't want to use those. By contrast, CyanogenMod (another modified version of Android) is nonfree, as it contains some nonfree programs.

Many Android devices are "tyrants": they are designed so users cannot install and run their own modified software, only the versions approved by some company. In that situation, the executables are not free even if they were made from sources that are free and available to you. However, some Android devices can be "rooted" so users can install different software.

Important firmware or drivers are generally proprietary also. These handle the phone network radio, WiFi, bluetooth, GPS, 3D graphics, the camera, the speaker, and in some cases the microphone too. On some models, a few of these drivers are free, and there are some that you can do without—but you can't do without the microphone or the phone network radio.

The phone network firmware comes preinstalled. If all it did was sit there and talk to the phone network when you wish, we could regard it as equivalent to a circuit. When we insist that the software in a computing device must be free, we can overlook preinstalled firmware that will never be upgraded, because it makes no difference to the user that it's a program rather than a circuit.

Unfortunately, in this case it would be a malicious circuit. Malicious features are unacceptable no matter how they are implemented.

On most Android devices, this firmware has so much control that it could turn the product into a listening device. On some, it controls the microphone. On some, it can take full control of the main computer, through shared memory, and can thus override or replace whatever free software you have installed. With some, perhaps all, models it is possible to exercise remote control of this firmware to overwrite the rest of the software in the device. The point of free software is that we have control of our software and our computing; a system with a back door doesn't qualify. While any computing system might *have* bugs, these devices can *be* bugs. (Craig Murray, in Murder in Samarkand, relates his

involvement in an intelligence operation that remotely converted an unsuspecting target's non-Android portable phone into a listening device.)

In any case, the phone network firmware in an Android phone is not equivalent to a circuit, because the hardware allows installation of new versions and this is actually done. Since it is proprietary firmware, in practice only the manufacturer can make new versions—users can't.

Putting these points together, we can tolerate nonfree phone network firmware provided new versions of it won't be loaded, it can't take control of the main computer, and it can only communicate when and as the free operating system chooses to let it communicate. In other words, it has to be equivalent to circuitry, and that circuitry must not be malicious. There is no technical obstacle to building an Android phone which has these characteristics, but we don't know of any.

Android is not a self-hosting system; development for Android needs to be done on some other system. The tools in Google's "software development kit" (SDK) appear to be free, but it is hard work to check this. The definition files for certain Google APIs are nonfree. Installing the SDK requires signing a proprietary software license, which you should refuse to sign. Replicant's SDK is a free replacement.

Recent press coverage of Android focuses on the patent wars. During 20 years of campaigning for the abolition of software patents, we have warned such wars could happen. Software patents could force elimination of features from Android, or even make it unavailable. See endsoftpatents.org for more information about why software patents must be abolished.

However, the patent attacks and Google's responses are not directly relevant to the topic of this article: how Android products partly approach an ethically system of distribution, and how they fall short. This issue merits the attention of the press too.

Android is a major step towards an ethical, user-controlled, free software portable phone, but there is a long way to go, and Google is taking it in the wrong direction. Hackers are working on Replicant, but it's a big job to support a new device model, and there remains the problem of the firmware. Even though the Android phones of today are considerably less bad than Apple or Windows phones, they cannot be said to respect your freedom.

## Footnote

1. The extreme example of this confusion appears in the site linuxonandroid.com, which offers help to "install Linux [sic] on your Android devices." This is entirely false: what they are installing is a version of the GNU system, *excluding* Linux, which is already present as part of Android. Since that site supports only nonfree GNU/Linux distros, we do not recommend it.

# Chapter XII: Free Software is More Reliable!

Apologists for proprietary software like to say, "free software is a nice dream, but we all know that only the proprietary system can produce reliable products. A bunch of hackers just can't do this."

Empirical evidence disagrees, however; scientific tests, described below, have found GNU software to be *more* reliable than comparable proprietary software.

This should not be a surprise; there are good reasons for the high reliability of GNU software, good reasons to expect free software will often (though not always) have high reliability.

**GNU Utilities Safer!**

Barton P. Miller and his colleagues tested the reliability of Unix utility programs in 1990 and 1995. Each time, GNU's utilities came out considerably ahead. They tested seven commercial Unix systems as well as GNU. By subjecting them to a random input stream, they could "crash (with core dump) or hang (infinite loop) over 40% (in the worst case) of the basic utility programs..."

These researchers found that the commercial Unix systems had a failure rate that ranged from 15% to 43%. In contrast, the failure rate for GNU was only 7%.

Miller also said that, "the three commercial systems that we compared in both 1990 and 1995 noticeably improved in reliability, but still had significant rates of failure (the basic utilities from GNU/Linux still were noticeably better than those of the commercial systems)."

For details, see their paper: Fuzz Revisited: A Re-examination of the Reliability of Unix Utilities and Services (postscript 223k)[75] by Barton P. Miller <bart@cs.wisc.edu>, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, and Jeff Steidl.

**Why Free Software is More Reliable**

It is no fluke that the GNU utilities are so reliable. There are good reasons why free software tends to be of high quality.

One reason is that free software gets the whole community involved in working together to fix problems. Users not only report bugs, they even fix bugs and send in fixes. Users work together, conversing by email, to get to the bottom of a problem and make the software work trouble-free.

Another is that developers really care about reliability. Free software packages do not always compete commercially, but they still compete for a good reputation, and a program which is unsatisfactory will

not achieve the popularity that developers hope for. What's more, an author who makes the source code available for all to see puts his reputation on the line, and had better make the software clean and clear, on pain of the community's disapproval.

**Cancer Clinic Relies on Free Software!**

The Roger Maris Cancer Center in Fargo, North Dakota (the same Fargo which was the scene of a movie and a flood) uses Linux-based GNU systems precisely because reliability is essential. A network of GNU/Linux machines runs the information system, coordinates drug therapies, and performs many other functions. This network needs to be available to the Center's staff at a moment's notice.

According to Dr. G.W. Wettstein <greg@wind.rmcc.com>:

The proper care of our cancer patients would not be what it is today without [GNU/]Linux ... The tools that we have been able to deploy from free software channels have enabled us to write and develop innovative applications which ... do not exist through commercial avenues.

- END OF CHAPTER

# Chapter XIII: Why Free Software Needs Free Documentation

The biggest deficiency in free operating systems is not in the software—it is the lack of good free manuals that we can include in these systems. Many of our most important programs do not come with full manuals. Documentation is an essential part of any software package; when an important free software package does not come with a free manual, that is a major gap. We have many such gaps today.

Once upon a time, many years ago, I thought I would learn Perl. I got a copy of a free manual, but I found it hard to read. When I asked Perl users about alternatives, they told me that there were better introductory manuals—but those were not free (not freedom-respecting).

Why was this? The authors of the good manuals had written them for O'Reilly Associates, which published them with restrictive terms—no copying, no modification, source files not available—which made them nonfree, thus excluded them from the Free World.

That wasn't the first time this sort of thing has happened, and (to our community's great loss) it was far from the last. Proprietary manual publishers have enticed a great many authors to restrict their manuals since then. Many times I have heard a GNU user eagerly tell me about a manual that he is writing, with which he expects to help the GNU Project—and then had my hopes dashed, as he proceeded to explain that he had signed a contract with a publisher that would restrict it so that we cannot use it.

Given that writing good English is a rare skill among programmers, we can ill afford to lose manuals this way.

Free documentation, like free software, is a matter of freedom, not price. The problem with these manuals was not that O'Reilly Associates charged a price for printed copies—that in itself is fine. (The Free Software Foundation sells printed copies[76] of free GNU manuals[77], too.) But GNU manuals are available in source code form, while these manuals are available only on paper. GNU manuals come with permission to copy and modify; the Perl manuals do not. These restrictions are the problems.

The criterion for a free manual is pretty much the same as for free software: it is a matter of giving all users certain freedoms. Redistribution (including commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, on line or on paper. Permission for modification is crucial too.

- The GNU Free Documentation License[78]

As a general rule, I don't believe that it is essential for people to have permission to modify all sorts of articles and books. The issues for writings are not necessarily the same as those for software. For

example, I don't think you or I are obliged to give permission to modify articles like this one, which describe our actions and our views.

But there is a particular reason why the freedom to modify is crucial for documentation for free software. When people exercise their right to modify the software, and add or change its features, if they are conscientious they will change the manual too—so they can provide accurate and usable documentation with the modified program. A manual which forbids programmers from being conscientious and finishing the job, or more precisely requires them to write a new manual from scratch if they change the program, does not fill our community's needs.

While a blanket prohibition on modification is unacceptable, some kinds of limits on the method of modification pose no problem. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are OK. It is also no problem to require modified versions to include notice that they were modified, even to have entire sections that may not be deleted or changed, as long as these sections deal with nontechnical topics. (Some GNU manuals have them.)

These kinds of restrictions are not a problem because, as a practical matter, they don't stop the conscientious programmer from adapting the manual to fit the modified program. In other words, they don't block the free software community from making full use of the manual.

However, it must be possible to modify all the *technical* content of the manual, and then distribute the result through all the usual media, through all the usual channels; otherwise, the restrictions do block the community, the manual is not free, and so we need another manual.

Unfortunately, it is often hard to find someone to write another manual when a proprietary manual exists. The obstacle is that many users think that a proprietary manual is good enough—so they don't see the need to write a free manual. They do not see that the free operating system has a gap that needs filling.

Why do users think that proprietary manuals are good enough? Some have not considered the issue. I hope this article will do something to change that.

Other users consider proprietary manuals acceptable for the same reason so many people consider proprietary software acceptable: they judge in purely practical terms, not using freedom as a criterion. These people are entitled to their opinions, but since those opinions spring from values which do not include freedom, they are no guide for those of us who do value freedom.

Please spread the word about this issue. We continue to lose manuals to proprietary publishing. If we spread the word that proprietary manuals are not sufficient, perhaps the next person who wants to help GNU by writing documentation will realize, before it is too late, that he must above all make it free.

We can also encourage commercial publishers to sell free, copylefted manuals instead of proprietary ones. One way you can help this is to check the distribution terms of a manual before you buy it, and prefer copylefted manuals to noncopylefted ones.

[Note: We maintain a page that lists free books available from other publishers[79]].

- END OF CHAPTER

# Chapter XIV: Should Rockets Have Only Free Software? Free Software and Appliances

*by Richard Stallman*

Could there be a rocket that is totally free software? Should we demand that SpaceX liberate the software in its satellite launching rockets? I don't think the person who asked me this was serious, but answering that question may illuminate similar issues about the sorts of products people really buy today.

As far as I know, software as such is not capable of generating thrust. A rocket is necessarily principally a physical device, so it can't literally *be* free software. But it may include computerized control and telemetry systems, and thus software.

If someone offered to sell me a rocket, I would treat it like any other appliance. Consider, for instance, a thermostat. If it contains software to be modified, all the software in it needs to be free, and I alone should have the authority to decide whether to install some change. If, however, the software in it is not meant ever to be altered, and it communicates *only* through some limited interface, such as buttons on the control panel, a TV remote control, or a USB interface with a fixed set of commands, I would not consider it crucial to know what is inside the thermostat: whether it contains a special-purpose chip, or a processor running code, makes no direct difference to me as user. If it does contain code, it might as well have a special chip instead, so I don't need to care which it is.

I would object if that thermostat sent someone data about my activities, regardless of how that was implemented. Once again, special chip or special code makes no direct difference. Free software in it could give me a way to turn off the surveillance, but that is not the only way. Another is by disconnecting its digital communication antennas, or switching them off.

If the rocket contains software, releasing that as free software can be a contribution to the community, and we should appreciate that contribution—but that is a different issue. Such release also makes it possible for people who have bought the rockets to work on improving the software in them, though the irreversible nature of many rocket failures may discourage tinkering.

Readers have pointed out that SpaceX has received important financial support from the US government[80] to develop its rockets. By rights, accepting this support should require SpaceX to release the rocket software under a free license, even if it uses that software only inside its own rockets.

Given the experience of Tesla cars, which are full of surveillance and tracking malware that Tesla can change but the owner can't, I suppose SpaceX rockets have that too. If someday rockets are sold like

today's cars and tractors, software in them would be unjust, and it would probably be malware. If the manufacturer could install modified software in it but the owner could not, that too would be unjust. People are starting to recognize this: look at the right-to-repair movement, which demands only the beginning of these freedoms (much less than freeing the car's software) and nonetheless faces a hard fight.

However, I don't think SpaceX sells rockets; I think it provides the service of launching payloads in its own rockets. That makes the issue totally different: if you are a customer, you're not operating the rocket; SpaceX is doing that.

The rocket that SpaceX uses is not like your own car or van, or even a car or van leased to you. Rather, it's comparable to a moving company's van that is, for the moment, transporting your books and furniture to your specified destination. It is the moving company that deserves control over the software in that van—not the customer of the moment.

It makes sense to treat the job of transporting your things to Outer Mongolia, or to outer space, as a service because the job is mostly self-contained and mostly independent of the customer ("mostly" does not mean "absolutely" or "100%"), so the instructions for the job are simple (take these boxes to address A by date D).

If SpaceX has released the rocket software under a free license, that would give you the right to make, use and distribute modified versions, but would not give you the right to modify the code running in SpaceX's rocket.

But there is a kind of activity which a hypothetical future spaceship might do, which should never be treated as a service: private computational activity. That's because a private computational activity is exactly what you could do on your own computer in freedom, given suitable free software.

When a program's task is to do computing for you, you are entitled to demand control over what it does and how, not just that it obey your orders as it interprets them. You are entitled, in other words, to use your own copy of a free program, running on a computer you control.

No wonder there are companies that would like you to cede control over your computing activities to them, by labeling those activities as "services" to be done on their servers with programs that they control. Even things as minutely directed by the user as text editing! This is a scheme to get you to substitute their power for your freedom. We call that "Service as a Software Substitute," SaaSS for short (see "Who does that server really serve?"), and we reject it.

For instance, imagine a hypothetical SpaceX Smart Spaceship, which as a "service" wants to know all about your business so SpaceX servers can decide for you what cargoes to buy and sell on which

planets. That planning service would be SaaSS—therefore a dis-service. Instead of using that dis-service, you should do that planning with your copy of free software on your own computer.

SpaceX and others could then legitimately offer you the non-computational service of transporting cargoes, and you could use it sometimes; or you could choose some other method, perhaps to buy a spaceship and operate it yourself.

- END OF CHAPTER

# Chapter XV: Free Hardware and Free Hardware Designs

*by Richard Stallman*

*To what extent do the ideas of free software extend to hardware? Is it a moral obligation to make our hardware designs free, just as it is to make our software free? Does maintaining our freedom require rejecting hardware made from nonfree designs?*

## Definitions

*Free software* is a matter of freedom, not price; broadly speaking, it means that users are free to use the software and to copy and redistribute the software, with or without changes. More precisely, the definition is formulated in terms of the four essential freedoms. To emphasize that "free"refers to freedom, not price, we often use the French or Spanish word "libre" along with "free."

Applying the same concept directly to hardware, *free hardware* means hardware that users are free to use and to copy and redistribute with or without changes. However, there are no copiers for hardware, aside from keys, DNA, and plastic objects' exterior shapes. Most hardware is made by fabrication from some sort of design. The design comes before the hardware.

Thus, the concept we really need is that of a *free hardware design*. That's simple: it means a design that permits users to use the design (i.e., fabricate hardware from it) and to copy and redistribute it, with or without changes. The design must provide the same four freedoms that define free software.

Then we can refer to hardware made from a free design as "free hardware," but "free-design hardware" is a clearer term since it avoids possible misunderstanding.

People first encountering the idea of free software often think it means you can get a copy gratis. Many free programs are available for zero price, since it costs you nothing to download your own copy, but that's not what "free" means here. (In fact, some spyware programs such as Flash Player and Angry Birds[81] are gratis although they are not free.) Saying "libre" along with "free" helps clarify the point.

For hardware, this confusion tends to go in the other direction; hardware costs money to produce, so commercially made hardware won't be gratis (unless it is a loss-leader or a tie-in), but that does not prevent its design from being free/libre. Things you make in your own 3D printer can be quite cheap to make, but not exactly gratis since the raw materials will typically cost something. In ethical terms, the freedom issue trumps the price issue totally, since a device that denies freedom to its users is worth less than nothing.

We can use the term “libre hardware” as a concise equivalent for “hardware made from a free (libre) design.”

The terms “open hardware” and “open source hardware” are used by some with the same concrete meaning as “free-design hardware,” but those terms downplay freedom as an issue. They were derived from the term “open source software,” which refers more or less to free software but without talking about freedom or presenting the issue as a matter of right or wrong. To underline the importance of freedom, we make a point of referring to freedom whenever it is pertinent; since “open” fails to do that, let's not substitute it for “free.”

### Hardware and Software

Hardware and software are fundamentally different. A program, even in compiled executable form, is a collection of data which can be interpreted as instructions for a computer. Like any other digital work, it can be copied and changed using a computer. A copy of a program has no inherent preferred physical form or embodiment.

By contrast, hardware is a physical structure and its physicality is crucial. While the hardware's design might be represented as data, in some cases even as a program, the design is not the hardware. A design for a CPU can't execute a program. You won't get very far trying to type on a design for a keyboard or display pixels on a design for a screen.

Furthermore, while you can use a computer to modify or copy the hardware design, a computer can't convert the design into the physical structure it describes. That requires fabrication equipment.

## The Boundary between Hardware and Software

What is the boundary, in digital devices, between hardware and software? It follows from the definitions. Software is the operational part of a device that can be copied, and modified with a computer; hardware is the operational part that can't be. This is the right way to make the distinction because it relates to the practical consequences.

There is a gray area between hardware and software that contains firmware that *can* be upgraded or replaced, but is not meant ever to be upgraded or replaced once the product is sold. Or perhaps it is possible but unusual, or the manufacturer can release a replacement but you can't. In conceptual terms, the gray area is rather narrow. In practice, it is important because many products fall in it. Indeed, nowadays keyboards, cameras, disk drives and USB memories typically contain an embedded nonfree program that could be replaced by the manufacturer.

We can think of the difference between built-in firmware and equivalent hardware as a minor implementation detail, provided that we are sure in either case that it won't be changed. A hardware circuit can't be changed; that's its nature. If it's acceptable for a device to be implemented with internal circuitry that no one can alter, then an internal program that no one can alter is no worse. It would not be sensible to reject an equivalent internal software implementation, when operationally they are indistinguishable.

The equivalence falls apart, however, when the software implementation is not totally internal and some company can modify that code. For example, when firmware needs to be copied into the device to make the device function, or included in the system distribution that you install, that is no *internal* software implementation; rather, it is a piece of installed nonfree software. It is unjust because some manufacturer can change it but you can't.

In order for a firmware program to be morally equivalent to hardware, it must be unmodifiable. What about when the device can't possibly run without some firmware and it offers a way to modify that? We can make that firmware unmodifiable in practice by taking care never to let that replacement happen. This solution is not entirely clean, but no entirely clean solution has been proposed; this is the only way we know to preserve some meaning for the rejection of nonfree software while using that device. This is much better than just giving up.

But we can't have it both ways. To make preinstalled firmware effectively unmodifiable by not letting anyone invoke the method to change it, we must carry that out without exception even when there are changes we would wish were installed. That means rejecting all upgrades or patches to that firmware.

Some have said that preinstalled firmware programs and Field-Programmable Gate Array chips (FPGAs) "blur the boundary between hardware and software," but I think that is a misinterpretation of the facts. Firmware that is installed during use is software; firmware that is delivered inside the device and can't be changed is software by nature, but we can treat it as if it were a circuit. As for FPGAs, the FPGA itself is hardware, but the gate pattern that is loaded into the FPGA is a kind of firmware.

Running free gate patterns on FPGAs could potentially be a useful method for making digital devices that are free at the circuit level. However, to make FPGAs usable in the free world, we need free development tools for them. The obstacle is that the format of the gate pattern file that gets loaded into the FPGA is secret. For many years there was no model of FPGA for which those files could be produced without nonfree (proprietary) tools.

As of 2015, free software tools are available for programming the Lattice iCE40[82], a common model of FPGA, from input written in a hardware description language (HDL). It is also possible to compile C programs and run them on the Xilinx Spartan 6 LX9 FPGA with free tools[83], but those do not

support HDL input. We recommend that you reject other FPGA models until they too are supported by free tools.

As for the HDL code itself, it can act as software (when it is run on an emulator or loaded into an FPGA) or as a hardware design (when it is realized in immutable silicon or a circuit board).

## The Ethical Question for 3D Printers

Ethically, software must be free; a nonfree program is an injustice. Should we take the same view for hardware designs?

We certainly should, in the fields that 3D printing (or, more generally, any sort of personal fabrication) can handle. Printer patterns to make a useful, practical object (i.e., functional rather than decorative) *must* be free because they are works made for practical use. Users deserve control over these works, just as they deserve control over the software they use. Distributing a nonfree functional object design is as wrong as distributing a nonfree program.

So make sure to choose 3D printers that work with exclusively free software; the Free Software Foundation endorses such printers[84].

## Must We Reject Nonfree Digital Hardware?

Is a nonfree digital [1] hardware design an injustice? Must we, for our freedom's sake, reject all digital hardware made from nonfree designs, as we must reject nonfree software?

Due to the conceptual parallel between hardware designs and software source code, many hardware hackers are quick to condemn nonfree hardware designs just like nonfree software. I disagree because the circumstances for hardware and software are different.

Present-day chip and board fabrication technology resembles the printing press: it lends itself to mass production in a factory. It is more like copying books in 1950 than like copying software today.

Freedom to copy and change software is an ethical imperative because those activities are feasible for those who use software: the equipment that enables you to use the software (a computer) is also sufficient to copy and change it. Today's mobile computers are too weak to be good for this, but anyone can find a computer that's powerful enough.

Moreover, a computer suffices to download and run a version changed by someone else who knows how, even if you are not a programmer. Indeed, nonprogrammers download software and run it every day. This is why free software makes a real difference to nonprogrammers.

How much of this applies to hardware? Not everyone who can use digital hardware knows how to change a circuit design, or a chip design, but anyone who has a PC has the equipment needed to do so. Thus far, hardware is parallel to software, but next comes the big difference.

You can't build and run a circuit design or a chip design in your computer. Constructing a big circuit is a lot of painstaking work, and that's once you have the circuit board. Fabricating a chip is not feasible for individuals today; only mass production can make them cheap enough. With today's hardware technology, users can't download and run a modified version of a widely used digital hardware design, as they could run a modified version of a widely used program. Thus, the four freedoms don't give users today collective control over a hardware design as they give users collective control over a program. That's where the reasoning showing that all software must be free fails to apply to today's hardware technology.

In 1983 there was no free operating system, but it was clear that if we had one, we could immediately use it and get software freedom. All that was missing was the code for one.

In 2014, if we had a free design for a CPU chip suitable for a PC, mass-produced chips made from that design would not give us the same freedom in the hardware domain. If we're going to buy a product mass produced in a factory, this dependence on the factory causes most of the same problems as a nonfree design. For free designs to give us hardware freedom, we need future fabrication technology.

We can envision a future in which our personal fabricators can make chips, and our robots can assemble and solder them together with transformers, switches, keys, displays, fans and so on. In that future we will all make our own computers (and fabricators and robots), and we will all be able to take advantage of modified designs made by those who know hardware. The arguments for rejecting nonfree software will then apply to nonfree hardware designs too.

That future is years away, at least. In the meantime, there is no need to reject hardware with nonfree designs on principle.

## We Need Free Digital Hardware Designs

Although we need not reject digital hardware made from nonfree designs in today's circumstances, we need to develop free designs and should use them when feasible. They provide advantages today, and in the future they may be the only way to use free software.

Free hardware designs offer practical advantages. Multiple companies can fabricate one, which reduces dependence on a single vendor. Groups can arrange to fabricate them in quantity. Having circuit diagrams or HDL code makes it possible to study the design to look for errors or malicious functionalities (it is known that the NSA has procured malicious weaknesses in some computing

hardware). Furthermore, free designs can serve as building blocks to design computers and other complex devices, whose specs will be published and which will have fewer parts that could be used against us.

Free hardware designs may become usable for some parts of our computers and networks, and for embedded systems, before we are able to make entire computers this way.

Free hardware designs may become essential even before we can fabricate the hardware personally, if they become the only way to avoid nonfree software. As common commercial hardware is increasingly designed to subjugate users, it becomes increasingly incompatible with free software, because of secret specifications and requirements for code to be signed by someone other than you. Cell phone modem chips and even some graphics accelerators already require firmware to be signed by the manufacturer. Any program in your computer, that someone else is allowed to change but you're not, is an instrument of unjust power over you; hardware that imposes that requirement is malicious hardware. In the case of cell phone modem chips, all the models now available are malicious.

Some day, free-design digital hardware may be the only platform that permits running a free system at all. Let us aim to have the necessary free digital designs before then, and hope that we have the means to fabricate them cheaply enough for all users.

If you design hardware, please make your designs free. If you use hardware, please join in urging and pressuring companies to make hardware designs free.

## Levels of Design

Software has levels of implementation; a package might include libraries, commands and scripts, for instance. But these levels don't make a significant difference for software freedom because it is feasible to make all the levels free. Designing components of a program is the same sort of work as designing the code that combines them; likewise, building the components from source is the same sort of operation as building the combined program from source. To make the whole thing free simply requires continuing the work until we have done the whole job.

Therefore, we insist that a program be free at all levels. For a program to qualify as free, every line of the source code that composes it must be free, so that you can rebuild the program out of free source code alone.

Physical objects, by contrast, are often built out of components that are designed and build in a different kind of factory. For instance, a computer is made from chips, but designing (or fabricating) chips is very different from designing (or fabricating) the computer out of chips.

Thus, we need to distinguish *levels* in the design of a digital product (and maybe some other kinds of products). The circuit that connects the chips is one level; each chip's design is another level. In an FPGA, the interconnection of primitive cells is one level, while the primitive cells themselves are another level. In the ideal future we will want the design be free at all levels. Under present circumstances, just making one level free is a significant advance.

However, if a design at one level combines free and nonfree parts—for example, a "free" HDL circuit that incorporates proprietary "soft cores"—we must conclude that the design as a whole is nonfree at that level. Likewise for nonfree "wizards" or "macros," if they specify part of the interconnections of chips or programmably connected parts of chips. The free parts may be a step towards the future goal of a free design, but reaching that goal entails replacing the nonfree parts. They can never be admissible in the free world.

## Licenses and Copyright for Free Hardware Designs

You make a hardware design free by releasing it under a free license. We recommend using the GNU General Public License, version 3 or later. We designed GPL version 3 with a view to such use.

Copyleft on circuits, and on nondecorative object shapes, doesn't go as far as one might suppose. The copyright on these designs only applies to the way the design is drawn or written. Copyleft is a way of using copyright law, so its effect carries only as far as copyright law carries.

For instance, a circuit, as a topology, cannot be copyrighted (and therefore cannot be copylefted). Definitions of circuits written in HDL can be copyrighted (and therefore copylefted), but the copyleft covers only the details of expression of the HDL code, not the circuit topology it generates. Likewise, a drawing or layout of a circuit can be copyrighted, so it can be copylefted, but this only covers the drawing or layout, not the circuit topology. Anyone can legally draw the same circuit topology in a different-looking way, or write a different HDL definition that produces the same circuit.

Copyright doesn't cover physical circuits, so when people build instances of the circuit, the design's license will have no legal effect on what they do with the devices they have built.

For drawings of objects, and 3D printer models, copyright doesn't cover making a different drawing of the same purely functional object shape. It also doesn't cover the functional physical objects made from the drawing. As far as copyright is concerned, everyone is free to make them and use them (and that's a freedom we need very much). In the US, copyright does not cover the functional aspects that the design describes, but does cover decorative aspects[85]. When one object has decorative aspects and functional aspects, you get into tricky ground [2].

All this may be true in your country as well, or it may not. Before producing objects commercially or in quantity, you should consult a local lawyer. Copyright is not the only issue you need to be concerned with. You might be attacked using patents, most likely held by entities that had nothing to do with making the design you're using, and there may be other legal issues as well.

Keep in mind that copyright law and patent law are totally different. It is a mistake to suppose that they have anything in common. This is why the term "intellectual property[86]" is pure confusion and should be totally rejected.

## Promoting Free Hardware Designs Through Repositories

The most effective way to push for published hardware designs to be free is through rules in the repositories where they are published. Repository operators should place the freedom of the people who will use the designs above the preferences of people who make the designs. This means requiring designs of useful objects to be free, as a condition for posting them.

For decorative objects, that argument does not apply, so we don't have to insist they must be free. However, we should insist that they be sharable. Thus, a repository that handles both decorative object models and functional ones should have an appropriate license policy for each category.

For digital designs, I suggest that the repository insist on GNU GPL v3-or-later, Apache 2.0, or CC0. For functional 3D designs, the repository should ask the design's author to choose one of four licenses: GNU GPL v3-or-later, Apache 2.0, CC BY-SA, CC BY or CC0. For decorative designs, it should suggest GNU GPL v3-or-later, Apache 2.0, CC0, or any of the CC licenses.

The repository should require all designs to be published as source code, and source code in secret formats usable only by proprietary design programs is not really adequate. For a 3D model, the STL format[87] is not the preferred format for changing the design and thus is not source code, so the repository should not accept it, except perhaps accompanying real source code.

There is no reason to choose one single format for the source code of hardware designs, but source formats that cannot yet be handled with free software should be accepted reluctantly at best.

## Free Hardware Designs and Warranties

In general, the authors of free hardware designs have no moral obligation to offer a warranty to those that fabricate the design. This is a different issue from the sale of physical hardware, which ought to come with a warranty from the seller and/or the manufacturer.

## Conclusion

We already have suitable licenses to make our hardware designs free. What we need is to recognize as a community that this is what we should do and to insist on free designs when we fabricate objects ourselves.

## Footnotes

1. As used here, "digital hardware" includes hardware with some analog circuits and components in addition to digital ones.
2. An article by Public Knowledge gives useful information about this complexity[88], for the US, though it falls into the common mistake of using the bogus concept of "intellectual property" and the propaganda term "protection[89]."

- END OF CHAPTER

# Chapter XVI: What Does It Mean for Your Computer to Be Loyal?

*by Richard Stallman*

We say that running free software on your computer means that its operation is under your control. Implicitly this presupposes that your computer will do what your programs tell it to do, and no more. In other words, that your computer will be loyal to you.

In 1990 we took that for granted; nowadays, many computers are designed to be disloyal to their users. It has become necessary to spell out what it means for your computer to be a loyal platform that obeys your decisions, which you express by telling it to run certain programs.

Our tentative definition consists of these principles.

**Installability**

Any software that can be replaced by someone else, the user must be empowered to replace.

Thus, if the computer requires a password or some other secret in order to replace some of the software in it, whoever sells you the computer must tell you that secret as well.

**Neutrality towards software**

The computer will run, without prejudice, whatever software you install in it, and let that software do whatever its code says to do.

A feature to check for signatures on the programs that run is compatible with this principle provided the signature checking is fully under the user's control. When that is so, the feature helps implement the user's decisions about which programs to run, rather than thwarting the user's decisions. By contrast, signature checking that is not fully under the user's control violates this principle.

**Neutrality towards protocols**

The computer will communicate, without prejudice, through whatever protocol your installed software implements, with whatever users and whatever other networked computers you direct it to communicate with.

This means that computer does not impose one particular service rather than another, or one protocol rather than another. It does not require the user to get anyone else's permission to communicate via a certain protocol.

**Neutrality towards implementations**

When the computer communicates using any given protocol, it will support doing so, without prejudice, via whatever code you choose (assuming the code implements the intended protocol), and it will do nothing to help any other part of the Internet to distinguish which code you are using or what changes you may have made in it, or to discriminate based on your choice.

This entails that the computer rejects remote attestation, that is, that it does not permit other computers to determine over the network whether your computer is running one particular software load. Remote attestation gives web sites the power to compel you to connect to them only through an application with DRM that you can't break, denying you effective control over the software you use to communicate with them.

We can comprehend remote attestation as a general scheme to allow any web site to impose tivoization or “lockdown” on the local software you connect to it with. Simple tivoization of a program bars modified versions from functioning properly; that makes the program nonfree. Remote attestation by web sites bars modified versions from working with those sites that use it, which makes the program effectively nonfree when using those sites. If a computer allows web sites to bar you from using a modified program with them, it is loyal to them, not to you.

**Neutrality towards data communicated**

When the computer receives data using whatever protocol, it will not limit what the program can do with the data received through that communication.

Any hardware-level DRM violates this principle. For instance, the hardware must not deliver video streams encrypted such that only the monitor can decrypt them.

**Debugability**

The computer always permits you to analyze the operation of a program that is running.

**Completeness**

The principles above apply to all the computer's software interfaces and all communication the computer does. The computer must not have any disloyal programmable facility or do any disloyal communication.

For instance, the AMT functionality in recent Intel processors runs nonfree software that can talk to Intel remotely. Unless disabled, this makes the system disloyal.

For a computer to be fully at your service, it should come with documentation of all the interfaces intended for software running in the computer to use to control the computer. A documentation gap as such doesn't mean the computer is actively disloyal, but does mean there are some aspect of it that are not at your service. Depending on what that aspect does, this might or might not be a real problem.

We ask readers to send criticisms and suggestions about this definition to <computer-principles@gnu.org>.

Loyalty as defined here is the most basic criterion we could think of that is meaningful. It does not require that all the software in the computer be free. However, the presence of nonfree software in the computer is an obstacle to verifying that the computer is loyal, or making sure it remains so.

## History

Here is the list of substantive changes in this page.

- Version 1.6: Add installability requirement.
- Version 1.5: Full documentation is not a requirement for loyalty.

- END OF CHAPTER

# Chapter XVII: Network Services Aren't Free or Nonfree; They Raise Other Issues

*by Richard Stallman*

For programs, we make a distinction between free and nonfree (proprietary). More precisely, this distinction applies to a program that you have a copy of: either you have the four freedoms for your copy or you don't. If you don't, that program does a specific kind of injustice to you, simply because it is nonfree.

The copyright holders of a nonfree program can cure that injustice in a simple, clear way: release the same source code under a free software license. Convincing them to *do* this may be difficult, but the action itself is straightforward.

An activity (such as a service) doesn't exist in the form of copies, so it's not possible for a user to have a copy of it, let alone make more copies. Lacking a copy to modify, the user can't modify it either. As a result, the four freedoms that define free software don't make sense for services. It is meaningless to say that the service is "nonfree," or that it is "free." That distinction makes no sense, for services.

That does not mean that the service treats users justly. Quite the contrary—many services do wrong to their users, in various ways, and we call them "dis-services"—but there is no simple universal fix for this, comparable to that for a nonfree program (to release it as free software so users can run and control their copies and their versions).

To use a culinary analogy, my way of cooking can't be a copy of your way of cooking, not even if I learned to cook by watching you. I might have and use a copy of the *recipe* you use to do your cooking, because a recipe, like a program, is a work and exists in copies, but your recipe is not the same as your way of cooking. (And neither of those is the same as the food produced by your cooking.)

With today's technology, services are often implemented by running programs on computers, but that is not the only way to implement them. (In fact, there are network services that are implemented by asking human beings to enter responses to questions.) In any case, the implementation is not visible to users of the service, so it has no direct effect on them.

A network service can raise issues of free vs nonfree software for its users through the client software needed to use it. If the service requires using a nonfree client program, use of the service requires ceding your freedom to that program. With many web services, the nonfree software is JavaScript code silently installed in the user's browser. The GNU LibreJS program makes it easier to refuse to run this nonfree JavaScript code. But the issue of the client software is logically separate from the service as such.

There is one case where a service is directly comparable to a program: when using the service is equivalent to having a copy of a hypothetical program and running it yourself. In this case, we call it Service as a Software Substitute, or SaaSS (we coined that to be less vague and general than "Software as a Service"), and such a service is always a bad thing. The job it does is the users' own computing, and the users ought to have full control over that. The way for users to have control over their own computing is to do it by running their own copies of a free program. Using someone else's server to do that computing implies losing control of it.

SaaSS is equivalent to using a nonfree program with surveillance features and a universal back door, so you should reject it and replace it with a free program that does the same job.

However, most services' principal functions are communicating or publishing information; they are nothing like running any program yourself, so they are not SaaSS. They could not be replaced by your copy of a program, either; a program running in your own computers, used solely by you and isolated from others, is not communicating with anyone else.

A non-SaaSS service can mistreat users by doing something specific and unjust to the user. For instance, it could misuse the data users send it, or collect too much data (surveillance). It could be designed to mislead or cheat users (for instance, with "dark patterns"). It could impose antisocial or unjust usage conditions. The Franklin Street Statement made a stab at addressing these issues, but we don't have full understanding of them as yet. What's clear is that the issues about a service are *different* from the issues about a program. Thus, for clarity's sake, it is better not to apply the terms "free" and "nonfree" to a service.

Let's suppose a service is implemented using software: the server operator has copies of many programs, and runs them to implement the service. These copies may be free software or not. If the operator developed them and uses them without distributing copies, they are free in a trivial sense since every user (there's only one) has the four freedoms.

If some of them are nonfree, that usually doesn't directly affect users of the service. They are not running those programs; the service operator is running them. In a special situation, these programs can indirectly affect the users of the service: if the service holds private information, users might be concerned that nonfree programs on the server might have back doors allowing someone else to see their data. In effect, nonfree programs on the server require users to trust those programs' developers as well as the service operator. How significant this is in practice depends on the details, including what jobs the nonfree programs do.

However, the one party that is *certainly* mistreated by the nonfree programs implementing the service is the server operator herself. We don't condemn the server operator for being at the mercy of nonfree software, and we certainly don't boycott her for this. Rather, we are concerned for her freedom, as with

any user of nonfree software. Given an opportunity, we try to explain how it curtails her freedom, hoping she will switch to free software.

Conversely, if the service operator runs GNU/Linux or other free software, that's not a virtue that affects you, but rather a benefit for her. We don't praise or thank her for this; rather we felicitate her for making the wise choice.

If she has developed some software for the service, and released it as free software, that's the point at which we have a reason to thank her. We suggest releasing these programs under the GNU Affero GPL, since evidently they are useful on servers.

Why the Affero GPL?[90]

Thus, we don't have a rule that free systems shouldn't use (or shouldn't depend on) services (or sites) implemented with nonfree software. However, they should not depend on, suggest or encourage use of services which are SaaSS; use of SaaSS needs to be replaced by use of free software. All else being equal, it is good to favor those service providers who contribute to the community by releasing useful free software, and good to favor peer-to-peer communication over server-based centralized communication, for activities that don't inherently require a central hub.

- END OF CHAPTER

# Chapter XVIII: Regarding Gnutella

"Gnutella" is, at present, the name for a protocol for distributed file sharing, mostly used for music files. The name also sometimes refers to the network itself, as well as the original Gnutella software. The situation is quite confusing. For more on Gnutella's origin and history, please refer to the Wikipedia article[91] on the subject.

In any case, the name was originally a word play on "GNU" (the original developers planned to release their code under the GNU GPL, and may have had in mind contributing it to the GNU project) and "Nutella" (a candy bar that the original developers enjoyed). However, neither the original software nor any of the related current projects are official GNU packages[92]. We have asked that the Gnutella developers change the name to avoid confusion; perhaps that will happen in the future.

There are a number of free software programs that implement the Gnutella protocol, such as Gtk-Gnutella[93], Mutella[94], and Gnucleus[95]. Please note, however, that none of these programs are officially GNU software either. GNU has its own peer-to-peer networking program, GNUnet[96], whose documentation includes a comparison of the protocols[97].

The Free Software Foundation is concerned with the freedom to copy and change software; music is outside our scope. But there is a partial similarity in the ethical issues of copying software and copying recordings of music. Some articles in the philosophy[98] directory relate to the issue of copying for things other than software. Some of the other people's articles[99] we have links to are also relevant.

No matter what sort of published information is being shared, we urge people to reject the assumption that some person or company has a natural right to prohibit sharing and dictate exactly how the public can use it. Even the US legal system nominally rejects[100] that anti-social idea.

- END OF CHAPTER

# Chapter XIX: When Free Software Depends on Nonfree

*by Richard Stallman*

When a program is free software (free as in freedom), that means it gives users the four freedoms, so that they control what the program does. In most cases, that is sufficient for the program's distribution to be ethical; but not always. There are additional problems that can arise in specific circumstances. This article describes a subtle problem, where upgrading the free program requires using a nonfree program.

If the free program's use depends unavoidably on another program which is nonfree, we say that the free program is "trapped." Its code is free software, and you may be able to copy pieces of its code into other free programs with good, ethical results. But you shouldn't *run* the trapped program, because that entails surrendering your freedom to the other nonfree program.

Someone who upholds the principles of free software would not knowingly make a program that is trapped. However, many free programs are developed by people or companies that don't particularly support these principles, or don't understand the problem.

Dependence on a nonfree program can take various forms. The most basic form is when the programming language used has no free implementation. The first programs I wrote for the GNU system in the 1980s, including GNU Emacs, GDB and GNU Make, had to be compiled with AT&T's nonfree C compiler, because there was no free C compiler until I wrote GCC. Fortunately, this kind of problem is mostly a thing of the past; we now have free compilers and platforms for just about all the languages anyone uses for writing free software.

We can release the program from this kind of trap by translating it to another language, or by releasing a free implementation of the language it's written in. Thus, when a full free Java implementation became available, that released all the free Java programs from the Java Trap[101].

This kind of dependence is conceptually simple because it stems from the situation at one given instant in time. At time T, free program P won't run without nonfree programming platform Q. To borrow a term from linguistics, this relationship is "synchronic."

More recently, we have seen another kind of dependence in database programs, where you can build and run any given version of the program in the free world, but upgrading from version N to version N+1 requires a nonfree program.

This happens because the internal format of the database changes from version N to version N+1. If you have been seriously using version N, you probably have a large existing database in the version N format. To upgrade to version N+1 of the database software, you need to reformat that database.

If the way you are supposed to do this is by running a proprietary database reformat program, or using the developer's service which is SaaSS (Service as a Software Substitute), the database software is trapped—but in a more subtle way. Any single version of the database program can be used without nonfree software or SaaSS. The problem arises when you try to keep using the program for the long term, which entails upgrading it from time to time; you can't use it this way without some nonfree software or equivalent. This database program is trapped across time—we could call it "diachronically trapped," borrowing another term from linguistics.

For example, the program OpenERP (since renamed "Odoo"), though free, is diachronically trapped. GNU Health[102], our free package for running a medical clinic, initially used OpenERP. In 2011, GNU Health developer Luis Falcón discovered that upgrading to the next version of OpenERP required sending the database (full of patients' medical data) to OpenERP's server for reformatting. This is SaaSS: it requires the user of GNU Health (a clinic) to entrust its own computing and its data to the company developer of OpenERP. Rather than bow down, Falcón rewrote GNU Health to use Tryton[103] instead.

Using SaaSS is inherently equivalent to running a proprietary program with snooping functionality and a universal back door. The service could keep a copy of the databases that users reformat. Even if we can trust the company that runs the service never to intentionally show any form of the data to anyone, we can't be sure that it won't be accessed by the intelligence agencies of various countries[104] or security-breaking crackers (please don't call them "hackers"[105]).

When a program is diachronically trapped, releasing it from the trap requires more than a one-time job of programming. Rather, the job has to be done continually, each time there is a change in the data format. Launching a project with a long-term commitment to continue doing this is not easy. It may be easier to pressure the company to stop trying to trap users—by rejecting the trapped program until it does so. Given how difficult it is to free the program, you had better stay away from it.

It is possible to try out a diachronically trapped free program without nonfree software, but if you're going to do more than dabble, you must steer clear of really using it. Both businesses and individuals will find fine free alternatives that don't have such a problem; all it takes to avoid the trap is to recognize it.

- END OF CHAPTER

# Chapter XX: Is It Ever a Good Thing to Use a Nonfree Program?

*by Richard Stallman*

The question here is, is it ever a good thing to use a nonfree program? Our conclusion is that it is usually a bad thing, harmful to yourself and in some cases to others.

If you run a nonfree program on your computer, it denies your freedom; the immediate wrong is directed at you [1].

That does *not* mean you're an "evildoer" or "sinner" for running a nonfree program. When the harm you're doing is mainly to yourself, we hope you will stop, for your own sake.

Sometimes you may face great pressure to run a nonfree program; we don't say you must defy that pressure at all costs (though it is inspiring when someone does that), but we do urge you to look for occasions to where you can refuse, even in small ways[106].

If you recommend that others run the nonfree program, or lead them to do so, you're leading them to give up their freedom. Thus, we have a responsibility not to lead or encourage others to run nonfree software. Where the program uses a secret protocol for communication, as in the case of Skype, your own use of it pressures others to use it too, so it is especially important to avoid any use of these programs.

But there is one special case where using some nonfree software, and even urging others to use it, can be a positive thing. That's when the use of the nonfree software aims directly at putting an end to the use of that very same nonfree software [2].

**In the past**

In 1983 I decided to develop the GNU operating system, as a free replacement for Unix. The feasible way to do it was to write and test the components one by one on Unix. But was it legitimate to use Unix for this? And was it legitimate to ask others to use Unix for this, given that Unix was proprietary software? (Of course, if it had not been proprietary, it would not have required replacing.)

The conclusion I reached was that using Unix to put an end to the use of Unix was legitimate for me to suggest to other developers. I likened it to participating in small ways in some evil activity, such as a criminal gang or a dishonest political campaign, in order to expose it and shut it down. While participating in the activity is wrong in itself, shutting it down excuses minor peripheral participation,

comparable to merely using Unix. This argument would not justify being a ringleader, but I was only considering using Unix, not going to work for its development team.

The job of replacing Unix was completed when the last essential component was replaced by Linux, the kernel started by Linus Torvalds in 1991. We still add to the GNU/Linux system, but that doesn't require using Unix, so it isn't a reason for using Unix—not any more. Thus, whenever you're using a nonfree program for this sort of reason, you should reconsider from time to time whether the need still exists.

**Nowadays**

However, there are other nonfree programs we still need to replace, and the analogous question often arises. Should you run the nonfree driver for a peripheral to help you develop a free replacement driver? (More precisely, is it ethical for us to suggest that you do so?) Yes, by all means. Is it ok to run the nonfree JavaScript on a web site in order to file complaint asking the webmasters to free that JavaScript code, or make the site work without it? Definitely—but other than that, you should have LibreJS block it for you.

But this justification won't stretch any further. People that develop nonfree software, even software with malicious functionalities, often try to excuse this on the grounds that they fund some development of free software. However, a business that is basically wrong can't be legitimized by spending some of the profits on a worthy cause. For instance, some (not all) of the activities of the Gates Foundation are laudable, but they don't excuse Bill Gates's career, or Microsoft. If the business works directly against the worthy cause it tries to legitimize itself with, that is a self-contradiction and it undermines the cause.

Even using a nonfree program to develop free software in general is better to avoid, and not suggest to others. For instance, we should not ask people to run Windows or MacOS in order to make free applications run on them. As developer of Emacs and GCC, I accepted changes to make them support nonfree systems such as VMS, Windows and MacOS. I had no reason to reject that code, even though people had run nonfree systems to write it. Their use of unjust systems was not at my request or suggestion; rather, they were already using them before starting to write changes for GNU. They also did the packaging of releases for those systems.

The “developing its own replacement” exception is valid within its limits, and crucial for the progress of free software, but we must resist stretching it any further lest it turn into an all-purpose excuse for any profitable activity with nonfree software.

## Footnotes

1. Using the nonfree program can have unfortunate indirect effects, such as rewarding the perpetrator and encouraging more use of that program. This is a further reason to shun use of nonfree programs.
   Most proprietary programs come with an End User License Agreement that hardly anyone reads. Tucked away in it, in most cases, is an unethical commitment to behave like an uncooperative, bad neighbor. It claims you promised not to distribute copies to others, or even lend someone a copy.
   To carry out such a commitment is more wrong than to break it. No matter what legalistic arguments they might make, the developers can hardly claim their shady trick gives users a moral obligation to be uncooperative.
   However, we think that the truly moral path is to carefully reject such agreements.

2. Occasionally it is necessary to use and even upgrade a nonfree system on a machine in order to install a free system to replace it on that machine. This is not exactly the same issue, but the same arguments apply: it is legitimate to recommend running some nonfree software momentarily in order to remove it.

- END OF CHAPTER

# Chapter XXI: The Free Software Movement and UDI

*by Richard Stallman*

A project called UDI (Uniform Driver Interface) aims to define a single interface between operating system kernels and device drivers. What should the free software movement make of this idea?

If we imagine a number of operating systems and hardware developers, all cooperating on an equal footing, UDI (if technically feasible) would be a very good idea. It would permit us to develop just one driver for any given hardware device, and then all share it. It would enable a higher level of cooperation.

When we apply the idea to the actual world, which contains both free software developers seeking cooperation, and proprietary software developers seeking domination, the consequences are very different. No way of using UDI can benefit the free software movement. If it does anything, it will divide and weaken us.

If Linux supported UDI, and if we started designing new drivers to communicate with Linux through UDI, what would the consequences be?

- People could run free GPL-covered Linux drivers with Windows systems.This would help only Windows users; it would do nothing for us users of free operating systems. It would not directly hurt us, either; but the developers of GPL-covered free drivers could be discouraged to see them used in this way, and that would be very bad. It can also be a violation of the GNU GPL to link the drivers into a proprietary kernel. To increase the temptation to do so is asking for trouble.

- People could run nonfree Windows drivers on GNU/Linux systems.This would not directly affect the range of hardware supported by free software. But indirectly it would tend to decrease the range, by offering a temptation to the millions of GNU/Linux users who have not learned to insist on freedom for its own sake. To the extent that the community began to accept the temptation, we would be moving to using nonfree drivers instead of writing free ones.
  UDI would not in itself obstruct development of free drivers. So if enough of us rejected the temptation, we could still develop free drivers despite UDI, just as we do without UDI.
  But why encourage the community to be weaker than it needs to be? Why make unnecessary difficulties for the future of free software? Since UDI does no good for us, it is better to reject UDI.

Given these consequences, it is no surprise that Intel, a supporter of UDI, has started to "look to the Linux community for help with UDI." How does a rich and self-seeking company approach a

cooperating community? By asking for a handout, of course. They have nothing to lose by asking, and we might be caught off guard and say yes.

Cooperation with UDI is not out of the question. We should not label UDI, Intel, or anyone, as a Great Satan. But before we participate in any proposed deal, we must judge it carefully, to make sure it is advantageous for the free software community, not just for proprietary system developers. On this particular issue, that means requiring that cooperation take us a step further along a path that leads to the ultimate goal for free kernels and drivers: supporting *all* important hardware with free drivers.

One way to make a deal a good one could be by modifying the UDI project itself. Eric Raymond has proposed that UDI compliance could require that the driver be free software. That would be ideal, but other alternatives could also work. Just requiring source for the driver to be published, and not a trade secret, could do the job—because even if that driver is not free, it would at least tell us what we need to know to write a free driver.

Intel could also do something outside of UDI to help the free software community solve this problem. For example, there may be some sort of certification that hardware developers seek, that Intel plays a role in granting. If so, Intel could agree to make certification more difficult if the hardware specs are secret. That might not be a complete solution to the problem, but it could help quite a bit.

One difficulty with any deal with Intel about UDI is that we would do our part for Intel at the beginning, but Intel's payback would extend over a long time. In effect, we would be extending credit to Intel. But would Intel continue to repay its loan? Probably yes, if we get it in writing and there are no loopholes; otherwise, we can't count on it. Corporations are notoriously untrustworthy; the people we are dealing with may have integrity, but they could be overruled from above, or even replaced at any time with different people. Even a CEO who owns most of the stock can be replaced through a buy-out. When making a deal with a corporation, always get a binding commitment in writing.

It does not seem likely that Intel would offer a deal that gives us what we need. In fact, UDI seems designed to make it easier to keep specifications secret.

Still, there is no harm in keeping the door unlocked, as long as we are careful about who we let in.

- END OF CHAPTER

# Chapter XXII: Why Open Source Misses the Point of Free Software

by Richard Stallman

The terms “free software” and “open source” stand for almost the same range of programs. However, they say deeply different things about those programs, based on different values. The free software movement campaigns for freedom for the users of computing; it is a movement for freedom and justice. By contrast, the open source idea values mainly practical advantage and does not campaign for principles. This is why we do not agree with open source, and do not use that term.

When we call software “free,” we mean that it respects the users' essential freedoms: the freedom to run it, to study and change it, and to redistribute copies with or without changes. This is a matter of freedom, not price, so think of “free speech,” not “free beer.”

These freedoms are vitally important. They are essential, not just for the individual users' sake, but for society as a whole because they promote social solidarity—that is, sharing and cooperation. They become even more important as our culture and life activities are increasingly digitized. In a world of digital sounds, images, and words, free software becomes increasingly essential for freedom in general.

Tens of millions of people around the world now use free software; the public schools of some regions of India and Spain now teach all students to use the free GNU/Linux operating system. Most of these users, however, have never heard of the ethical reasons for which we developed this system and built the free software community, because nowadays this system and community are more often spoken of as “open source,” attributing them to a different philosophy in which these freedoms are hardly mentioned.

The free software movement has campaigned for computer users' freedom since 1983. In 1984 we launched the development of the free operating system GNU, so that we could avoid the nonfree operating systems that deny freedom to their users. During the 1980s, we developed most of the essential components of the system and designed the GNU General Public License (GNU GPL) to release them under—a license designed specifically to protect freedom for all users of a program.

Not all of the users and developers of free software agreed with the goals of the free software movement. In 1998, a part of the free software community splintered off and began campaigning in the name of “open source.” The term was originally proposed to avoid a possible misunderstanding of the term “free software,” but it soon became associated with philosophical views quite different from those of the free software movement.

Some of the supporters of open source considered the term a “marketing campaign for free software,” which would appeal to business executives by highlighting the software's practical benefits, while not raising issues of right and wrong that they might not like to hear. Other supporters flatly rejected the free software movement's ethical and social values. Whichever their views, when campaigning for open source, they neither cited nor advocated those values. The term “open source” quickly became associated with ideas and arguments based only on practical values, such as making or having powerful, reliable software. Most of the supporters of open source have come to it since then, and they make the same association. Most discussion of “open source” pays no attention to right and wrong, only to popularity and success; here's a typical example[107]. A minority of supporters of open source do nowadays say freedom is part of the issue, but they are not very visible among the many that don't.

The two now describe almost the same category of software, but they stand for views based on fundamentally different values. For the free software movement, free software is an ethical imperative, essential respect for the users' freedom. By contrast, the philosophy of open source considers issues in terms of how to make software “better”—in a practical sense only. It says that nonfree software is an inferior solution to the practical problem at hand.

For the free software movement, however, nonfree software is a social problem, and the solution is to stop using it and move to free software.

“Free software.” “Open source.” If it's the same software (or nearly so), does it matter which name you use? Yes, because different words convey different ideas. While a free program by any other name would give you the same freedom today, establishing freedom in a lasting way depends above all on teaching people to value freedom. If you want to help do this, it is essential to speak of “free software.”

We in the free software movement don't think of the open source camp as an enemy; the enemy is proprietary (nonfree) software. But we want people to know we stand for freedom, so we do not accept being mislabeled as open source supporters. What we advocate is not “open source,” and what we oppose is not “closed source.” To make this clear, we avoid using those terms.

## Practical Differences between Free Software and Open Source

In practice, open source stands for criteria a little looser than those of free software. As far as we know, all existing released free software source code would qualify as open source. Nearly all open source software is free software, but there are exceptions.

First, some open source licenses are too restrictive, so they do not qualify as free licenses. For example, Open Watcom is nonfree because its license does not allow making a modified version and using it privately. Fortunately, few programs use such licenses.

Second, trademark requirements added on top of the code's copyright license can make a program nonfree. For instance, the Rust compiler may be nonfree, because the trademark conditions forbid selling copies or distributing modified versions, unless you fully remove all *uses of the trademark*. Just what that requires in practice is not clear.

Third, the criteria for open source are concerned solely with the use of the source code. Indeed, almost all the items in the Open Source Definition[108] are formulated as conditions on the software's *source license* rather than on what users are *free to do*. However, people often describe an executable as "open source," because its source code is available that way. That causes confusion in paradoxical situations where the source code is open source (and free) but the executable itself is nonfree.

The trivial case of this paradox is when a program's source code carries a weak free license, one without copyleft, but its executables carry additional nonfree conditions. Supposing the executables correspond exactly to the released sources—which may or may not be so—users can compile the source code to make and distribute free executables. That's why this case is trivial; it is no grave problem.

The nontrivial case is harmful and important. Many products containing computers check signatures on their executable programs to block users from effectively using different executables; only one privileged company can make executables that can run in the device and use its full capabilities. We call these devices "tyrants," and the practice is called "tivoization" after the product (Tivo) where we first saw it. Even if the executable is made from free source code, and nominally carries a free license, the users cannot usefully run modified versions of it, so the executable is de-facto nonfree.

Many Android products contain nonfree tivoized executables of Linux, even though its source code is under GNU GPL version 2. (We designed GNU GPL version 3 to prohibit this practice; too bad Linux did not adopt it.) These executables, made from source code that is open source and free, are generally spoken of as "open source," but they are *not* free software.

## Common Misunderstandings of "Free Software" and "Open Source"

The term "free software" is prone to misinterpretation: an unintended meaning, "software you can get for zero price," fits the term just as well as the intended meaning, "software which gives the user certain freedoms." We address this problem by publishing the definition of free software, and by saying "Think of 'free speech,' not 'free beer.'" This is not a perfect solution; it cannot completely eliminate the problem. An unambiguous and correct term would be better, if it didn't present other problems.

Unfortunately, all the alternatives in English have problems of their own. We've looked at many that people have suggested, but none is so clearly "right" that switching to it would be a good idea. (For instance, in some contexts the French and Spanish word "libre" works well, but people in India do not

recognize it at all.) Every proposed replacement for "free software" has some kind of semantic problem —and this includes "open source software."

The official definition of open source software (which is published by the Open Source Initiative and is too long to include here) was derived indirectly from our criteria for free software. It is not the same; it is a little looser in some respects. Nonetheless, their definition agrees with our definition in most cases.

However, the obvious meaning for the expression "open source software" is "You can look at the source code." Indeed, most people seem to misunderstand "open source software" that way. (The clear term for that meaning is "source available.") That criterion is much weaker than the free software definition, much weaker also than the official definition of open source. It includes many programs that are neither free nor open source.

Why do people misunderstand it that way? Because that is the natural meaning of the words "open source." But the concept for which the open source advocates sought another name was a variant of that of free software.

Since the obvious meaning for "open source" is not the meaning that its advocates intend, the result is that most people misunderstand the term. According to writer Neal Stephenson, "Linux is 'open source' software meaning, simply, that anyone can get copies of its source code files." I don't think he deliberately sought to reject or dispute the official definition. I think he simply applied the conventions of the English language to come up with a meaning for the term. The state of Kansas[109] published a similar definition: "Make use of open-source software (OSS). OSS is software for which the source code is freely and publicly available, though the specific licensing agreements vary as to what one is allowed to do with that code."

The *New York Times* ran an article that stretched the meaning of the term[110] to refer to user beta testing—letting a few users try an early version and give confidential feedback—which proprietary software developers have practiced for decades.

The term has even been stretched to include designs for equipment that are published without a patent[111]. Patent-free equipment designs can be laudable contributions to society, but the term "source code" does not pertain to them.

Open source supporters try to deal with this by pointing to their official definition, but that corrective approach is less effective for them than it is for us. The term "free software" has two natural meanings, one of which is the intended meaning, so a person who has grasped the idea of "free speech, not free beer" will not get it wrong again. But the term "open source" has only one natural meaning, which is different from the meaning its supporters intend. So there is no succinct way to explain and justify its official definition. That makes for worse confusion.

Another misunderstanding of "open source" is the idea that it means "not using the GNU GPL." This tends to accompany another misunderstanding that "free software" means "GPL-covered software." These are both mistaken, since the GNU GPL qualifies as an open source license and most of the open source licenses qualify as free software licenses. There are many free software licenses aside from the GNU GPL.

The term "open source" has been further stretched by its application to other activities, such as government, education, and science, where there is no such thing as source code, and where criteria for software licensing are simply not pertinent. The only thing these activities have in common is that they somehow invite people to participate. They stretch the term so far that it only means "participatory" or "transparent," or less than that. At worst, it has become a vacuous buzzword[112].

## Different Values Can Lead to Similar Conclusions—but Not Always

Radical groups in the 1960s had a reputation for factionalism: some organizations split because of disagreements on details of strategy, and the two daughter groups treated each other as enemies despite having similar basic goals and values. The right wing made much of this and used it to criticize the entire left.

Some try to disparage the free software movement by comparing our disagreement with open source to the disagreements of those radical groups. They have it backwards. We disagree with the open source camp on the basic goals and values, but their views and ours lead in many cases to the same practical behavior—such as developing free software.

As a result, people from the free software movement and the open source camp often work together on practical projects such as software development. It is remarkable that such different philosophical views can so often motivate different people to participate in the same projects. Nonetheless, there are situations where these fundamentally different views lead to very different actions.

The idea of open source is that allowing users to change and redistribute the software will make it more powerful and reliable. But this is not guaranteed. Developers of proprietary software are not necessarily incompetent. Sometimes they produce a program that is powerful and reliable, even though it does not respect the users' freedom. Free software activists and open source enthusiasts will react very differently to that.

A pure open source enthusiast, one that is not at all influenced by the ideals of free software, will say, "I am surprised you were able to make the program work so well without using our development model, but you did. How can I get a copy?" This attitude will reward schemes that take away our freedom, leading to its loss.

The free software activist will say, “Your program is very attractive, but I value my freedom more. So I reject your program. I will get my work done some other way, and support a project to develop a free replacement.” If we value our freedom, we can act to maintain and defend it.

## Powerful, Reliable Software Can Be Bad

The idea that we want software to be powerful and reliable comes from the supposition that the software is designed to serve its users. If it is powerful and reliable, that means it serves them better.

But software can be said to serve its users only if it respects their freedom. What if the software is designed to put chains on its users? Then powerfulness means the chains are more constricting, and reliability that they are harder to remove. Malicious features, such as spying on the users, restricting the users, back doors, and imposed upgrades are common in proprietary software, and some open source supporters want to implement them in open source programs.

Under pressure from the movie and record companies, software for individuals to use is increasingly designed specifically to restrict them. This malicious feature is known as Digital Restrictions Management (DRM) (see DefectiveByDesign.org) and is the antithesis in spirit of the freedom that free software aims to provide. And not just in spirit: since the goal of DRM is to trample your freedom, DRM developers try to make it hard, impossible, or even illegal for you to change the software that implements the DRM.

Yet some open source supporters have proposed “open source DRM” software. Their idea is that, by publishing the source code of programs designed to restrict your access to encrypted media and by allowing others to change it, they will produce more powerful and reliable software for restricting users like you. The software would then be delivered to you in devices that do not allow you to change it.

This software might be open source and use the open source development model, but it won't be free software since it won't respect the freedom of the users that actually run it. If the open source development model succeeds in making this software more powerful and reliable for restricting you, that will make it even worse.

## Fear of Freedom

The main initial motivation of those who split off the open source camp from the free software movement was that the ethical ideas of free software made some people uneasy. That's true: raising ethical issues such as freedom, talking about responsibilities as well as convenience, is asking people to think about things they might prefer to ignore, such as whether their conduct is ethical. This can trigger discomfort, and some people may simply close their minds to it. It does not follow that we ought to stop talking about these issues.

That is, however, what the leaders of open source decided to do. They figured that by keeping quiet about ethics and freedom, and talking only about the immediate practical benefits of certain free software, they might be able to "sell" the software more effectively to certain users, especially business.

When open source proponents talk about anything deeper than that, it is usually the idea of making a "gift" of source code to humanity. Presenting this as a special good deed, beyond what is morally required, presumes that distributing proprietary software without source code is morally legitimate.

This approach has proved effective, in its own terms. The rhetoric of open source has convinced many businesses and individuals to use, and even develop, free software, which has extended our community —but only at the superficial, practical level. The philosophy of open source, with its purely practical values, impedes understanding of the deeper ideas of free software; it brings many people into our community, but does not teach them to defend it. That is good, as far as it goes, but it is not enough to make freedom secure. Attracting users to free software takes them just part of the way to becoming defenders of their own freedom.

Sooner or later these users will be invited to switch back to proprietary software for some practical advantage. Countless companies seek to offer such temptation, some even offering copies gratis. Why would users decline? Only if they have learned to value the freedom free software gives them, to value freedom in and of itself rather than the technical and practical convenience of specific free software. To spread this idea, we have to talk about freedom. A certain amount of the "keep quiet" approach to business can be useful for the community, but it is dangerous if it becomes so common that the love of freedom comes to seem like an eccentricity.

That dangerous situation is exactly what we have. Most people involved with free software, especially its distributors, say little about freedom—usually because they seek to be "more acceptable to business." Nearly all GNU/Linux operating system distributions add proprietary packages to the basic free system, and they invite users to consider this an advantage rather than a flaw.

Proprietary add-on software and partially nonfree GNU/Linux distributions find fertile ground because most of our community does not insist on freedom with its software. This is no coincidence. Most GNU/Linux users were introduced to the system through "open source" discussion, which doesn't say that freedom is a goal. The practices that don't uphold freedom and the words that don't talk about freedom go hand in hand, each promoting the other. To overcome this tendency, we need more, not less, talk about freedom.

**"FLOSS" and "FOSS"**

The terms "FLOSS" and "FOSS" are used to be neutral between free software and open source[113]. If neutrality is your goal, "FLOSS" is the better of the two, since it really is neutral. But if you want to

stand up for freedom, using a neutral term isn't the way. Standing up for freedom entails showing people your support for freedom.

**Rivals for Mindshare**

"Free" and "open" are rivals for mindshare. Free software and open source are different ideas but, in most people's way of looking at software, they compete for the same conceptual slot. When people become habituated to saying and thinking "open source," that is an obstacle to their grasping the free software movement's philosophy and thinking about it. If they have already come to associate us and our software with the word "open," we may need to shock them intellectually before they recognize that we stand for something *else*. Any activity that promotes the word "open" tends to extend the curtain that hides the ideas of the free software movement.

Thus, free software activists are well advised to decline to work on an activity that calls itself "open." Even if the activity is good in and of itself, each contribution you make does a little harm on the side by promoting the open source idea. There are plenty of other good activities which call themselves "free" or "libre." Each contribution to those projects does a little extra good on the side. With so many useful projects to choose from, why not choose one which does extra good?

**Conclusion**

As the advocates of open source draw new users into our community, we free software activists must shoulder the task of bringing the issue of freedom to their attention. We have to say, "It's free software and it gives you freedom!"—more and louder than ever. Every time you say "free software" rather than "open source," you help our cause.

## Notes

- Joe Barr wrote an article called Live and let license[114] that gives his perspective on this issue.
- Lakhani and Wolf's paper on the motivation of free software developers[115] says that a considerable fraction are motivated by the view that software should be free. This is despite the fact that they surveyed the developers on SourceForge, a site that does not support the view that this is an ethical issue.

- END OF CHAPTER

# Chapter XXIII: FLOSS and FOSS

*by Richard Stallman*

The two political camps in the free software community are the free software movement and open source. The free software movement is a campaign for computer users' freedom; we say that a nonfree program is an injustice to its users. The open source camp declines to see the issue as a matter of justice to the users, and bases its arguments on practical benefits only.

To emphasize that "free software" refers to freedom and not to price, we sometimes write or say "free (libre) software," adding the French or Spanish word that means free in the sense of freedom. In some contexts, it works to use just "libre software."

A researcher studying practices and methods used by developers in the free software community decided that these questions were independent of the developers' political views, so he used the term "FLOSS," meaning "Free/Libre and Open Source Software," to explicitly avoid a preference between the two political camps. If you wish to be neutral, this is a good way to do it, since this makes the names of the two camps equally prominent.

Others use the term "FOSS," which stands for "Free and Open Source Software." This is meant to mean the same thing as "FLOSS," but it is less clear, since it fails to explain that "free" refers to *freedom*. It also makes "free software" less visible than "open source," since it presents "open source" prominently but splits "free software" apart.

"Free and Open Source Software" is misleading in another way: it suggests that "free and open source" names a single point of view, rather than mentioning two different ones. This conceptualization of the field is an obstacle to understanding the fact that free software and open source are different political positions that disagree fundamentally.

Thus, if you want to be neutral between free software and open source, and clear about them, the way to achieve that is to say "FLOSS," not "FOSS."

We in the free software movement don't use either of these terms, because we don't want to be neutral on the political question. We stand for freedom, and we show it every time—by saying "free" and "libre"—or "free (libre)."

- END OF CHAPTER

# Chapter XXIV: Measures Governments Can Use to Promote Free Software

## And why it is their duty to do so

*by Richard Stallman*

*This article suggests policies for a strong and firm effort to promote free software within the state, and to lead the rest of the country towards software freedom.*

The mission of the state is to organize society for the freedom and well-being of the people. One aspect of this mission, in the computing field, is to encourage users to adopt free software: software that respects the users' freedom. A proprietary (nonfree) program tramples the freedom of those that use it; it is a social problem that the state should work to eradicate.

The state needs to insist on free software in its own computing for the sake of its computational sovereignty (the state's control over its own computing). All users deserve control over their computing, but the state has a responsibility to the people to maintain control over the computing it does on their behalf. Most government activities now depend on computing, and its control over those activities depends on its control over that computing. Losing this control in an agency whose mission is critical undermines national security.

Moving state agencies to free software can also provide secondary benefits, such as saving money and encouraging local software support businesses.

In this text, “state entities” refers to all levels of government, and means public agencies including schools, public-private partnerships, largely state-funded activities such as charter schools, and “private” corporations controlled by the state or established with special privileges or functions by the state.

### Education

The most important policy concerns education, since that shapes the future of the country:

- **Teach only free software**
  Educational activities, or at least those of state entities, must teach only free software (thus, they should never lead students to use a nonfree program), and should teach the civic reasons for insisting on free software. To teach a nonfree program is to teach dependence, which is contrary to the mission of the school.

### The State and the Public

Also crucial are state policies that influence what software individuals and organizations use:

- **Never require nonfree programs**
  Laws and public sector practices must be changed so that they never require or pressure individuals or organizations to use a nonfree program. They should also discourage communication and publication practices that imply such consequences (including Digital Restrictions Management[116]).
- **Distribute only free software**
  Whenever a state entity distributes software to the public, including programs included in or specified by its web pages, it must be distributed as free software, and must be capable of running on a platform containing exclusively free software.
- **State web sites**
  State entity web sites and network services must be designed so that users can use them, without disadvantage, by means of free software exclusively.
- **Free formats and protocols**
  State entities must use only file formats and communication protocols that are well supported by free software, preferably with published specifications. (We do not state this in terms of "standards" because it should apply to nonstandardized interfaces as well as standardized ones.) For example, they must not distribute audio or video recordings in formats that require Flash or nonfree codecs, and public libraries must not distribute works with Digital Restrictions Management.
  To support the policy of distributing publications and works in freedom-respecting formats, the state must insist that all reports developed for it be delivered in freedom-respecting formats.
- **Untie computers from licenses**
  Sale of computers must not require purchase of a proprietary software license. The seller should be required by law to offer the purchaser the option of buying the computer without the proprietary software and without paying the license fee.
  The imposed payment is a secondary wrong, and should not distract us from the essential injustice of proprietary software, the loss of freedom which results from using it. Nonetheless, the abuse of forcing users to pay for it gives certain proprietary software developers an additional unfair advantage, detrimental to users' freedom. It is proper for the state to prevent this abuse.

### Computational Sovereignty

Several policies affect the computational sovereignty of the state. State entities must maintain control over their computing, not cede control to private hands. These points apply to all computers, including smartphones.

- **Migrate to free software**
  State entities must migrate to free software, and must not install, or continue using, any nonfree software except under a temporary exception. Only one agency should have the authority to grant these temporary exceptions, and only when shown compelling reasons. This agency's goal should be to reduce the number of exceptions to zero.
- **Develop free IT solutions**
  When a state entity pays for development of a computing solution, the contract must require it be delivered as free software, and that it be designed such that one can both run it and develop it on a 100%-free environment. All contracts must require this, so that if the developer does not comply with these requirements, the work cannot be paid for.
- **Choose computers for free software**
  When a state entity buys or leases computers, it must choose among the models that come closest, in their class, to being capable of running without any proprietary software. The state should maintain, for each class of computers, a list of the models authorized based on this criterion. Models available to both the public and the state should be preferred to models available only to the state.
- **Negotiate with manufacturers**
  The state should negotiate actively with manufacturers to bring about the availability in the market (to the state and the public) of suitable hardware products, in all pertinent product areas, that require no proprietary software.
- **Unite with other states**
  The state should invite other states to negotiate collectively with manufacturers about suitable hardware products. Together they will have more clout.

### Computational Sovereignty II

The computational sovereignty (and security) of the state includes control over the computers that do the state's work. This requires avoiding Service as a Software Substitute, unless the service is run by a state agency under the same branch of government, as well as other practices that diminish the state control over its computing. Therefore,

- **State must control its computers**
  Every computer that the state uses must belong to or be leased by the same branch of government that uses it, and that branch must not cede to outsiders the right to decide who has physical access to the computer, who can do maintenance (hardware or software) on it, or what software should be installed in it. If the computer is not portable, then while in use it must be in a physical space of which the state is the occupant (either as owner or as tenant).

### Influence Development

State policy affects free and nonfree software development:

- **Encourage free**
  The state should encourage developers to create or enhance free software and make it available to the public, e.g. by tax breaks and other financial incentive. Contrariwise, no such incentives should be granted for development, distribution or use of nonfree software.
- **Don't encourage nonfree**
  In particular, proprietary software developers should not be able to "donate" copies to schools and claim a tax write-off for the nominal value of the software. Proprietary software is not legitimate in a school.

### E-waste

Freedom should not imply e-waste:

- **Replaceable software**
  Many modern computers are designed to make it impossible to replace their preloaded software with free software. Thus, the only way to free them is to junk them. This practice is harmful to society.
  Therefore, it should be illegal, or at least substantially discouraged through heavy taxation, to sell, import or distribute in quantity a new computer (that is, not second-hand) or computer-based product for which secrecy about hardware interfaces or intentional restrictions prevent users from developing, installing and using replacements for any and all of the installed software that the manufacturer could upgrade. This would apply, in particular, to any device on which "jailbreaking"[117] is needed to install a different operating system, or in which the interfaces for some peripherals are secret.

### Technological neutrality

With the measures in this article, the state can recover control over its computing, and lead the country's citizens, businesses and organizations towards control over their computing. However, some object on the grounds that this would violate the "principle" of technological neutrality.

The idea of technological neutrality is that the state should not impose arbitrary preferences on technical choices. Whether that is a valid principle is disputable, but it is limited in any case to issues that are merely technical. The measures advocated here address issues of ethical, social and political importance, so they are outside the scope of *technological* neutrality[118]. Only those who wish to

subjugate a country would suggest that its government be “neutral” about its sovereignty or its citizens' freedom.

- END OF CHAPTER

# Chapter XXV: Why Schools Should Exclusively Use Free Software

*by Richard Stallman*

Educational activities, including schools of all levels from kindergarten to university, have a moral duty to teach only free software.

All computer users ought to insist on free software: it gives users the freedom to control their own computers—with proprietary software, the program does what its owner or developer wants it to do, not what the user wants it to do. Free software also gives users the freedom to cooperate with each other, to lead an upright life. These reasons apply to schools as they do to everyone. However, the purpose of this article is to present the additional reasons that apply specifically to education.

Free software can save schools money, but this is a secondary benefit. Savings are possible because free software gives schools, like other users, the freedom to copy and redistribute the software; the school system can give a copy to every school, and each school can install the program in all its computers, with no obligation to pay for doing so.

This benefit is useful, but we firmly refuse to give it first place, because it is shallow compared to the important ethical issues at stake. Moving schools to free software is more than a way to make education a little "better": it is a matter of doing good education instead of bad education. So let's consider the deeper issues.

Schools have a social mission: to teach students to be citizens of a strong, capable, independent, cooperating and free society. They should promote the use of free software just as they promote conservation and voting. By teaching students free software, they can graduate citizens ready to live in a free digital society. This will help society as a whole escape from being dominated by megacorporations.

In contrast, to teach a nonfree program is implanting dependence, which goes counter to the schools' social mission. Schools should never do this.

Why, after all, do some proprietary software developers offer gratis copies(1) of their nonfree programs to schools? Because they want to *use* the schools to implant dependence on their products, like tobacco companies distributing gratis cigarettes to school children(2). They will not give gratis copies to these students once they've graduated, nor to the companies that they go to work for. Once you're dependent, you're expected to pay, and future upgrades may be expensive.

Free software permits students to learn how software works. Some students, natural-born programmers, on reaching their teens yearn to learn everything there is to know about their computer and its software. They are intensely curious to read the source code of the programs that they use every day.

Proprietary software rejects their thirst for knowledge: it says, "The knowledge you want is a secret—learning is forbidden!" Proprietary software is the enemy of the spirit of education, so it should not be tolerated in a school, except as an object for reverse engineering.

Free software encourages everyone to learn. The free software community rejects the "priesthood of technology", which keeps the general public in ignorance of how technology works; we encourage students of any age and situation to read the source code and learn as much as they want to know.

Schools that use free software will enable gifted programming students to advance. How do natural-born programmers learn to be good programmers? They need to read and understand real programs that people really use. You learn to write good, clear code by reading lots of code and writing lots of code. Only free software permits this.

How do you learn to write code for large programs? You do that by writing lots of changes in existing large programs. Free Software lets you do this; proprietary software forbids this. Any school can offer its students the chance to master the craft of programming, but only if it is a free software school.

The deepest reason for using free software in schools is for moral education. We expect schools to teach students basic facts and useful skills, but that is only part of their job. The most fundamental task of schools is to teach good citizenship, including the habit of helping others. In the area of computing, this means teaching people to share software. Schools, starting from nursery school, should tell their students, "If you bring software to school, you must share it with the other students. You must show the source code to the class, in case someone wants to learn. Therefore bringing nonfree software to class is not permitted, unless it is for reverse-engineering work."

Of course, the school must practice what it preaches: it should bring only free software to class (except objects for reverse-engineering), and share copies including source code with the students so they can copy it, take it home, and redistribute it further.

Teaching the students to use free software, and to participate in the free software community, is a hands-on civics lesson. It also teaches students the role model of public service rather than that of tycoons. All levels of school should use free software.

If you have a relationship with a school —if you are a student, a teacher, an employee, an administrator, a donor, or a parent— it's your responsibility to campaign for the school to migrate to

free software. If a private request doesn't achieve the goal, raise the issue publicly in those communities; that is the way to make more people aware of the issue and find allies for the campaign.

1. Warning: a school that accepts such an offer may find subsequent upgrades rather expensive.
2. RJ Reynolds Tobacco Company was fined $15m in 2002 for handing out free samples of cigarettes at events attended by children. See http://www.bbc.co.uk/worldservice/sci_tech/features/health/tobaccotrial/usa.htm.

- END OF CHAPTER

# Chapter XXVI: Technological Neutrality and Free Software

*by Richard Stallman*

Proprietary developers arguing against laws to move towards free software often claim this violates the principle of "technological neutrality." The conclusion is wrong, but where is the error?

Technological neutrality is the principle that the state should not impose preferences for or against specific kinds of technology. For example, there should not be a rule that specifies whether state agencies should use solid state memory or magnetic disks, or whether they should use GNU/Linux or BSD. Rather, the agency should let bidders propose any acceptable technology as part of their solutions, and choose the best/cheapest offer by the usual rules.

The principle of technological neutrality is valid, but it has limits. Some kinds of technology are harmful; they may pollute air or water, encourage antibiotic resistance, abuse their users, abuse the workers that make them, or cause massive unemployment. These should be taxed, regulated, discouraged, or even banned.

The principle of technological neutrality applies only to purely technical decisions. It is not "ethical neutrality" or "social neutrality"; it does not apply to decisions about ethical and social issues—such as the choice between free software and proprietary software.

For instance, when the state adopts a policy of migrating to free software in order to restore the computing sovereignty of the country and lead the people towards freedom and cooperation, this isn't a technical preference. This is an ethical, social and political policy, not a technological policy. The state is not supposed to be neutral about maintaining the people's freedom or encouraging cooperation. It is not supposed to be neutral about maintaining or recovering its sovereignty.

It is the state's duty to insist that the software in its public agencies respect the computing sovereignty of the country, and that the software taught in its schools educate its students in freedom and cooperation. The state must insist on free software, exclusively, in public agencies and in education. The state has the responsibility to maintain control of its computing, so it must not surrender that control to Service as a Software Substitute. In addition, the state must not reveal to companies the personal data that it maintains about citizens.

When no ethical imperatives apply to a certain technical decision, it can be left to the domain of technological neutrality.

# Chapter XXVII: The Moral and the Legal

*by Richard Stallman*

Every legal issue about free/libre software is at root a moral issue. Before we think about the legal level of the issue, we need to understand the moral level.

The legal level is about what current laws require. When we in the free software movement make a legal argument, that is what we are arguing about. However, the moral level is what matters most—it is where our goals come from. Liberty resides at that level, which is why we also call it "libre" software.

The two levels are not the same or even parallel. In general, that *X* is currently lawful says nothing about whether *X* is morally legitimate, and vice versa. We might propose to change some laws to better follow some of our moral ideas.

There is a pervasive tendency, especially in the US, to assume that laws dictate right and wrong. If we in the free software movement post articles or letters that discuss only the legal level, readers will tend to assume we agree with that assumption—that what we judge by is legality above all, so that if an action is lawful we are unable to criticize it.

Since our overall purpose is to end the lawful but unjust computing practices (nonfree software and SaaSS) because we judge morally that they are unjust, we must show we do not define morality as "not breaking any laws." We need to keep reminding the public to pay attention to the deeper level, which is the moral level. If, in a communication, we focus on the shallow aspects alone, we miss an opportunity to show the public our deeper message. Because some readers are interested only in the legalities, we must show we don't consider those to be paramount.

In some cases, we contend, morality and legality say opposite things. In the US, distributing a program that can break DRM is illegal; the companies that implement DRM point to this, and hope you will confuse legality with morality. We are careful not to get confused that way. Breaking DRM is morally admirable; what's immoral is to *implement* DRM.

In anything we publish, and anything we send to strangers (they might redistribute it to the public), we have to show that our views about issues are primarily based on the moral level. Even when the immediately crucial part is at the legal level, we must show how we judge programs, and laws themselves, at the moral level. Thus, when people ask whether a program follows the *XYZ* law, we can say, "We believe it does—and, most importantly, it respects users' freedom."

Presenting the two levels in relation to each other is a very good way of showing them both, and also showing how they are related. For instance, when speaking for the FSF, it can be useful to say, "Your program *FOO* contains part of the source code of *GNU BAR*" (a legal issue) "and fails to follow the

GNU GPL rules" (a legal issue), "and that denies other users some of the rights they are entitled to" (the deeper moral issue). "To ensure all users of code from *GNU BAR* fully enjoy the four freedoms for it" (the goal at the moral level), "we invoke our copyright to require you to stop distributing the code that way" (using legal power as a tool to achieve the moral goal).

That is not the only way to present them both. In other contexts, not the FSF, you might need to say something very different. The main thing is to remember to talk about the moral level often, so readers realize it is the deeper and more important of the two levels.

- END OF CHAPTER

# Chapter XXVIII: Saying No to unjust computing even once is help

*by Richard Stallman*

A misunderstanding is circulating that the GNU Project demands you run 100% free software, all the time. Anything less (90%?), and we will tell you to get lost—they say. Nothing could be further from the truth.

Our ultimate goal is digital freedom for all, a world without nonfree software. Some of us, who have made campaigning for digital freedom our goal, reject all nonfree programs. However, as a practical matter, even a little step towards that goal is good. A walk of a thousand miles consists of lots of steps. Each time you don't install some nonfree program, or decide not to run it that day, that is a step towards your own freedom. Each time you decline to run a nonfree program with others, you show them a wise example of long-term thinking. That is a step towards freedom for the world.

If you're caught in a web of nonfree programs, you're surely looking for a chance to pull a few strands off of your body. Each one pulled off is an advance.

Each time you tell the people in some activity, “I'd rather use Zoom less—please count me out today,” you help the free software movement. “I'd like to do this with you, but with Zoom on the other side of the scale, I've decided to decline.” If you accepted the nonfree software before, you could say this: “I'd like to participate, but the software we are using is not good for us. I've decided I should cut down.” Once in a while, you may convince them to use free software instead. At least they will learn that some people care about freedom enough to decline participation for freedom's sake.

If you say no, on one occasion, to conversing with someone or some group via Skype, you have helped. If you say no, on one occasion, to conversing via WhatsApp, Facebook, or Slack, you have helped. If you say no, on one occasion, to editing something via Google Docs, you have helped. If you say no to registering for one meeting in eventbrite.com or meetup.com, you have helped. If you tell one organization you won't use its “portal” or app, so you will deal with it by phone, that helps. Of course, you help more if you stick to your refusal (with kind firmness, of course) and don't let the others change your mind.

Steps add up. If on another day you decline the nonfree program again, you will have helped again. If you say no a few times a week, that adds up over time. When people see you say no, even once, you may inspire them to follow your example.

To give help consistently, you can make this refusal a firm practice, but refusing occasionally is still help. You will help more if you reject several of the nonfree programs that communities have blindly swallowed. Would you ever want to reject them all? There is no need to decide that now.

So tell someone, "Thanks for inviting me, but Zoom/Skype/WhatsApp/whichever is a freedom-denying program, and almost surely snoops on its users; please count me out. I want a different kind of world, and by declining to use it today I am taking a step towards that world."

The FSF recommends freedom-respecting methods[119] for the sorts of communication that unjust systems do. If one of them would be usable, you could add, "If we use XYZ for this conversation, or some other libre software, I could participate."

You can take one step. And once you've done it, sooner or later you can do it again. Eventually you may find you have changed your practices; if you get used to saying no to some nonfree program, you could do it most of the time, maybe even every time. Not only will you have gained an increment of freedom; you will have helped your whole community by spreading awareness of the issue.

- END OF CHAPTER

# Chapter XXIX: Motives For Writing Free Software

Don't make the mistake of supposing that all software development has one simple motive. Here are some of the motives we know influence many people to write free software.

**Fun**

For some people, often the best programmers, writing software is the greatest fun, especially when there is no boss to tell you what to do.
Nearly all free software developers share this motive.

**Political idealism**

The desire to build a world of freedom, and help computer users escape from the power of software developers.

**To be admired**

If you write a successful, useful free program, the users will admire you. That feels very good.

**Professional reputation**

If you write a successful, useful free program, that will suffice to show you are a good programmer.

**Community**

Being part of a community by collaborating with other people in public free software projects is a motive for many programmers.

**Education**

If you write free software, it is often an opportunity to dramatically improve both your technical and social skills; if you are a teacher, encouraging your students to take part in an existing free software project or organizing them into a free software project may provide an excellent opportunity for them.

**Gratitude**
If you have used the community's free programs for years, and it has been important to your work, you feel grateful and indebted to their developers. When you write a program that could be useful to many people, that is your chance to pay it forward.

**Hatred for Microsoft**

It is a mistake to focus our criticism narrowly on Microsoft. Indeed, Microsoft is evil, since it makes nonfree software. Even worse, it is often malware in various ways including DRM. However, many other companies do these things, and the nastiest enemy of our freedom nowadays is Apple. Nonetheless, it is a fact that many people utterly despise Microsoft, and some contribute to free software based on that feeling.

**Money**

A considerable number of people are paid to develop free software or have built businesses around it.

**Wanting a better program to use**

People often work on improvements in programs they use, in order to make them more convenient. (Some commentators recognize no motive other than this, but their picture of human nature is too narrow.)
Human nature is complex, and it is quite common for a person to have multiple simultaneous motives for a single action.

Each person is different, and there could be other motives that are missing from this list. If you know of other motives not listed here, please send email to <campaigns@gnu.org>. If we think the other motives are likely to influence many developers, we will add them to the list.

- END OF CHAPTER

# Chapter XXX: Why Copyleft?

*When it comes to defending everyone's freedom, to lie down and do nothing is an act of weakness, not humility.*

In the GNU Project we usually recommend people use copyleft licenses like GNU GPL, rather than permissive non-copyleft free software licenses. We don't argue harshly against the non-copyleft licenses —in fact, we occasionally recommend them in special circumstances—but the advocates of those licenses show a pattern of arguing harshly against the GPL.

In one such argument, a person stated that his use of one of the BSD licenses was an “act of humility”: “I ask nothing of those who use my code, except to credit me.” It is rather a stretch to describe a legal demand for credit as “humility,” but there is a deeper point to be considered here.

Humility is disregarding your own self-interest, but the interest you abandon when you don't copyleft your code is much bigger than your own. Someone who uses your code in a nonfree program is denying freedom to others, so if you allow that, you're failing to defend those people's freedom. When it comes to defending everyone's freedom, to lie down and do nothing is an act of weakness, not humility.

Releasing your code under one of the BSD licenses, or some other lax, permissive license, is not doing wrong; the program is still free software, and still a contribution to our community. But it is weak, and in most cases it is not the best way to promote users' freedom to share and change software.

Here are specific examples of nonfree versions of free programs that have done major harm to the free world.

- Those who released LLVM under a non-copyleft license enabled nVidia to release a high-quality nonfree compiler for its GPUs, while keeping its instruction set secret. Thus, we can't write a free compiler for that platform without a big reverse engineering job. The nonfree adaptation of LLVM is the only compiler for those machines, and is likely to remain so.
- Intel uses a proprietary version of the MINIX system, which is free but not copylefted, in the Management Engine back door in its modern processors.

- END OF CHAPTER

# Chapter XXXI: Copyleft: Pragmatic Idealism

*by Richard Stallman*

Every decision a person makes stems from the person's values and goals. People can have many different goals and values; fame, profit, love, survival, fun, and freedom, are just some of the goals that a good person might have. When the goal is a matter of principle, we call that idealism.

My work on free software is motivated by an idealistic goal: spreading freedom and cooperation. I want to encourage free software to spread, replacing proprietary software that forbids cooperation, and thus make our society better.

That's the basic reason why the GNU General Public License is written the way it is—as a copyleft. All code added to a GPL-covered program must be free software, even if it is put in a separate file. I make my code available for use in free software, and not for use in proprietary software, in order to encourage other people who write software to make it free as well. I figure that since proprietary software developers use copyright to stop us from sharing, we cooperators can use copyright to give other cooperators an advantage of their own: they can use our code.

Not everyone who uses the GNU GPL has this goal. Many years ago, a friend of mine was asked to rerelease a copylefted program under noncopyleft terms, and he responded more or less like this:

“Sometimes I work on free software, and sometimes I work on proprietary software—but when I work on proprietary software, I expect to get *paid*.”

He was willing to share his work with a community that shares software, but saw no reason to give a handout to a business making products that would be off-limits to our community. His goal was different from mine, but he decided that the GNU GPL was useful for his goal too.

If you want to accomplish something in the world, idealism is not enough—you need to choose a method that works to achieve the goal. In other words, you need to be “pragmatic.” Is the GPL pragmatic? Let's look at its results.

Consider GNU C++. Why do we have a free C++ compiler? Only because the GNU GPL said it had to be free. GNU C++ was developed by an industry consortium, MCC, starting from the GNU C compiler. MCC normally makes its work as proprietary as can be. But they made the C++ front end free software, because the GNU GPL said that was the only way they could release it. The C++ front end included many new files, but since they were meant to be linked with GCC, the GPL did apply to them. The benefit to our community is evident.

Consider GNU Objective C. NeXT initially wanted to make this front end proprietary; they proposed to release it as .o files, and let users link them with the rest of GCC, thinking this might be a way around the GPL's requirements. But our lawyer said that this would not evade the requirements, that it was not allowed. And so they made the Objective C front end free software.

Those examples happened years ago, but the GNU GPL continues to bring us more free software.

Many GNU libraries are covered by the GNU Lesser General Public License, but not all. One GNU library which is covered by the ordinary GNU GPL is Readline, which implements command-line editing. I once found out about a nonfree program which was designed to use Readline, and told the developer this was not allowed. He could have taken command-line editing out of the program, but what he actually did was rerelease it under the GPL. Now it is free software.

The programmers who write improvements to GCC (or Emacs, or Bash, or Linux, or any GPL-covered program) are often employed by companies or universities. When the programmer wants to return his improvements to the community, and see his code in the next release, the boss may say, "Hold on there—your code belongs to us! We don't want to share it; we have decided to turn your improved version into a proprietary software product."

Here the GNU GPL comes to the rescue. The programmer shows the boss that this proprietary software product would be copyright infringement, and the boss realizes that he has only two choices: release the new code as free software, or not at all. Almost always he lets the programmer do as he intended all along, and the code goes into the next release.

The GNU GPL is not Mr. Nice Guy. It says no to some of the things that people sometimes want to do. There are users who say that this is a bad thing—that the GPL "excludes" some proprietary software developers who "need to be brought into the free software community."

But we are not excluding them from our community; they are choosing not to enter. Their decision to make software proprietary is a decision to stay out of our community. Being in our community means joining in cooperation with us; we cannot "bring them into our community" if they don't want to join.

What we *can* do is offer them an inducement to join. The GNU GPL is designed to make an inducement from our existing software: "If you will make your software free, you can use this code." Of course, it won't win 'em all, but it wins some of the time.

Proprietary software development does not contribute to our community, but its developers often want handouts from us. Free software users can offer free software developers strokes for the ego—recognition and gratitude—but it can be very tempting when a business tells you, "Just let us put your package in our proprietary program, and your program will be used by many thousands of people!" The temptation can be powerful, but in the long run we are all better off if we resist it.

The temptation and pressure are harder to recognize when they come indirectly, through free software organizations that have adopted a policy of catering to proprietary software. The X Consortium (and its successor, the Open Group) offers an example: funded by companies that made proprietary software, they strived for a decade to persuade programmers not to use copyleft. When the Open Group tried to make X11R6.4 nonfree software[120], those of us who had resisted that pressure were glad that we did.

In September 1998, several months after X11R6.4 was released with nonfree distribution terms, the Open Group reversed its decision and rereleased it under the same noncopyleft free software license that was used for X11R6.3. Thank you, Open Group—but this subsequent reversal does not invalidate the conclusions we draw from the fact that adding the restrictions was *possible*.

Pragmatically speaking, thinking about greater long-term goals will strengthen your will to resist this pressure. If you focus your mind on the freedom and community that you can build by staying firm, you will find the strength to do it. “Stand for something, or you will fall for anything.”

And if cynics ridicule freedom, ridicule community…if “hard-nosed realists” say that profit is the only ideal…just ignore them, and use copyleft all the same.

- END OF CHAPTER

# Chapter XXXII: The JavaScript Trap

*by Richard Stallman*

*There are two kinds of moral wrongs a web page can do. This page describes the wrong of sending nonfree programs to run in your computer. There is also the wrong we call SaaSS, "Service as a Software Substitute" where the page invites you to send your data so it can do computing on it in the server—computing which is unjust because you have no control over what computing is done.*

**You may be running nonfree programs on your computer every day without realizing it—through your web browser.**

In the free software community, the idea that any nonfree program mistreats its users is familiar. Some of us defend our freedom by rejecting all proprietary software on our computers. Many others recognize nonfreeness as a strike against the program.

Many users are aware that this issue applies to the plug-ins that browsers offer to install, since they can be free or nonfree. But browsers run other nonfree programs which they don't ask you about, or even tell you about—programs that web pages contain or link to. These programs are most often written in JavaScript, though other languages are also used.

JavaScript (officially called ECMAScript, but few use that name) was once used for minor frills in web pages, such as cute but inessential navigation and display features. It was acceptable to consider these as mere extensions of HTML markup, rather than as true software, and disregard the issue.

Some sites still use JavaScript that way, but many use it for major programs that do large jobs. For instance, Google Docs tries to install into your browser a JavaScript program which measures half a megabyte, in a compacted form that we could call Obfuscript. This compacted form is made from the source code, by deleting the extra spaces that make the code readable and the explanatory remarks that make it comprehensible, and replacing each meaningful name in the code with an arbitrary short name so we can't tell what it is supposed to mean.

Part of the meaning of free software is that users have access to the program's source code (its plan). The source code of a program means the preferred form for programmers to modify—including helpful spacing, explanatory remarks, and meaningful names. Compacted code is a bogus, useless substitute for source code; the real source code of these programs is not available to the users, so users cannot understand it; therefore the programs are nonfree.

In addition to being nonfree, many of these programs are *malware* because they snoop on the user[121]. Even nastier, some sites use services which record all the user's actions while looking at the page[122]. The services supposedly "redact" the recordings to exclude some sensitive data

that the web site shouldn't get. But even if that works reliably, the whole purpose of these services is to give the web site other personal data that it shouldn't get.

Browsers don't normally tell you when they load JavaScript programs. Some browsers have a way to turn off JavaScript entirely, but even if you're aware of this issue, it would take you considerable trouble to identify the nontrivial nonfree programs and block them. However, even in the free software community most users are not aware of this issue; the browsers' silence tends to conceal it.

To be clear, the language JavaScript is not inherently better or worse for users' freedom than any other language. It is possible to release a JavaScript program as free software, by distributing the source code under a free software license. If the program is self-contained—if its functioning and purpose are independent of the page it came in—that is fine; you can copy it to a file on your machine, modify it, and visit that file with a browser to run it. It's even possible to package it for installation just like other free programs and invocation with a shell command. These programs present no special moral issue different from those of C programs.

The issue of the JavaScript trap applies when the JavaScript program comes along with a web page that users visit. Those JavaScript programs are written to work with a particular page or site, and the page or site depends on them to function.

Suppose you copy and modify the page's JavaScript code. Then another problem arises: even if the program's source is available, browsers do not offer a way to run your modified version instead of the original when visiting that page or site. The effect is comparable to tivoization, although in principle not quite so hard to overcome.

JavaScript is not the only language web sites use for programs sent to the user. Flash supported programming through an extended variant of JavaScript, but that is a thing of the past. Microsoft Silverlight seems likely to create a problem similar to Flash, except worse, since Microsoft uses it as a platform for nonfree codecs. A free replacement for Silverlight does not do the job adequately for the free world unless it normally comes with free replacement codecs.

Java applets also run in the browser, and raise similar issues. In general, any sort of applet system poses this sort of problem. Having a free execution environment for an applet only brings us far enough to encounter the problem.

It is theoretically possible to program in HTML and CSS, but in practice this capability is limited and inconvenient; merely to make it do something is an impressive hack. Such programs ought to be free, but CSS is not a serious problem for users' freedom as of 2019.

A strong movement has developed that calls for web sites to communicate only through formats and protocols that are free (some say “open”); that is to say, whose documentation is published and which anyone is free to implement. However, the presence of JavaScript programs in web pages makes that criterion insufficient. The JavaScript language itself, as a format, is free, and

use of JavaScript in a web site is not necessarily bad. However, as we've seen above, it can be bad—if the JavaScript program is nonfree. When the site transmits a program to the user, it is not enough for the program to be written in a documented and unencumbered language; that program must be free, too. “Transmits only free programs to the user” must become part of the criterion for an ethical web site.

Silently loading and running nonfree programs is one among several issues raised by “web applications.” The term “web application” was designed to disregard the fundamental distinction between software delivered to users and software running on a server. It can refer to a specialized client program running in a browser; it can refer to specialized server software; it can refer to a specialized client program that works hand in hand with specialized server software. The client and server sides raise different ethical issues, even if they are so closely integrated that they arguably form parts of a single program. This article addresses only the issue of the client-side software. We are addressing the server issue separately.

In practical terms, how can we deal with the problem of nontrivial nonfree JavaScript programs in web sites? The first step is to avoid running it.

What do we mean by “nontrivial”? It is a matter of degree, so this is a matter of designing a simple criterion that gives good results, rather than finding the one correct answer.

Our current criterion is to consider a JavaScript program nontrivial if any of these conditions is met:

- it is referred to as an external script (from another page).
- it declares an array more than 50 elements long.
- it defines a named entity (function or method) that calls anything other than a primitive.
- it defines a named entity with more than three conditional constructs and loop construction.
- code outside of named definitions calls anything but primitives and functions defined further up in the page.
- code outside of named definitions contains more than three conditional constructs and loop construction, total.
- it calls **eval**.
- it does Ajax calls.
- it uses bracket notation for dynamic object property access, which looks like ***object*[*property*]**.
- it alters the DOM.
- it uses dynamic JavaScript constructs that are difficult to analyze without interpreting the program, or is loaded along with scripts that use such constructs. Specifically, using any other construct than a string literal with certain methods (**Obj.write**, **Obj.createElement**, and others).

How do we tell whether the JavaScript code is free? In a separate article[123], we propose a method by which a nontrivial JavaScript program in a web page can state the URL where its source code is located, and can state its license too, using stylized comments.

Finally, we need to change free browsers to detect and block nontrivial nonfree JavaScript in web pages. The program LibreJS detects nonfree, nontrivial JavaScript in pages you visit, and blocks it. LibreJS is included in IceCat, and available as an add-on for Firefox.

Browser users also need a convenient facility to specify JavaScript code to use *instead* of the JavaScript in a certain page. (The specified code might be total replacement, or a modified version of the free JavaScript program in that page.) Greasemonkey comes close to being able to do this, but not quite, since it doesn't guarantee to modify the JavaScript code in a page before that program starts to execute. Using a local proxy works, but is too inconvenient now to be a real solution. We need to construct a solution that is reliable and convenient, as well as sites for sharing changes. The GNU Project would like to recommend sites which are dedicated to free changes only.

These features will make it possible for a JavaScript program included in a web page to be free in a real and practical sense. JavaScript will no longer be a particular obstacle to our freedom—no more than C and Java are now. We will be able to reject and even replace the nonfree nontrivial JavaScript programs, just as we reject and replace nonfree packages that are offered for installation in the usual way. Our campaign for web sites to free their JavaScript can then begin.

In the mean time, there's one case where it is acceptable to run a nonfree JavaScript program: to send a complaint to the website operators saying they should free or remove the JavaScript code in the site. Please don't hesitate to enable JavaScript temporarily to do that—but remember to disable it again afterwards.

**Acknowledgements:** I thank Matt Lee and John Resig for their help in defining our proposed criterion, and David Parunakian for bringing the problem to my attention.

- END OF CHAPTER

# Chapter XXXIII: Giving the Software Field Protection from Patents

*by Richard Stallman*

Patents threaten every software developer, and the patent wars we have long feared have broken out. Software developers and software users—which, in our society, is most people—need software to be free of patents.

The patents that threaten us are often called “software patents,” but that term is misleading. Such patents are not about any specific program. Rather, each patent describes some practical idea, and says that anyone carrying out the idea can be sued. So it is clearer to call them “computational idea patents.”

The US patent system doesn't label patents to say this one's a “software patent” and that one isn't. Software developers are the ones who make a distinction between the patents that threaten us—those that cover ideas that can be implemented in software—and the rest. For example, if the patented idea is the shape of a physical structure or a chemical reaction, no program can implement that idea; that patent doesn't threaten the software field. But if the idea that's patented is a computation, that patent's barrel points at software developers and users.

This is not to say that computational idea patents prohibit only software. These ideas can also be implemented in hardware—and many of them have been. Each patent typically covers both hardware *and* software implementations of the idea.

## The Special Problem of Software

Still, software is where computational idea patents cause a special problem. In software, it's easy to implement thousands of ideas together in one program. If 10 percent are patented, that means hundreds of patents threaten it.

When Dan Ravicher of the Public Patent Foundation studied one large program (Linux, which is the kernel of the GNU/Linux operating system) in 2004, he found 283 US patents that appeared to cover computing ideas implemented in the source code of that program. That same year, a magazine estimated that Linux was .25 percent of the whole GNU/Linux system. Multiplying 300 by 400 we get the order-of-magnitude estimate that the system as a whole was *threatened by around 100,000 patents*.

If half of those patents were eliminated as “bad quality”—mistakes of the patent system, that is—it would not really change things. Whether 100,000 patents or 50,000, it's the same disaster. This is why it's a mistake to limit our criticism of software patents to just “patent trolls” or “bad quality” patents.

The worst patent aggressor today is Apple, which isn't a "troll" by the usual definition; I don't know whether Apple's patents are "good quality," but the better the patent's "quality" the more dangerous its threat.

We need to fix the whole problem, not just part of it.

The usual suggestions for correcting this problem legislatively involve changing the criteria for granting patents—for instance, to ban issuance of patents on computational practices and systems to perform them. This approach has two drawbacks.

First, patent lawyers are clever at reformulating patents to fit whatever rules may apply; they transform any attempt at limiting the substance of patents into a requirement of mere form. For instance, many US computational idea patents describe a system including an arithmetic unit, an instruction sequencer, a memory, plus controls to carry out a particular computation. This is a peculiar way of describing a computer running a program that does a certain computation; it was designed to make the patent application satisfy criteria that the US patent system was believed for a time to require.

Second, the US already has many thousands of computational idea patents, and changing the criteria to prevent issuing more would not get rid of the existing ones. We would have to wait almost 20 years for the problem to be entirely corrected through the expiration of these patents. We could envision legislating the abolition of these existing patents, but that is probably unconstitutional. (The Supreme Court has perversely insisted that Congress can extend private privileges at the expense of the public's rights but that it can't go in the other direction.)

## A Different Approach: Limit Effect, Not Patentability

My suggestion is to change the *effect* of patents. We should legislate that developing, distributing, or running a program on generally used computing hardware does not constitute patent infringement. This approach has several advantages:

- It does not require classifying patents or patent applications as "software" or "not software."
- It provides developers and users with protection from both existing and potential future computational idea patents.
- Patent lawyers cannot defeat the intended effect by writing applications differently.

This approach doesn't entirely invalidate existing computational idea patents, because they would continue to apply to implementations using special-purpose hardware. This is an advantage because it eliminates an argument against the legal validity of the plan. The US passed a law some years ago shielding surgeons from patent lawsuits, so that even if surgical procedures are patented, surgeons are safe. That provides a precedent for this solution.

Software developers and software users need protection from patents. This is the only legislative solution that would provide full protection for all. We could then go back to competing or cooperating... without the fear that some stranger will wipe away our work.

- END OF CHAPTER

# Chapter XXXIV: Misinterpreting Copyright—A Series of Errors

*by Richard Stallman*

Something strange and dangerous is happening in copyright law. Under the US Constitution, copyright exists to benefit users—those who read books, listen to music, watch movies, or run software—not for the sake of publishers or authors. Yet even as people tend increasingly to reject and disobey the copyright restrictions imposed on them "for their own benefit," the US government is adding more restrictions, and trying to frighten the public into obedience with harsh new penalties.

How did copyright policies come to be diametrically opposed to their stated purpose? And how can we bring them back into alignment with that purpose? To understand, we should start by looking at the root of United States copyright law: the US Constitution.

**Copyright in the US Constitution**

When the US Constitution was drafted, the idea that authors were entitled to a copyright monopoly was proposed—and rejected. The founders of our country adopted a different premise, that copyright is not a natural right of authors, but an artificial concession made to them for the sake of progress. The Constitution gives permission for a copyright system with this paragraph (Article I, Section 8):

[Congress shall have the power] to promote the Progress of Science and the useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries.

The Supreme Court has repeatedly affirmed that promoting progress means benefit for the users of copyrighted works. For example, in *Fox Film v. Doyal*, the court said,

The sole interest of the United States and the primary object in conferring the [copyright] monopoly lie in the general benefits derived by the public from the labors of authors.

This fundamental decision explains why copyright is not **required** by the Constitution, only **permitted** as an option—and why it is supposed to last for "limited times." If copyright were a natural right, something that authors have because they deserve it, nothing could justify terminating this right after a certain period of time, any more than everyone's house should become public property after a certain lapse of time from its construction.

**The “copyright bargain”**

The copyright system works by providing privileges and thus benefits to publishers and authors; but it does not do this for their sake. Rather, it does this to modify their behavior: to provide an incentive for authors to write more and publish more. In effect, the government spends the public's natural rights, on the public's behalf, as part of a deal to bring the public more published works. Legal scholars call this concept the “copyright bargain.” It is like a government purchase of a highway or an airplane using taxpayers' money, except that the government spends our freedom instead of our money.

But is the bargain as it exists actually a good deal for the public? Many alternative bargains are possible; which one is best? Every issue of copyright policy is part of this question. If we misunderstand the nature of the question, we will tend to decide the issues badly.

The Constitution authorizes granting copyright powers to authors. In practice, authors typically cede them to publishers; it is usually the publishers, not the authors, who exercise these powers and get most of the benefits, though authors may get a small portion. Thus it is usually the publishers that lobby to increase copyright powers. To better reflect the reality of copyright rather than the myth, this article refers to publishers rather than authors as the holders of copyright powers. It also refers to the users of copyrighted works as “readers,” even though using them does not always mean reading, because “the users” is remote and abstract.

**The first error: “striking a balance”**

The copyright bargain places the public first: benefit for the reading public is an end in itself; benefits (if any) for publishers are just a means toward that end. Readers' interests and publishers' interests are thus qualitatively unequal in priority. The first step in misinterpreting the purpose of copyright is the elevation of the publishers to the same level of importance as the readers.

It is often said that US copyright law is meant to “strike a balance” between the interests of publishers and readers. Those who cite this interpretation present it as a restatement of the basic position stated in the Constitution; in other words, it is supposed to be equivalent to the copyright bargain.

But the two interpretations are far from equivalent; they are different conceptually, and different in their implications. The balance concept assumes that the readers' and publishers' interests differ in importance only quantitatively, in *how much weight* we should give them, and in what actions they apply to. The term “stakeholders” is often used to frame the issue in this way; it assumes that all kinds of interest in a policy decision are equally important. This view rejects the qualitative distinction between the readers' and publishers' interests which is at the root of the government's participation in the copyright bargain.

The consequences of this alteration are far-reaching, because the great protection for the public in the copyright bargain—the idea that copyright privileges can be justified only in the name of the readers, never in the name of the publishers—is discarded by the "balance" interpretation. Since the interest of the publishers is regarded as an end in itself, it can justify copyright privileges; in other words, the "balance" concept says that privileges can be justified in the name of someone other than the public.

As a practical matter, the consequence of the "balance" concept is to reverse the burden of justification for changes in copyright law. The copyright bargain places the burden on the publishers to convince the readers to cede certain freedoms. The concept of balance reverses this burden, practically speaking, because there is generally no doubt that publishers will benefit from additional privilege. Unless harm to the readers can be proved, sufficient to "outweigh" this benefit, we are led to conclude that the publishers are entitled to almost any privilege they request.

Since the idea of "striking a balance" between publishers and readers denies the readers the primacy they are entitled to, we must reject it.

**Balancing against what?**

When the government buys something for the public, it acts on behalf of the public; its responsibility is to obtain the best possible deal—best for the public, not for the other party in the agreement.

For example, when signing contracts with construction companies to build highways, the government aims to spend as little as possible of the public's money. Government agencies use competitive bidding to push the price down.

As a practical matter, the price cannot be zero, because contractors will not bid that low. Although not entitled to special consideration, they have the usual rights of citizens in a free society, including the right to refuse disadvantageous contracts; even the lowest bid will be high enough for some contractor to make money. So there is indeed a balance, of a kind. But it is not a deliberate balancing of two interests each with claim to special consideration. It is a balance between a public goal and market forces. The government tries to obtain for the taxpaying motorists the best deal they can get in the context of a free society and a free market.

In the copyright bargain, the government spends our freedom instead of our money. Freedom is more precious than money, so government's responsibility to spend our freedom wisely and frugally is even greater than its responsibility to spend our money thus. Governments must never put the publishers' interests on a par with the public's freedom.

**Not "balance" but "trade-off"**

The idea of balancing the readers' interests against the publishers' is the wrong way to judge copyright policy, but there are indeed two interests to be weighed: two interests **of the readers**. Readers have an

interest in their own freedom in using published works; depending on circumstances, they may also have an interest in encouraging publication through some kind of incentive system.

The word "balance," in discussions of copyright, has come to stand as shorthand for the idea of "striking a balance" between the readers and the publishers. Therefore, to use the word "balance" in regard to the readers' two interests would be confusing.[1] We need another term.

In general, when one party has two goals that partly conflict, and cannot completely achieve both of them, we call this a "trade-off." Therefore, rather than speaking of "striking the right balance" between parties, we should speak of "finding the right trade-off between spending our freedom and keeping it."

**The second error: maximizing one output**

The second mistake in copyright policy consists of adopting the goal of maximizing—not just increasing—the number of published works. The erroneous concept of "striking a balance" elevated the publishers to parity with the readers; this second error places them far above the readers.

When we purchase something, we do not generally buy the whole quantity in stock or the most expensive model. Instead we conserve funds for other purchases, by buying only what we need of any particular good, and choosing a model of sufficient rather than highest quality. The principle of diminishing returns suggests that spending all our money on one particular good is likely to be an inefficient allocation of resources; we generally choose to keep some money for another use.

Diminishing returns applies to copyright just as to any other purchase. The first freedoms we should trade away are those we miss the least, and whose sacrifice gives the largest encouragement to publication. As we trade additional freedoms that cut closer to home, we find that each trade is a bigger sacrifice than the last, while bringing a smaller increment in literary activity. Well before the increment becomes zero, we may well say it is not worth its incremental price; we would then settle on a bargain whose overall result is to increase the amount of publication, but not to the utmost possible extent.

Accepting the goal of maximizing publication rejects all these wiser, more advantageous bargains in advance—it dictates that the public must cede nearly all of its freedom to use published works, for just a little more publication.

**The rhetoric of maximization**

In practice, the goal of maximizing publication regardless of the cost to freedom is supported by widespread rhetoric which asserts that public copying is illegitimate, unfair, and intrinsically wrong. For instance, the publishers call people who copy "pirates," a smear term designed to equate sharing information with your neighbor with attacking a ship. (This smear term was formerly used by authors to describe publishers who found lawful ways to publish unauthorized editions; its modern use by the

publishers is almost the reverse.) This rhetoric directly rejects the constitutional basis for copyright, but presents itself as representing the unquestioned tradition of the American legal system.

The "pirate" rhetoric is typically accepted because it so pervades the media that few people realize how radical it is. It is effective because if copying by the public is fundamentally illegitimate, we can never object to the publishers' demand that we surrender our freedom to do so. In other words, when the public is challenged to show why publishers should not receive some additional power, the most important reason of all—"We want to copy"—is disqualified in advance.

This leaves no way to argue against increasing copyright power except using side issues. Hence, opposition to stronger copyright powers today almost exclusively cites side issues, and never dares cite the freedom to distribute copies as a legitimate public value.

As a practical matter, the goal of maximization enables publishers to argue that "A certain practice is reducing our sales—or we think it might—so we presume it diminishes publication by some unknown amount, and therefore it should be prohibited." We are led to the outrageous conclusion that the public good is measured by publishers' sales: What's good for General Media is good for the USA.

**The third error: maximizing publishers' power**

Once the publishers have obtained assent to the policy goal of maximizing publication output at any cost, their next step is to infer that this requires giving them the maximum possible powers—making copyright cover every imaginable use of a work, or applying some other legal tool such as "shrink wrap" licenses to equivalent effect. This goal, which entails the abolition of "fair use" and the "right of first sale," is being pressed at every available level of government, from states of the US to international bodies.

This step is erroneous because strict copyright rules obstruct the creation of useful new works. For instance, Shakespeare borrowed the plots of some of his plays from works others had published a few decades before, so if today's copyright law had been in effect, his plays would have been illegal.

Even if we wanted the highest possible rate of publication, regardless of cost to the public, maximizing publishers' power is the wrong way to get it. As a means of promoting progress, it is self-defeating.

**The results of the three errors**

The current trend in copyright legislation is to hand publishers broader powers for longer periods of time. The conceptual basis of copyright, as it emerges distorted from the series of errors, rarely offers a basis for saying no. Legislators give lip service to the idea that copyright serves the public, while in fact giving publishers whatever they ask for.

For example, here is what Senator Hatch said when introducing S. 483, a 1995 bill to increase the term of copyright by 20 years:

I believe we are now at such a point with respect to the question of whether the current term of copyright adequately protects the interests of authors and the related question of whether the term of protection continues to provide a sufficient incentive for the creation of new works of authorship.

This bill extended the copyright on already published works written since the 1920s. This change was a giveaway to publishers with no possible benefit to the public, since there is no way to retroactively increase now the number of books published back then. Yet it cost the public a freedom that is meaningful today—the freedom to redistribute books from that era. Note the use of the propaganda term, "protect," which embodies the second of the three errors.

The bill also extended the copyrights of works yet to be written. For works made for hire, copyright would last 95 years instead of the present 75 years. Theoretically this would increase the incentive to write new works; but any publisher that claims to need this extra incentive should be required substantiate the claim with projected balance sheets for 75 years in the future.

Needless to say, Congress did not question the publishers' arguments: a law extending copyright was enacted in 1998. It was officially called the Sonny Bono Copyright Term Extension Act, named after one of its sponsors who died earlier that year. We usually call it the Mickey Mouse Copyright Act, since we presume its real motive was to prevent the copyright on the appearance of Mickey Mouse from expiring. Bono's widow, who served the rest of his term, made this statement:

Actually, Sonny wanted the term of copyright protection to last forever. I am informed by staff that such a change would violate the Constitution. I invite all of you to work with me to strengthen our copyright laws in all of the ways available to us. As you know, there is also Jack Valenti's proposal for term to last forever less one day. Perhaps the Committee may look at that next Congress.

The Supreme Court later heard a case that sought to overturn the law on the grounds that the retroactive extension fails to serve the Constitution's goal of promoting progress. The court responded by abdicating its responsibility to judge the question; on copyright, the Constitution requires only lip service.

Another law, passed in 1997, made it a felony to make sufficiently many copies of any published work, even if you give them away to friends just to be nice. Previously this was not a crime in the US at all.

An even worse law, the Digital Millennium Copyright Act (DMCA), was designed to bring back what was then called "copy protection"—now known as DRM (Digital Restrictions Management)—which users already detested, by making it a crime to defeat the restrictions, or even publish information about how to defeat them. This law ought to be called the "Domination by Media Corporations Act" because it effectively offers publishers the chance to write their own copyright law. It says they can impose any restrictions whatsoever on the use of a work, and these restrictions take the force of law provided the work contains some sort of encryption or license manager to enforce them.

One of the arguments offered for this bill was that it would implement a recent treaty to increase copyright powers. The treaty was promulgated by the World Intellectual Property Organization, an organization dominated by copyright- and patent-holding interests, with the aid of pressure from the Clinton administration; since the treaty only increases copyright power, whether it serves the public interest in any country is doubtful. In any case, the bill went far beyond what the treaty required.

Libraries were a key source of opposition to this bill, especially to the aspects that block the forms of copying that are considered fair use. How did the publishers respond? Former representative Pat Schroeder, now a lobbyist for the Association of American Publishers, said that the publishers "could not live with what [the libraries were] asking for." Since the libraries were asking only to preserve part of the status quo, one might respond by wondering how the publishers had survived until the present day.

Congressman Barney Frank, in a meeting with me and others who opposed this bill, showed how far the US Constitution's view of copyright has been disregarded. He said that new powers, backed by criminal penalties, were needed urgently because the "movie industry is worried," as well as the "music industry" and other "industries." I asked him, "But is this in the public interest?" His response was telling: "Why are you talking about the public interest? These creative people don't have to give up their rights for the public interest!" The "industry" has been identified with the "creative people" it hires, copyright has been treated as its entitlement, and the Constitution has been turned upside down.

The DMCA was enacted in 1998. As enacted, it says that fair use remains nominally legitimate, but allows publishers to prohibit all software or hardware that you could practice it with. Effectively, fair use is prohibited.

Based on this law, the movie industry has imposed censorship on free software for reading and playing DVDs, and even on the information about how to read them. In April 2001, Professor Edward Felten of Princeton University was intimidated by lawsuit threats from the Recording Industry Association of America (RIAA) into withdrawing a scientific paper stating what he had learned about a proposed encryption system for restricting access to recorded music.

We are also beginning to see e-books that take away many of readers' traditional freedoms—for instance, the freedom to lend a book to your friend, to sell it to a used book store, to borrow it from a library, to buy it without giving your name to a corporate data bank, even the freedom to read it twice. Encrypted e-books generally restrict all these activities—you can read them only with special secret software designed to restrict you.

I will never buy one of these encrypted, restricted e-books, and I hope you will reject them too. If an e-book doesn't give you the same freedoms as a traditional paper book, don't accept it!

Anyone independently releasing software that can read restricted e-books risks prosecution. A Russian programmer, Dmitry Sklyarov, was arrested in 2001 while visiting the US to speak at a conference,

because he had written such a program in Russia, where it was lawful to do so. Now Russia is preparing a law to prohibit it too, and the European Union recently adopted one.

Mass-market e-books have been a commercial failure so far, but not because readers chose to defend their freedom; they were unattractive for other reasons, such as that computer display screens are not easy surfaces to read from. We can't rely on this happy accident to protect us in the long term; the next attempt to promote e-books will use "electronic paper"—book-like objects into which an encrypted, restricted e-book can be downloaded. If this paper-like surface proves more appealing than today's display screens, we will have to defend our freedom in order to keep it. Meanwhile, e-books are making inroads in niches: NYU and other dental schools require students to buy their textbooks in the form of restricted e-books.

The media companies are not satisfied yet. In 2001, Disney-funded Senator Hollings proposed a bill called the "Security Systems Standards and Certification Act" (SSSCA)[2], which would require all computers (and other digital recording and playback devices) to have government-mandated copy-restriction systems. That is their ultimate goal, but the first item on their agenda is to prohibit any equipment that can tune digital HDTV unless it is designed to be impossible for the public to "tamper with" (i.e., modify for their own purposes). Since free software is software that users can modify, we face here for the first time a proposed law that explicitly prohibits free software for a certain job. Prohibition of other jobs will surely follow. If the FCC adopts this rule, existing free software such as GNU Radio would be censored.

To block these bills and rules requires political action.[3]

**Finding the right bargain**

What is the proper way to decide copyright policy? If copyright is a bargain made on behalf of the public, it should serve the public interest above all. The government's duty when selling the public's freedom is to sell only what it must, and sell it as dearly as possible. At the very least, we should pare back the extent of copyright as much as possible while maintaining a comparable level of publication.

Since we cannot find this minimum price in freedom through competitive bidding, as we do for construction projects, how can we find it?

One possible method is to reduce copyright privileges in stages, and observe the results. By seeing if and when measurable diminutions in publication occur, we will learn how much copyright power is really necessary to achieve the public's purposes. We must judge this by actual observation, not by what publishers say will happen, because they have every incentive to make exaggerated predictions of doom if their powers are reduced in any way.

Copyright policy includes several independent dimensions, which can be adjusted separately. After we find the necessary minimum for one policy dimension, it may still be possible to reduce other dimensions of copyright while maintaining the desired publication level.

One important dimension of copyright is its duration, which is now typically on the order of a century. Reducing the monopoly on copying to ten years, starting from the date when a work is published, would be a good first step. Another aspect of copyright, which covers the making of derivative works, could continue for a longer period.

Why count from the date of publication? Because copyright on unpublished works does not directly limit readers' freedom; whether we are free to copy a work is moot when we do not have copies. So giving authors a longer time to get a work published does no harm. Authors (who generally do own the copyright prior to publication) will rarely choose to delay publication just to push back the end of the copyright term.

Why ten years? Because that is a safe proposal; we can be confident on practical grounds that this reduction would have little impact on the overall viability of publishing today. In most media and genres, successful works are very profitable in just a few years, and even successful works are usually out of print well before ten. Even for reference works, whose useful life may be many decades, ten-year copyright should suffice: updated editions are issued regularly, and many readers will buy the copyrighted current edition rather than copy a ten-year-old public domain version.

Ten years may still be longer than necessary; once things settle down, we could try a further reduction to tune the system. At a panel on copyright at a literary convention, where I proposed the ten-year term, a noted fantasy author sitting beside me objected vehemently, saying that anything beyond five years was intolerable.

But we don't have to apply the same time span to all kinds of works. Maintaining the utmost uniformity of copyright policy is not crucial to the public interest, and copyright law already has many exceptions for specific uses and media. It would be foolish to pay for every highway project at the rates necessary for the most difficult projects in the most expensive regions of the country; it is equally foolish to “pay” for all kinds of art with the greatest price in freedom that we find necessary for any one kind.

So perhaps novels, dictionaries, computer programs, songs, symphonies, and movies should have different durations of copyright, so that we can reduce the duration for each kind of work to what is necessary for many such works to be published. Perhaps movies over one hour long could have a twenty-year copyright, because of the expense of producing them. In my own field, computer programming, three years should suffice, because product cycles are even shorter than that.

Another dimension of copyright policy is the extent of fair use: some ways of reproducing all or part of a published work that are legally permitted even though it is copyrighted. The natural first step in

reducing this dimension of copyright power is to permit occasional private small-quantity noncommercial copying and distribution among individuals. This would eliminate the intrusion of the copyright police into people's private lives, but would probably have little effect on the sales of published works. (It may be necessary to take other legal steps to ensure that shrink-wrap licenses cannot be used to substitute for copyright in restricting such copying.) The experience of Napster shows that we should also permit noncommercial verbatim redistribution to the general public—when so many of the public want to copy and share, and find it so useful, only draconian measures will stop them, and the public deserves to get what it wants.

For novels, and in general for works that are used for entertainment, noncommercial verbatim redistribution may be sufficient freedom for the readers. Computer programs, being used for functional purposes (to get jobs done), call for additional freedoms beyond that, including the freedom to publish an improved version. See “Free Software Definition,” in this book, for an explanation of the freedoms that software users should have. But it may be an acceptable compromise for these freedoms to be universally available only after a delay of two or three years from the program's publication.

Changes like these could bring copyright into line with the public's wish to use digital technology to copy. Publishers will no doubt find these proposals “unbalanced”; they may threaten to take their marbles and go home, but they won't really do it, because the game will remain profitable and it will be the only game in town.

As we consider reductions in copyright power, we must make sure media companies do not simply replace it with end-user license agreements. It would be necessary to prohibit the use of contracts to apply restrictions on copying that go beyond those of copyright. Such limitations on what mass-market nonnegotiated contracts can require are a standard part of the US legal system.

**A personal note**

I am a software designer, not a legal scholar. I've become concerned with copyright issues because there's no avoiding them in the world of computer networks, such as the Internet. As a user of computers and networks for 30 years, I value the freedoms that we have lost, and the ones we may lose next. As an author, I can reject the romantic mystique of the author as semidivine creator, often cited by publishers to justify increased copyright powers for authors—powers which these authors will then sign away to publishers.

Most of this article consists of facts and reasoning that you can check, and proposals on which you can form your own opinions. But I ask you to accept one thing on my word alone: that authors like me don't deserve special power over you. If you wish to reward me further for the software or books I have written, I would gratefully accept a check—but please don't surrender your freedom in my name.

**Footnotes**

1. See Julian Sanchez's article "The Trouble with 'Balance' Metaphors" for an examination of "how the analogy between sound judgment and balancing weights may constrain our thinking in unhealthy ways."
2. Since renamed to the unpronounceable CBDTPA, for which a good mnemonic is "Consume, But Don't Try Programming Anything," but it really stands for the "Consumer Broadband and Digital Television Promotion Act."
3. If you would like to help, I recommend the Web sites DefectiveByDesign.org, publicknowledge.org and www.eff.org.

- END OF CHAPTER

# Chapter XXXV: Did You Say “Intellectual Property”? It's a Seductive Mirage

*by Richard Stallman*

*It has become fashionable to toss copyright, patents, and trademarks—three separate and different entities involving three separate and different sets of laws—plus a dozen other laws into one pot and call it “intellectual property.” The distorting and confusing term did not become common by accident. Companies that gain from the confusion promoted it. The clearest way out of the confusion is to reject the term entirely.*

According to Professor Mark Lemley, now of the Stanford Law School, the widespread use of the term “intellectual property” is a fashion that followed the 1967 founding of the World “Intellectual Property” Organization (WIPO), and only became really common in recent years. (WIPO is formally a UN organization, but in fact represents the interests of the holders of copyrights, patents, and trademarks.) Wide use dates from around 1990[124]. (Local image copy[125])

The term carries a bias that is not hard to see: it suggests thinking about copyright, patents and trademarks by analogy with property rights for physical objects. (This analogy is at odds with the legal philosophies of copyright law, of patent law, and of trademark law, but only specialists know that.) These laws are in fact not much like physical property law, but use of this term leads legislators to change them to be more so. Since that is the change desired by the companies that exercise copyright, patent and trademark powers, the bias introduced by the term “intellectual property” suits them.

The bias is reason enough to reject the term, and people have often asked me to propose some other name for the overall category—or have proposed their own alternatives (often humorous). Suggestions include IMPs, for Imposed Monopoly Privileges, and GOLEMs, for Government-Originated Legally Enforced Monopolies. Some speak of “exclusive rights regimes,” but referring to restrictions as “rights” is doublethink too.

Some of these alternative names would be an improvement, but it is a mistake to replace “intellectual property” with any other term. A different name will not address the term's deeper problem: overgeneralization. There is no such unified thing as “intellectual property”—it is a mirage. The only reason people think it makes sense as a coherent category is that widespread use of the term has misled them about the laws in question.

The term “intellectual property” is at best a catch-all to lump together disparate laws. Nonlawyers who hear one term applied to these various laws tend to assume they are based on a common principle and function similarly.

Nothing could be further from the case. These laws originated separately, evolved differently, cover different activities, have different rules, and raise different public policy issues.

For instance, copyright law was designed to promote authorship and art, and covers the details of expression of a work. Patent law was intended to promote the publication of useful ideas, at the price of giving the one who publishes an idea a temporary monopoly over it—a price that may be worth paying in some fields and not in others.

Trademark law, by contrast, was not intended to promote any particular way of acting, but simply to enable buyers to know what they are buying. Legislators under the influence of the term "intellectual property," however, have turned it into a scheme that provides incentives for advertising. And these are just three out of many laws that the term refers to.

Since these laws developed independently, they are different in every detail, as well as in their basic purposes and methods. Thus, if you learn some fact about copyright law, you'd be wise to assume that patent law is different. You'll rarely go wrong!

In practice, nearly all general statements you encounter that are formulated using "intellectual property" will be false. For instance, you'll see claims that "its" purpose is to "promote innovation," but that only fits patent law and perhaps plant variety monopolies. Copyright law is not concerned with innovation; a pop song or novel is copyrighted even if there is nothing innovative about it. Trademark law is not concerned with innovation; if I start a tea store and call it "rms tea," that would be a solid trademark even if I sell the same teas in the same way as everyone else. Trade secret law is not concerned with innovation, except tangentially; my list of tea customers would be a trade secret with nothing to do with innovation.

You will also see assertions that "intellectual property" is concerned with "creativity," but really that only fits copyright law. More than creativity is needed to make a patentable invention. Trademark law and trade secret law have nothing to do with creativity; the name "rms tea" isn't creative at all, and neither is my secret list of tea customers.

People often say "intellectual property" when they really mean some larger or smaller set of laws. For instance, rich countries often impose unjust laws on poor countries to squeeze money out of them. Some of these laws are among those called "intellectual property" laws, and others are not; nonetheless, critics of the practice often grab for that label because it has become familiar to them. By using it, they misrepresent the nature of the issue. It would be better to use an accurate term, such as "legislative colonization," that gets to the heart of the matter.

Laymen are not alone in being confused by this term. Even law professors who teach these laws are lured and distracted by the seductiveness of the term "intellectual property," and make general statements that conflict with facts they know. For example, one professor wrote in 2006:

Unlike their descendants who now work the floor at WIPO, the framers of the US constitution had a principled, procompetitive attitude to intellectual property. They knew rights might be necessary, but...they tied congress's hands, restricting its power in multiple ways.

That statement refers to Article 1, Section 8, Clause 8 of the US Constitution, which authorizes copyright law and patent law. That clause, though, has nothing to do with trademark law, trade secret law, or various others. The term "intellectual property" led that professor to make a false generalization.

The term "intellectual property" also leads to simplistic thinking. It leads people to focus on the meager commonality in form that these disparate laws have—that they create artificial privileges for certain parties—and to disregard the details which form their substance: the specific restrictions each law places on the public, and the consequences that result. This simplistic focus on the form encourages an "economistic" approach to all these issues.

Economics operates here, as it often does, as a vehicle for unexamined assumptions. These include assumptions about values, such as that amount of production matters while freedom and way of life do not, and factual assumptions which are mostly false, such as that copyrights on music supports musicians, or that patents on drugs support life-saving research.

Another problem is that, at the broad scale implicit in the term "intellectual property," the specific issues raised by the various laws become nearly invisible. These issues arise from the specifics of each law—precisely what the term "intellectual property" encourages people to ignore. For instance, one issue relating to copyright law is whether music sharing should be allowed; patent law has nothing to do with this. Patent law raises issues such as whether poor countries should be allowed to produce life-saving drugs and sell them cheaply to save lives; copyright law has nothing to do with such matters.

Neither of these issues is solely economic in nature, and their noneconomic aspects are very different; using the shallow economic overgeneralization as the basis for considering them means ignoring the differences. Putting the two laws in the "intellectual property" pot obstructs clear thinking about each one.

Thus, any opinions about "the issue of intellectual property" and any generalizations about this supposed category are almost surely foolish. If you think all those laws are one issue, you will tend to choose your opinions from a selection of sweeping overgeneralizations, none of which is any good.

Rejection of “intellectual property” is not mere philosophical recreation. The term does real harm. Apple used it to warp debate about Nebraska's “right to repair” bill[126]. The bogus concept gave Apple a way to dress up its preference for secrecy, which conflicts with its customers' rights, as a supposed principle that customers and the state must yield to.

If you want to think clearly about the issues raised by patents, or copyrights, or trademarks, or various other different laws, the first step is to forget the idea of lumping them together, and treat them as separate topics. The second step is to reject the narrow perspectives and simplistic picture the term “intellectual property” suggests. Consider each of these issues separately, in its fullness, and you have a chance of considering them well.

And when it comes to reforming WIPO, here is one proposal for changing the name and substance of WIPO[127].

- END OF CHAPTER

# Chapter XXXVI: Opposing Digital Rights Mismanagement
## (Or Digital Restrictions Management, as we now call it)

*by Richard Stallman*

In 1989, in a very different world, I wrote the first version of the GNU General Public License, a license that gives computer users freedom. The GNU GPL, of all the free software licenses, is the one that most fully embodies the values and aims of the free software movement, by ensuring the four fundamental freedoms for every user. These are freedoms to 0) run the program as you wish; 1) study the source code and change it to do what you wish; 2) make and distribute copies, when you wish; 3) and distribute modified versions, when you wish.

Any license that grants these freedoms is a free software license. The GNU GPL goes further: it protects these freedoms for all users of all versions of the program by forbidding middlemen from stripping them off. Most components of the GNU/Linux operating system, including the Linux component that was made free software in 1992, are licensed under GPL version 2, released in 1991. Now, with legal advice from Professor Eben Moglen, I am designing version 3 of the GNU GPL.

GPLv3 must cope with threats to freedom that we did not imagine in 1989. The coming generation of computers, and many products with increasingly powerful embedded computers, are being turned against us by their manufacturers before we buy them—they are designed to restrict what we can use them to do.

First, there was the TiVo. People may think of it as an appliance to record TV programs, but it contains a real computer running a GNU/Linux system. As required by the GPL, you can get the source code for the system. You can change the code, recompile and install it. But once you install a changed version, the TiVo won't run at all, because of a special mechanism designed to sabotage you. Freedom No. 1, the freedom to change the software to do what you wish, has become a sham.

Then came Treacherous Computing, promoted as “Trusted Computing,” meaning that companies can “trust” your computer to obey them instead of you. It enables network sites to tell which program you are running; if you change the program, or write your own, they will refuse to talk to you. Once again, freedom No. 1 becomes a sham.

Microsoft has a scheme, originally called Palladium, that enables an application program to “seal” data so that no other program can access it. If Disney distributes movies this way, you'll be unable to exercise your legal rights of fair use and de minimis use. If an application records your data this way, it will be the ultimate in vendor lock-in. This too destroys freedom No. 1; if modified versions of a

program cannot access the same data, you can't really change the program to do what you wish. Something like Palladium is planned for a coming version of Windows.

AACS, the "Advanced Access Content System," promoted by Disney, IBM, Microsoft, Intel, Sony, and others, aims to restrict use of HDTV recordings—and software—so they can't be used except as these companies permit. Sony was caught last year installing a "rootkit" into millions of people's computers, and not telling them how to remove it. Sony has learned its lesson: it will install the "rootkit" in your computer before you get it, and you won't be able to remove it. This plan explicitly requires devices to be "robust"—meaning you cannot change them. Its implementors will surely want to include GPL-covered software, trampling freedom No. 1. This scheme should get "AACSed," and a boycott of HD DVD and Blu-ray has already been announced[128].

Allowing a few businesses to organize a scheme to deny our freedoms for their profit is a failure of government, but so far most of the world's governments, led by the U.S., have acted as paid accomplices rather than policemen for these schemes. The copyright industry has promulgated its peculiar ideas of right and wrong so vigorously that some readers may find it hard to entertain the idea that individual freedom can trump their profits.

Facing these threats to our freedom, what should the free software community do? Some say we should give in and accept the distribution of our software in ways that don't allow modified versions to function, because this will make our software more popular. Some refer to free software as "open source," that being the slogan of an amoral approach to the matter, which cites powerful and reliable software as the highest goals. If we allow companies to use our software to restrict us, this "open source DRM" could help them restrict us more powerfully and reliably. Those who wield the power could benefit by sharing and improving the source code of the software they use to do so. We too could read that source code—read it and weep, if we can't make a changed version run. For the goals of freedom and community—the goals of the free software movement—this concession would amount to failure.

We developed the GNU operating system so that we could control our own computers, and cooperate freely in using them in freedom. To seek popularity for our software by ceding this freedom would defeat the purpose; at best, we might flatter our egos. Therefore we have designed version 3 of the GNU GPL to uphold the user's freedom to modify the source code and put modified versions to real use.

The debate about the GPL v3 is part of a broader debate about DRM versus your rights. The motive for DRM schemes is to increase profits for those who impose them, but their profit is a side issue when millions of people's freedom is at stake; desire for profit, though not wrong in itself, cannot justify denying the public control over its technology. Defending freedom means thwarting DRM.

# Chapter XXXVII: Can You Trust Your Computer?

*by Richard Stallman*

Who should your computer take its orders from? Most people think their computers should obey them, not obey someone else. With a plan they call "trusted computing," large media corporations (including the movie companies and record companies), together with computer companies such as Microsoft and Intel, are planning to make your computer obey them instead of you. (Microsoft's version of this scheme is called Palladium.) Proprietary programs have included malicious features before, but this plan would make it universal.

Proprietary software means, fundamentally, that you don't control what it does; you can't study the source code, or change it. It's not surprising that clever businessmen find ways to use their control to put you at a disadvantage. Microsoft has done this several times: one version of Windows was designed to report to Microsoft all the software on your hard disk; a recent "security" upgrade in Windows Media Player required users to agree to new restrictions. But Microsoft is not alone: the KaZaa music-sharing software is designed so that KaZaa's business partner can rent out the use of your computer to its clients. These malicious features are often secret, but even once you know about them it is hard to remove them, since you don't have the source code.

In the past, these were isolated incidents. "Trusted computing" would make the practice pervasive. "Treacherous computing" is a more appropriate name, because the plan is designed to make sure your computer will systematically disobey you. In fact, it is designed to stop your computer from functioning as a general-purpose computer. Every operation may require explicit permission.

The technical idea underlying treacherous computing is that the computer includes a digital encryption and signature device, and the keys are kept secret from you. Proprietary programs will use this device to control which other programs you can run, which documents or data you can access, and what programs you can pass them to. These programs will continually download new authorization rules through the Internet, and impose those rules automatically on your work. If you don't allow your computer to obtain the new rules periodically from the Internet, some capabilities will automatically cease to function.

Of course, Hollywood and the record companies plan to use treacherous computing for Digital Restrictions Management (DRM), so that downloaded videos and music can be played only on one specified computer. Sharing will be entirely impossible, at least using the authorized files that you would get from those companies. You, the public, ought to have both the freedom and the ability to share these things. (I expect that someone will find a way to produce unencrypted versions, and to upload and share them, so DRM will not entirely succeed, but that is no excuse for the system.)

Making sharing impossible is bad enough, but it gets worse. There are plans to use the same facility for email and documents—resulting in email that disappears in two weeks, or documents that can only be read on the computers in one company.

Imagine if you get an email from your boss telling you to do something that you think is risky; a month later, when it backfires, you can't use the email to show that the decision was not yours. “Getting it in writing” doesn't protect you when the order is written in disappearing ink.

Imagine if you get an email from your boss stating a policy that is illegal or morally outrageous, such as to shred your company's audit documents, or to allow a dangerous threat to your country to move forward unchecked. Today you can send this to a reporter and expose the activity. With treacherous computing, the reporter won't be able to read the document; her computer will refuse to obey her. Treacherous computing becomes a paradise for corruption.

Word processors such as Microsoft Word could use treacherous computing when they save your documents, to make sure no competing word processors can read them. Today we must figure out the secrets of Word format by laborious experiments in order to make free word processors read Word documents. If Word encrypts documents using treacherous computing when saving them, the free software community won't have a chance of developing software to read them—and if we could, such programs might even be forbidden by the Digital Millennium Copyright Act.

Programs that use treacherous computing will continually download new authorization rules through the Internet, and impose those rules automatically on your work. If Microsoft, or the US government, does not like what you said in a document you wrote, they could post new instructions telling all computers to refuse to let anyone read that document. Each computer would obey when it downloads the new instructions. Your writing would be subject to 1984-style retroactive erasure. You might be unable to read it yourself.

You might think you can find out what nasty things a treacherous-computing application does, study how painful they are, and decide whether to accept them. Even if you can find this out, it would be foolish to accept the deal, but you can't even expect the deal to stand still. Once you come to depend on using the program, you are hooked and they know it; then they can change the deal. Some applications will automatically download upgrades that will do something different—and they won't give you a choice about whether to upgrade.

Today you can avoid being restricted by proprietary software by not using it. If you run GNU/Linux or another free operating system, and if you avoid installing proprietary applications on it, then you are in charge of what your computer does. If a free program has a malicious feature, other developers in the community will take it out, and you can use the corrected version. You can also run free application

programs and tools on nonfree operating systems; this falls short of fully giving you freedom, but many users do it.

Treacherous computing puts the existence of free operating systems and free applications at risk, because you may not be able to run them at all. Some versions of treacherous computing would require the operating system to be specifically authorized by a particular company. Free operating systems could not be installed. Some versions of treacherous computing would require every program to be specifically authorized by the operating system developer. You could not run free applications on such a system. If you did figure out how, and told someone, that could be a crime.

There are proposals already for US laws that would require all computers to support treacherous computing, and to prohibit connecting old computers to the Internet. The CBDTPA (we call it the Consume But Don't Try Programming Act) is one of them. But even if they don't legally force you to switch to treacherous computing, the pressure to accept it may be enormous. Today people often use Word format for communication, although this causes several sorts of problems (see "We Can Put an End to Word Attachments"[129]). If only a treacherous-computing machine can read the latest Word documents, many people will switch to it, if they view the situation only in terms of individual action (take it or leave it). To oppose treacherous computing, we must join together and confront the situation as a collective choice.

For further information about treacherous computing, see the "Trusted Computing" Frequently Asked Questions[130].

To block treacherous computing will require large numbers of citizens to organize. We need your help! Please support Defective by Design, the FSF's campaign against Digital Restrictions Management.

**Postscripts**

1. The computer security field uses the term "trusted computing" in a different way—beware of confusion between the two meanings.

2. The GNU Project distributes the GNU Privacy Guard, a program that implements public-key encryption and digital signatures, which you can use to send secure and private email. It is useful to explore how GPG differs from treacherous computing, and see what makes one helpful and the other so dangerous.
   When someone uses GPG to send you an encrypted document, and you use GPG to decode it, the result is an unencrypted document that you can read, forward, copy, and even reencrypt to send it securely to someone else. A treacherous-computing application would let you read the words on the screen, but would not let you produce an unencrypted document that you could

use in other ways. GPG, a free software package, makes security features available to the users; *they* use *it*. Treacherous computing is designed to impose restrictions on the users; *it* uses *them*.

3. The supporters of treacherous computing focus their discourse on its beneficial uses. What they say is often correct, just not important.
   Like most hardware, treacherous-computing hardware can be used for purposes which are not harmful. But these features can be implemented in other ways, without treacherous-computing hardware. The principal difference that treacherous computing makes for users is the nasty consequence: rigging your computer to work against you.
   What they say is true, and what I say is true. Put them together and what do you get? Treacherous computing is a plan to take away our freedom, while offering minor benefits to distract us from what we would lose.

4. Microsoft presents Palladium as a security measure, and claims that it will protect against viruses, but this claim is evidently false. A presentation by Microsoft Research in October 2002 stated that one of the specifications of Palladium is that existing operating systems and applications will continue to run; therefore, viruses will continue to be able to do all the things that they can do today.
   When Microsoft employees speak of "security" in connection with Palladium, they do not mean what we normally mean by that word: protecting your machine from things you do not want. They mean protecting your copies of data on your machine from access by you in ways others do not want. A slide in the presentation listed several types of secrets Palladium could be used to keep, including "third party secrets" and "user secrets"—but it put "user secrets" in quotation marks, recognizing that this is somewhat of an absurdity in the context of Palladium.
   The presentation made frequent use of other terms that we frequently associate with the context of security, such as "attack," "malicious code," "spoofing," as well as "trusted." None of them means what it normally means. "Attack" doesn't mean someone trying to hurt you, it means you trying to copy music. "Malicious code" means code installed by you to do what someone else doesn't want your machine to do. "Spoofing" doesn't mean someone's fooling you, it means you're fooling Palladium. And so on.

5. A previous statement by the Palladium developers stated the basic premise that whoever developed or collected information should have total control of how you use it. This would represent a revolutionary overturn of past ideas of ethics and of the legal system, and create an unprecedented system of control. The specific problems of these systems are no accident; they result from the basic goal. It is the goal we must reject.

As of 2015, the main method of distributing copies of anything is over the internet, and specifically over the web. Nowadays, the companies that want to impose DRM on the world want it to be enforced by programs that talk to web servers to get copies. This means that they are determined to

control your browser as well as your operating system. The way they do this is through "remote attestation"—a facility with which your computer can "attest" to the web server precisely what software it is running, such that there is no way you can disguise it. The software it would attest to would include the web browser (to prove it implements DRM and gives you no way to extract the unencrypted data), the kernel (to prove it gives no way to patch the running browser), the boot software (to prove it gives no way to patch the kernel when starting it), and anything else relating to the security of the DRM companies' dominion over you.

Under an evil empire, the only crack by which you can reduce its effective power over you is to have a way to hide or disguise what you are doing. In other words, you need a way to lie to the empire's secret police. "Remote attestation" is a plan to force your computer to tell the truth to a company when its web server asks the computer whether you have liberated it.

As of 2015, treacherous computing has been implemented for PCs in the form of the "Trusted Platform Module"; however, for practical reasons, the TPM has proved a total failure for the goal of providing a platform for remote attestation to verify Digital Restrictions Management. Thus, companies implement DRM using other methods. At present, "Trusted Platform Modules" are not being used for DRM at all, and there are reasons to think that it will not be feasible to use them for DRM. Ironically, this means that the only current uses of the "Trusted Platform Modules" are the innocent secondary uses—for instance, to verify that no one has surreptitiously changed the system in a computer.

Therefore, we conclude that the "Trusted Platform Modules" available for PCs as of 2015 are not dangerous, and there is no *immediate* reason not to include one in a computer or support it in system software.

This does not mean that everything is rosy. Other hardware systems for blocking the owner of a computer from changing the software in it are in use in some ARM PCs as well as processors in portable phones, cars, TVs and other devices, and these are fully as bad as we expected.

This also does not mean that remote attestation is not a threat. If ever a device succeeds in implementing that, it will be a grave threat to users' freedom. The current "Trusted Platform Module" is harmless only because it failed in the attempt to make remote attestation feasible. We must not presume that all future attempts will fail too.

As of 2022, the TPM2, a new "Trusted Platform Module", really does support remote attestation and can support DRM. The threat I warned about in 2002 has become terrifyingly real.

Remote attestation is actually in use by "Google SafetyNet[131]" (now part of the "Play Integrity API[132]"), which verifies that the Android operating system running in a snoop-phone is an official Google version.

This malicious functionality already makes it impossible to run some bank apps on GrapheneOS[133], which is a modified version of Android that eliminates some, though not all, of the nonfree software that Android normally contains.

A free version of Android, such as Replicant, would surely encounter the same obstacle. If you value your freedom enough to install Replicant, you might also refuse to tolerate any nonfree app (banking or not) on your computers. It is nonetheless unjust for Google to snoop on whether users have modified their operating system and dictate based on that what users can do with it.

- END OF CHAPTER

# Chapter XXXVIII: Who Does That Server Really Serve?

*by Richard Stallman*

*On the Internet, proprietary software isn't the only way to lose your computing freedom. Service as a Software Substitute, or SaaSS, is another way to give someone else power over your computing.*

The basic point is, you can have control over a program someone else wrote (if it's free), but you can never have control over a service someone else runs, so never use a service where in principle running a program would do.

SaaSS means using a service implemented by someone else as a substitute for running your copy of a program. The term is ours; articles and ads won't use it, and they won't tell you whether a service is SaaSS. Instead they will probably use the vague and distracting term "cloud," which lumps SaaSS together with various other practices, some abusive and some OK. And they talk about "delivering a program by offering a service to run it." With the explanation and examples in this page, you can tell whether a service is SaaSS.

**Background: How Proprietary Software Takes Away Your Freedom**

Digital technology can give you freedom; it can also take your freedom away. The first threat to our control over our computing came from *proprietary software*: software that the users cannot control because the owner (a company such as Apple or Microsoft) controls it. The owner often takes advantage of this unjust power by inserting malicious features such as spyware, back doors, and Digital Restrictions Management (DRM)[134](referred to as "Digital Rights Management" in their propaganda).

Our solution to this problem is developing *free software* and rejecting proprietary software. Free software gives you, as a user, four essential freedoms: (0) to run the program as you wish, (1) to study and change the source code so it does what you wish, (2) to redistribute exact copies, and (3) to redistribute copies of your modified versions. (See the free software definition.)

With free software, we, the users, take back control of our computing. Proprietary software still exists, but we can exclude it from our lives and many of us have done so. However, we are now offered another tempting way to cede control over our computing: Service as a Software Substitute (SaaSS). For our freedom's sake, we have to reject that too.

**What Does Service as a Software Substitute Look Like?**

Service as a Software Substitute (SaaSS) means using a service as a substitute for running your copy of a program. Concretely, it means that someone sets up a network server that does certain computing activities—for instance, modifying a photo, translating text into another language, etc.—then invites users to let that server do*their own computing* for them. As a user of the server, you would send your data to the server, which does that computing activity on the data thus provided, then sends the results back to you or else acts directly on your behalf.

**To Which Activities Is the Issue of SaaSS Applicable?**

The issue of SaaSS-or-not-SaaSS is meaningful for a computing activity that is *your own* computing. What does that mean, precisely? It means that no one else is inherently involved in the activity. To clarify the meaning of "inherently involved," we present a thought experiment in which we focus on one unspecified imaginary computing activity.

Suppose that all parts of the activity are implemented in free software and you have copies, and you have whatever data you might need, as well as computers of whatever speed, functionality and capacity might be required. Could you (if given those prerequisites) do this particular computing activity entirely within those computers, not communicating with anyone else's computers?

If you could, then the activity is *essentially your own*. Therefore, for your freedom's sake, you deserve to control it. The concept of SaaSS is applicable to such activities and not to other activities.

For such an activity, if you carry it out by running your copies of free programs, you do control it. That protects the freedom you deserve. However, doing it via someone else's service would give that someone else control over part of your computing activity. That denies you the control you deserve, so we say it is unjust. We call that scenario SaaSS.

By contrast, if due to the inherent nature of the computing to be done you couldn't possibly do that activity entirely in your own computers, then the activity isn't entirely your own, so the issue of SaaSS is not applicable to that activity. In general, these activities involve communication with others, so the others must be included in it. Buying something from a store is a typical example of an activity that needs to include some other party (the store).

If a certain activity is essentially your own, then maintaining your full control over it requires that you do it using your copies of free programs, running them on computers you control. Doing it in any other way is SaaSS because it denies you the control you deserve. This is independent of your reasons for doing it in some other way. If you choose some other way because of some convenience, it is SaaSS.

If it is because you can't obtain the free programs or the computer you'd need to keep control, that is still SaaSS.

**Using SaaSS Compared with Running Nonfree Software**

SaaSS servers wrest control from the users even more inexorably than proprietary software. With proprietary software, users typically get an executable file but not the source code. That makes it hard to study the code that is running, so it's hard to determine what the program really does, and hard to change it.

With SaaSS, the users do not have even the executable file that does their computing: it is on someone else's server, where the users can't see or touch it. Thus it is impossible for them to ascertain what it really does, and impossible to change it.

Furthermore, SaaSS automatically leads to consequences equivalent to the malicious features of certain proprietary software.

For instance, some proprietary programs are “spyware”: the program sends out data about users' computing activities[135]. Microsoft Windows sends information about users' activities to Microsoft. Windows Media Player reports what each user watches or listens to. The Amazon Kindle reports which pages of which books the user looks at, and when. Angry Birds reports the user's geolocation history.

Unlike proprietary software, SaaSS does not require covert code to obtain the user's data. Instead, its structure requires users to send their data to the server in order to use it. This has the same effect as spyware: the server operator gets the data—with no special effort, by the nature of SaaSS. Amy Webb, who intended never to post any photos of her daughter, made the mistake of using SaaSS (Instagram) to edit photos of her. Eventually they leaked from there[136].

Theoretically, homomorphic encryption might some day advance to the point where future SaaSS services might be constructed to be unable to understand some of the data that users send them. Such services *could* be set up not to snoop on users; this does not mean they *will* do no snooping. Also, snooping is only one among the secondary injustices of SaaSS.

Some proprietary operating systems have a universal back door, permitting someone to remotely install software changes. For instance, Windows has a universal back door with which Microsoft can forcibly change any software on the machine. Nearly all portable phones have them, too. Some proprietary applications also have universal back doors; for instance, the Steam client for GNU/Linux allows the developer to remotely install modified versions.

With SaaSS, the server operator can change the software in use on the server. Person ought to be able to do this, since it's per computer; but the result is the same as using a proprietary application program with a universal back door: someone has the power to silently impose changes in how the user's computing gets done.

It is common for SaaSS dis-services to charge a monthly fee for use. Usually one SaaSS site does not substitute for another, so if users become unhappy with one dis-service provider it is no easy matter to switch to another. When users become dependent on one, it can gouge them at will with repeated small price increases that over time add up to a lot[137]. We view the loss of freedom inherent in SaaSS as worse than the cost in money, but when a dis-service has you over a barrel, the cost can be painful. Thus, even users who don't see deeper than the bottom line should beware of SaaSS.

SaaSS is equivalent to running proprietary software with spyware and a universal back door. It gives the server operator unjust power over the user, and unjust power is something we must resist.

**SaaSS and SaaS**

Originally we referred to this problematical practice as "SaaS," which stands for "Software as a Service." It's a commonly used term for setting up software on a server rather than offering copies of it to users, and we thought it described precisely the cases where this problem occurs.

Subsequently we became aware that the term SaaS is sometimes used for communication services—activities for which this issue is not applicable. In addition, the term "Software as a Service" doesn't explain *why* the practice is bad. So we coined the term "Service as a Software Substitute," which defines the bad practice more clearly and says what is bad about it.

**Untangling the SaaSS Issue from the Proprietary Software Issue**

SaaSS and proprietary software lead to similar harmful results, but the mechanisms are different. With proprietary software, the mechanism is that you have and use a copy which is difficult and/or illegal to change. With SaaSS, the mechanism is that you don't have the copy that's doing your computing.

These two issues are often confused, and not only by accident. Web developers use the vague term "web application" to lump the server software together with programs run on your machine in your browser. Some web pages install nontrivial, even large JavaScript programs into your browser without informing you. When these JavaScript programs are nonfree, they cause the same sort of injustice as any other nonfree software. Here, however, we are concerned with the issue of using the service itself.

Many free software supporters assume that the problem of SaaSS will be solved by developing free software for servers. For the server operator's sake, the programs on the server had better be free; if they

are proprietary, their developers/owners have power over the server. That's unfair to the server operator, and doesn't help the server's users at all. But if the programs on the server are free, that doesn't protect *the server's users* from the effects of SaaSS. These programs liberate the server operator, but not the server's users.

Releasing the server software source code does benefit the community: it enables suitably skilled users to set up similar servers, perhaps changing the software. We recommend using the GNU Affero GPL as the license for programs often used on servers.

But none of these servers would give you control over computing you do on it, unless it's *your* server (one whose software load you control, regardless of whether the machine is your property). It may be OK to trust your friend's server for some jobs, just as you might let your friend maintain the software on your own computer. Outside of that, all these servers would be SaaSS for you. SaaSS always subjects you to the power of the server operator, and the only remedy is, *Don't use SaaSS!* Don't use someone else's server to do your own computing on data provided by you.

This issue demonstrates the depth of the difference between "open" and "free." Source code that is open source is, nearly always, free. However, the idea of an "open software" service[138], meaning one whose server software is open source and/or free, fails to address the issue of SaaSS.

Services are fundamentally different from programs, and the ethical issues that services raise are fundamentally different from the issues that programs raise. To avoid confusion, we avoid describing a service as "free" or "proprietary."

**Distinguishing SaaSS from Other Network Services**

Which online services are SaaSS? The clearest example is a translation service, which translates (say) English text into Spanish text. Translating a text for you is computing that is purely yours. You could do it by running a program on your own computer, if only you had the right program. (To be ethical, that program should be free.) The translation service substitutes for that program, so it is Service as a Software Substitute, or SaaSS. Since it denies you control over your computing, it does you wrong.

Another clear example is using a service such as Flickr or Instagram to modify a photo. Modifying photos is an activity that people have done in their own computers for decades; doing it in a server you don't control, rather than your own computer, is SaaSS.

Rejecting SaaSS does not mean refusing to use any network servers run by anyone other than you. Most servers are not SaaSS because the jobs they do are some sort of communication with visitors, rather than each visitor's own computing.

The original idea of web servers wasn't to do computing for you, a visitor; it was to publish information for you to access. Even today this is what most web sites do, and it doesn't raise the SaaSS issue, because accessing someone's published information on a web site isn't a matter of your own computing. Neither is use of a blog site to publish your own works, or using a microblogging service such as Mastodon, or StatusNet, or Ex-Twitter. (These services may or may not have other problems, depending on details.) The same goes for other communication not meant to be private, such as chat groups.

In its essence, social networking is a form of communication and publication, not SaaSS. However, a service whose main facility is social networking can have features or extensions which are SaaSS.

If a service is not SaaSS, that does not mean it is OK. There are other ethical issues about services. For instance, Facebook requires running nonfree JavaScript code, and it gives users a misleading impression of privacy while luring them into baring their lives to Facebook. Those are important issues, but distinct from the SaaSS issue.

Services such as search engines collect data from around the web and let you examine it. Looking through their collection of data isn't your own computing in the usual sense—you didn't provide that collection—so using such a service to search the web is not SaaSS. However, using someone else's server to implement a search facility for your own site *is* SaaSS.

Purchasing online is not SaaSS, because the computing isn't *your own* activity; rather, it is done jointly by and for you and the store. The real issue in online shopping is whether you trust the other party with your money and other personal information (starting with your name).

Repository sites such as Savannah and SourceForge are not inherently SaaSS, because a repository's job is publication of data supplied to it.

Some sites offer multiple services, and if one is not SaaSS, another may be SaaSS. For instance, the main service of Facebook is social networking, and that is not SaaSS; however, it supports third-party applications, some of which are SaaSS. Flickr's main service is distributing photos, which is not SaaSS, but it also has features for editing photos, which is SaaSS. Likewise, using Instagram to post a photo is not SaaSS, but using it to transform the photo is SaaSS.

Google Docs shows how complex the evaluation of a single service can become. It invites people to edit a document by running a large nonfree JavaScript program, clearly unjust, but not SaaSS. However, it offers an API for uploading and downloading documents in standard formats. A free software editor can do so through this API. (Whether it is possible to get an account for Google Docs without running some nonfree JavaScript code, we don't know.) Anyway, this usage scenario is not SaaSS, because it uses Google Docs as a mere repository. Handing your work data to a company is bad, but that is a matter of

privacy, not SaaSS; depending on a service for access to your data is bad, but that is a matter of risk, not SaaSS.

On the other hand, using Google Docs for converting document formats *is* SaaSS, because it's something you could have done by running a suitable program (free, one hopes) in your own computer.

Using Google Docs through a free editor is rare, of course. Most often, people edit their Google Docs documents with the nonfree JavaScript program it sends, which is bad like any nonfree program. This scenario might involve SaaSS, too; that depends on what part of the editing is done in the JavaScript program and what part in the server. We don't know, but since SaaSS and proprietary software do similar wrong to the user, we can judge the whole scenario morally without knowing which part is which.

Publishing via someone else's repository does not raise privacy issues, but publishing through Google Docs has a special problem: it is impossible even to *view the text* of a Google Docs document in a browser without running the nonfree JavaScript code. Thus, you should not use Google Docs to publish anything—but the reason is not a matter of SaaSS.

The IT industry discourages users from making these distinctions. That's what the buzzword “cloud computing” is for. This term is so nebulous that it could refer to almost any use of the Internet. It includes SaaSS as well as many other network usage practices. In any given context, an author who writes “cloud” (if a technical person) probably has a specific meaning in mind, but usually does not explain that in other articles the term has other specific meanings. The term leads people to generalize about practices they ought to judge separately.

If “cloud computing” has a meaning, it is not a way of doing computing, but rather a way of thinking about computing: a devil-may-care approach which says, “Don't ask questions. Don't worry about who controls your computing or who holds your data. Don't check for a hook hidden inside our service before you swallow it. Trust companies without hesitation.” In other words, “Be a sucker.” A cloud in the mind is an obstacle to clear thinking. For the sake of clear thinking about computing, let's avoid the term “cloud.”

**Renting a Server Distinguished from SaaSS**

If you rent a server (real or virtual), whose software load you have control over, that's not SaaSS. In SaaSS, someone else decides what software runs on the server and therefore controls the computing it does for you. In the case where you install the software on the server, you control what computing it does for you. Thus, the rented server is virtually your computer. For this issue, it counts as yours.

The *data* on the rented remote server is less secure than if you had the server at home, but that is a separate issue from SaaSS.

This kind of server rental is sometimes called “IaaS,” but that term fits into a conceptual structure that downplays the issues that we consider important.

**When the User Is a Collective Activity Or an Organization**

So far we have explained how SaaSS applies to an individual's computing. For those cases, we have clarified the concept of SaaSS pretty thoroughly. SaaSS is also an issue for computing done by a group activity, which may be informal (such as developing a free program often is), or formal (a charity like the FSF or a business). It is basically the same concept, but we have not clarified the boundaries for all sorts of situations.

Here are some line we have drawn so far.

The collective activity is likely to have web pages, which will be hosted on some web server. That server's treatment of visitors to its pages raises the usual moral issues: if they send nonfree JavaScript code, that is an injustice, and if they offer to do the visitor's computing, that is SaaSS.

However, the web server's own operations can also raise the issue of SaaSS with the collective activity as victim. A web server often offers visitors a way to search through the web pages; how does it implement that? If the collective activity runs a free program on its own computer to find the matches for the search string, the collective activity has control of this, as it should. But if it asks Google (or any other search engine) where the matches are and displays what is found, the collective activity is relying on SaaSS and forfeiting its freedom.

Using a joint project's servers to work on that project isn't SaaSS because the computing you do in this way isn't your own—it is the project's computing. For instance, if you edit pages on Wikipedia, you are not doing your own computing; rather, you are collaborating in Wikipedia's computing. Wikipedia controls its own servers, but organizations as well as individuals encounter the problem of SaaSS if they do their computing in someone else's server.

Use of simple software repositories is not SaaSS because most of the actual work (as distinguished from redistribution) is done in the contributors' computers. However, when the repository starts doing other kinds of computing work for the users, such as running tests, that starts to cross the line. When the users are contributing to the project, so the work is the project's work rather than the contributor's work, that still is not SaaSS for the users. But it may be SaaSS for the project. However, if the testing means running the programs that the project develops, it is not SaaSS because the project does control the crucial software being run.

**Dealing with the SaaSS Problem**

Only a small fraction of all web sites do SaaSS; most don't raise the issue. But what should we do about the ones that raise it?

For the simple case, where you are doing your own computing on data in your own hands, the solution is simple: use your own copy of a free software application. Do your text editing with your copy of a free text editor such as GNU Emacs or a free word processor. Do your photo editing with your copy of free software such as GIMP. What if there is no free program available? A proprietary program or SaaSS would take away your freedom, so you shouldn't use those. You can contribute your time or your money to development of a free replacement.

What about collaborating with other individuals as a group? It may be hard to do this at present without using a server, and your group may not know how to run its own server. If you use someone else's server, at least don't trust a server run by a company. A mere contract as a customer is no protection unless you could detect a breach and could really sue, and the company probably writes its contracts to permit a broad range of abuses. The state can subpoena your data from the company along with everyone else's, as Obama has done to phone companies, supposing the company doesn't volunteer them like the US phone companies that illegally wiretapped their customers for Bush. If you must use a server, use a server whose operators give you a basis for trust beyond a mere commercial relationship.

However, on a longer time scale, we can create alternatives to using servers. For instance, we can create a peer-to-peer program through which collaborators can share data encrypted. The free software community should develop distributed peer-to-peer replacements for important “web applications.” It may be wise to release them under the GNU Affero GPL, since they are likely candidates for being converted into server-based programs by someone else. The GNU project is looking for volunteers to work on such replacements. We also invite other free software projects to consider this issue in their design.

In the meantime, if a company invites you to use its server to do your own computing tasks, don't yield; don't use SaaSS. Don't buy or install “thin clients,” which are simply computers so weak they make you do the real work on a server, unless you're going to use them with *your* server. Use a real computer and keep your data there. Do your own computing with your own copy of a free program, for your freedom's sake.

- END OF CHAPTER

# Chapter XXXIX: Words to Avoid (or Use with Care) Because They Are Loaded or Confusing

There are a number of words and phrases that we recommend avoiding, or avoiding in certain contexts and usages. Some are ambiguous or misleading; others presuppose a viewpoint that we disagree with, and we hope you disagree with it too.

**"Ad-blocker"**

When the purpose of some program is to block advertisements, "ad-blocker" is a good term for it. However, the GNU browser IceCat blocks advertisements that track the user as consequence of broader measures to prevent surveillance by web sites. This is not an "ad-blocker," this is *surveillance protection*.

**"Access"**

It is a common misunderstanding to think free software means that the public has "access" to a program. That is not what free software means.

The criterion for free software is not about who has "access" to the program; the four essential freedoms concern what a user that has a copy of the program is allowed to do with it. For instance, freedom 2 says that that user is free to make another copy and give or sell it to you. But no user is *obligated* to do that for you; you do not have a *right* to demand a copy of that program from any user.

In particular, if you write a program yourself and never offer a copy to anyone else, that program is free software albeit in a trivial way, because every user that has a copy has the four essential freedoms (since the only such user is you).

In practice, when many users have copies of a program, someone is sure to post it on the internet, giving everyone access to it. We think people ought to do that, if the program is useful. But that isn't a requirement of free software.

There is one specific point in which a question of having access is directly pertinent to free software: the GNU GPL permits giving a particular user access to download a program's source code as a substitute for physically giving that user a copy of the source. This applies to the special case in which the user already has a copy of the program in non-source form.

Instead of **with free software, the public has access to the program**, we say, **with free software, the users have the essential freedoms** and **with free software, the users have control of what the program does for them**.

**"Alternative"**

We don't describe free software in general as an "alternative" to proprietary, because that word presumes all the "alternatives" are legitimate and each additional one makes users better off. In effect, it assumes that free software ought to coexist with software that does not respect users' freedom.

We believe that distribution as free software is the only ethical way to make software available for others to use. The other methods, nonfree software and Service as a Software Substitute subjugate their users. We do not think it is good to offer users those "alternatives" to free software.

Special circumstances can drive users toward running one particular program for a certain job. For instance, when a web page sends JavaScript client code to the user's browser, that drives users toward running that specific client program rather than any possible other. In such a case, there is a reason to describe any other code for that job as an alternative.

**"Artificial Intelligence"**

The moral panic over ChatGPT has led to confusion because people often speak of it as "artificial intelligence." Is ChatGPT properly described as artificial intelligence? Should we call it that? Professor Sussman of the MIT Artificial Intelligence Lab argues convincingly that we should not.

Normally, "intelligence" means having knowledge and understanding, at least about some kinds of things. A true artificial intelligence should have some knowledge and understanding. General artificial intelligence would be able to know and understand about all sorts of things; that does not exist, but we do have systems of limited artificial intelligence which can know and understand in certain limited fields.

By contrast, ChatGPT knows nothing and understands nothing. Its output is merely smooth babbling. Anything it states or implies about reality is fabrication (unless "fabrication" implies more understanding than that system really has). Seeking a correct answer to any real question in ChatGPT output is folly, as many have learned to their dismay.

That is not a matter of implementation details. It is an inherent limitation due to the fundamental approach these systems use[139].

Here is how we recommend using terminology for systems based on trained neural networks:

- "Artificial intelligence" is a suitable term for systems that have understanding and knowledge within some domain, whether small or large.
- "Bullshit generators" is a suitable term for large language models ("LLMs") such as ChatGPT, that generate smooth-sounding verbiage that appears to assert things about the world, without understanding that verbiage semantically. This conclusion has received support from the paper titled *ChatGPT is bullshit*[140] by Hicks et al., (2024).
- "Generative systems" is a suitable term for systems that generate artistic works for which "truth" and "falsehood" are not applicable.

Those three categories of jobs are mostly implemented, nowadays, with "machine learning systems." That means they work with data consisting of many numeric values, and adjust those numbers based on "training data." A machine learning system may be a bullshit generator, a generative system, or artificial intelligence.

Most machine learning systems today are implemented as "neural network systems" ("NNS"), meaning that they work by simulating a network of "neurons"—highly simplified models of real nerve cells. However, there are other kinds of machine learning which work differently.

There is a specific term for the neural-network systems that generate textual output which is plausible in terms of grammar and diction: "large language models" ("LLMs"). These systems cannot begin to grasp the *meanings* of their textual outputs, so they are invariably bullshit generators, never artificial intelligence.

There are systems which use machine learning to recognize specific important patterns in data. Their output can reflect real knowledge (even if not with perfect accuracy)—for instance, whether an image of tissue from an organism shows a certain medical condition, whether an insect is a bee-eating Asian hornet[141], or whether a toddler may be at risk of becoming autistic[142]. Scientists validate the output by comparing the system's judgment against experimental tests. That justifies referring to these systems as "artificial intelligence." Likewise the systems that antisocial media use to decide what to show or recommend to a user, since the companies validate that they actually understand what will increase "user engagement," even though that manipulation of users may be harmful *to them and to society as a whole*.

Businesses and governments use similar systems to evaluate how to deal with potential clients or people accused of various things. These evaluation results are often validated carelessly and the result can be systematic injustice. But since it purports to understand, it qualifies at least as attempted artificial intelligence.

As that example shows, artificial intelligence can be broken, or systematically biased, or work badly, just as natural intelligence can. Here we are concerned with whether specific instances fit that term, not with whether they do good or harm.

There are also systems of artificial intelligence which solve math problems[143], using machine learning to explore the space of possible solutions to find a valid solution. They qualify as artificial intelligence because they test the validity of a candidate solution using rigorous mathematical methods.

When bullshit generators output text that appears to make factual statements but describe nonexistent people, places, and things, or events that did not happen, it is fashionable to call those statements "hallucinations" or say that the system "made them up." That fashion spreads a conceptual confusion, because it presumes that the system has some sort of understanding of the meaning of its output, and that its understanding was mistaken *in a specific case*.

That presumption is false: these systems have no semantic understanding whatsoever.

**"Assets"**

To refer to published works as "assets," or "digital assets," is even worse than calling them "content"—it dismisses their value to society aside from commercial value.

**"BSD-style"**

The expression "BSD-style license" leads to confusion because it lumps together licenses that have important differences. For instance, the original BSD license with the advertising clause is incompatible with the GNU General Public License, but the revised BSD license is compatible with the GPL.

To avoid confusion, it is best to name the specific license in question and avoid the vague term "BSD-style."

**"Closed"**

Describing nonfree software as "closed" clearly refers to the term "open source." In the free software movement, we do not want to be confused with the open source camp, so we are careful to avoid saying things that would encourage people to lump us in with them. For instance, we avoid describing nonfree software as "closed." We call it "nonfree" or "proprietary".

**"Cloud Computing"**

The term "cloud computing" (or just "cloud," in the context of computing) is a marketing buzzword with no coherent meaning. It is used for a range of different activities whose only common

characteristic is that they use the Internet for something beyond transmitting files. Thus, the term spreads confusion. If you base your thinking on it, your thinking will be confused (or, could we say, "cloudy"?).

When thinking about or responding to a statement someone else has made using this term, the first step is to clarify the topic. What scenario is the statement about? What is a good, clear term for that scenario? Once the topic is clearly formulated, coherent thought about it becomes possible.

One of the many meanings of "cloud computing" is storing your data in online services. In most scenarios, that is foolish because it exposes you to surveillance.

Another meaning (which overlaps that but is not the same thing) is Service as a Software Substitute, which denies you control over your computing. You should never use SaaSS.

Another meaning is renting a remote physical server, or virtual server. These practices are ok under certain circumstances.

Another meaning is accessing your own server from your own mobile device. That raises no particular ethical issues.

The NIST definition of "cloud computing" mentions three scenarios that raise different ethical issues: Software as a Service, Platform as a Service, and Infrastructure as a Service. However, that definition does not match the common use of "cloud computing," since it does not include storing data in online services. Software as a Service as defined by NIST overlaps considerably with Service as a Software Substitute, which mistreats the user, but the two concepts are not equivalent.

These different computing practices don't even belong in the same discussion. The best way to avoid the confusion the term "cloud computing" spreads is not to use the term "cloud" in connection with computing. Talk about the scenario you mean, and call it by a specific term.

Curiously, Larry Ellison, a proprietary software developer, also noted the vacuity of the term "cloud computing." He decided to use the term anyway because, as a proprietary software developer, he isn't motivated by the same ideals as we are.

**"Commercial"**

Please don't use "commercial" as a synonym for "nonfree." That confuses two entirely different issues.

A program is commercial if it is developed as a business activity. A commercial program can be free or nonfree, depending on its manner of distribution. Likewise, a program developed by a school or an

individual can be free or nonfree, depending on its manner of distribution. The two questions—what sort of entity developed the program and what freedom its users have—are independent.

In the first decade of the free software movement, free software packages were almost always noncommercial; the components of the GNU/Linux operating system were developed by individuals or by nonprofit organizations such as the FSF and universities. Later, in the 1990s, free commercial software started to appear.

Free commercial software is a contribution to our community, so we should encourage it. But people who think that "commercial" means "nonfree" will tend to think that the "free commercial" combination is self-contradictory, and dismiss the possibility. Let's be careful not to use the word "commercial" in that way.

**"Compensation"**

To speak of "compensation for authors" in connection with copyright carries the assumptions that (1) copyright exists for the sake of authors and (2) whenever we read something, we take on a debt to the author which we must then repay. The first assumption is simply false, and the second is outrageous.

"Compensating the rights-holders" adds a further swindle: you're supposed to imagine that means paying the authors, and occasionally it does, but most of the time it means a subsidy for the same publishing companies that are pushing unjust laws on us.

**"Consume"**

"Consume" refers to what we do with food: we ingest it, after which the food as such no longer exists. By analogy, we employ the same word for other products whose use *uses them up*. Applying it to durable goods, such as clothing or appliances, is a stretch. Applying it to published works (programs, recordings on a disk or in a file, books on paper or in a file), whose nature is to last indefinitely and which can be run, played or read any number of times, is stretching the word so far that it snaps. Playing a recording, or running a program, does not consume it.

Those who use "consume" in this context will say they don't mean it literally. What, then, does it mean? It means to regard copies of software and other works from a narrow economistic point of view. "Consume" is associated with the economics of material commodities, such as the fuel or electricity that a car uses up. Gasoline is a commodity, and so is electricity. Commodities are *fungible*: there is nothing special about a drop of gasoline that your car burns today versus another drop that it burned last week.

What does it mean to think of works of authorship as a commodity, with the assumption that there is nothing special about any one story, article, program, or song? That is the twisted viewpoint of the owner or the accountant of a publishing company, someone who doesn't appreciate the published works as such. It is no surprise that proprietary software developers would like you to think of the use of software as a commodity. Their twisted viewpoint comes through clearly in this article, which also refers to publications as "content."

The narrow thinking associated with the idea that we "consume content" paves the way for laws such as the DMCA that forbid users to break the Digital Restrictions Management (DRM) facilities in digital devices. If users think what they do with these devices is "consume," they may see such restrictions as natural.

It also encourages the acceptance of "streaming" services, which use DRM to perversely limit listening to music, or watching video, to squeeze those activities into the assumptions of the word "consume."

Why is this perverse usage spreading? Some may feel that the term sounds sophisticated, but rejecting it with cogent reasons can appear even more sophisticated. Some want to generalize about all kinds of media, but the usual English verbs ("read," "listen to," "watch") don't do this. Others may be acting from business interests (their own, or their employers'). Their use of the term in prestigious forums gives the impression that it's the "correct" term.

To speak of "consuming" music, fiction, or any other artistic works is to treat them as commodities rather than as art. Do we want to think of published works that way? Do we want to encourage the public to do so?

Those who answer no, please join me in shunning the term "consume" for this.

What to use instead? You can use specific verbs such as "read," "listen to," "watch" or "look at," since they help to restrain the tendency to overgeneralize.

If you insist on generalizing, you can use the expression "attend to," which requires less of a stretch than "consume." For a work meant for practical use, "use" is best.

See also the following entry.

**"Consumer"**

The term "consumer," when used to refer to the users of computing, is loaded with assumptions we should reject. Some come from the idea that using the program "consumes" the program (see the

previous entry), which leads people to impose on copiable digital works the economic conclusions that were drawn about uncopiable material products.

In addition, describing the users of software as "consumers" refers to a framing in which people are limited to selecting between whatever "products" are available in the "market." There is no room in this framing for the idea that users can directly exercise control over what a program does.

To describe people who are not limited to passive use of works, we suggest terms such as "individuals" and "citizens," rather than "consumers."

This problem with the word "consumer" has been noted before.

**"Content"**

If you want to describe a feeling of comfort and satisfaction, by all means say you are "content," but using the word as a noun to describe works and communications through which people have expressed themselves adopts an attitude you might rather avoid: it treats them as a commodity whose purpose is to fill a box and make money. In effect, it disparages all the works by focusing on the box that is full. To avoid taking that attitude, you can call them "works," "publications," "messages," "communications," as well as various other words that are more specific.

Those who use the term "content" are often the publishers that push for increased copyright power in the name of the authors ("creators," as they say) of the works. The term "content" reveals their real attitude towards these works and their authors.

The same word, "content," has another usage which is different enough in meaning that it does not raise this issue. It appears in the expression, "technical content." The usage of that expression generally relates to *one specific document* or publication, and refers to "the information *in that one*." This usage doesn't embody any attitude towards publications and communications in general.

Likewise, the word "contents" does not raise this issue. It is a form of the word "content," but it has a different meaning. Talking about the "contents" of a file or the "table of contents" of a book does not imply an attitude towards files in general or books in general.

We first condemned this usage of "content" in 2002. Since then, Tom Chatfield recognized the same point in *The Guardian*:

Content itself is beside the point—as the very use of words like content suggests. The moment you start labelling every single piece of writing in the world "content," you have conceded its interchangeability: its primary purpose as mere grist to the metrical mill.

In other words, “content” reduces publications and writings to a sort of pap, fit to be metered and piped through the “tubes” of the internet.

Later, Peter Bradshaw noticed it too.

This is what happens when studios treat movies as pure, undifferentiated corporate “content,” a Gazprom pipeline of superhero mush which can be turned off when the accountants say that it makes sense to do so.

Martin Scorsese condemned the attitude of “content” in regard to films.

The attitude implied by “content” is illustrated pointedly in this critical description of the development path of platforms run by people who base their thinking on that concept.

The article uses this word over and over, along with “consume” and “creators.” Perhaps that is meant to illustrate the way those people like to think.

See also Courtney Love's open letter to Steve Case and search for “content provider” in that page. Alas, Ms. Love is unaware that the term “intellectual property” is also biased and confusing.

However, as long as other people use the term “content provider,” political dissidents can well call themselves “malcontent providers.”

The term “content management” takes the prize for vacuity. “Content” means “some sort of information,” and “management” in this context means “doing something with it.” So a “content management system” is a system for doing something to some sort of information. Nearly all programs fit that description.

In most cases, that term really refers to a system for updating pages on a web site. For that, we recommend the term “web site revision system” or “website revision system” (WRS).

**“Copyright Owner”**

Copyright is an artificial privilege, handed out by the state to achieve a public interest and lasting a period of time—not a natural right like owning a house or a shirt. Lawyers used to recognize this by referring to the recipient of that privilege as a “copyright holder.”

A few decades ago, copyright holders began trying to reduce awareness of this point. In addition to citing frequently the bogus concept of “intellectual property,” they also started calling themselves “copyright owners.” Please join us in resisting by using the traditional term “copyright holders” instead.

**“Creative Commons licensed”**

The most important licensing characteristic of a work is whether it is free. Creative Commons publishes seven licenses; three are free (CC BY, CC BY-SA and CC0) and the rest are nonfree. Thus, to describe a work as “Creative Commons licensed” fails to say whether it is free, and suggests that the question is not important. The statement may be accurate, but the omission is harmful.

To encourage people to pay attention to the most important distinction, always specify *which* Creative Commons license is used, as in “licensed under CC BY-SA.” If you don't know which license a certain work uses, find out and then make your statement.

**“Creator”**

The term “creator” as applied to authors implicitly compares them to a deity (“the creator”). The term is used by publishers to elevate authors' moral standing above that of ordinary people in order to justify giving them increased copyright power, which the publishers can then exercise in their name. We recommend saying “author” instead. However, in many cases “copyright holder” is what you really mean. These two terms are not equivalent: often the copyright holder is not the author.

**“Digital Goods”**

The term “digital goods,” as applied to copies of works of authorship, identifies them with physical goods—which cannot be copied, and which therefore have to be manufactured in quantity and sold. This metaphor encourages people to judge issues about software or other digital works based on their views and intuitions about physical goods. It also frames issues in terms of economics, whose shallow and limited values don't include freedom and community.

**“Digital Locks”**

“Digital locks” is used to refer to Digital Restrictions Management by some who criticize it. The problem with this term is that it fails to do justice to the badness of DRM. The people who adopted that term did not think it through.

Locks are not necessarily oppressive or bad. You probably own several locks, and their keys or codes as well; you may find them useful or troublesome, but they don't oppress you, because you can open and close them. Likewise, we find encryption invaluable for protecting our digital files. That too is a kind of digital lock that you have control over.

DRM is like a lock placed on you by someone else, who refuses to give you the key—in other words, like *handcuffs*. Therefore, the proper metaphor for DRM is “digital handcuffs,” not “digital locks.”

A number of opposition campaigns have chosen the unwise term “digital locks”; to get things back on the right track, we must firmly insist on correcting this mistake. The FSF can support a campaign that opposes “digital locks” if we agree on the substance; however, when we state our support, we conspicuously replace the term with “digital handcuffs” and say why.

**“Digital Rights Management”**

“Digital Rights Management” (abbreviated “DRM”) refers to technical mechanisms designed to impose restrictions on computer users. The use of the word “rights” in this term is propaganda, designed to lead you unawares into seeing the issue from the viewpoint of the few that impose the restrictions, and ignoring that of the general public on whom these restrictions are imposed.

Good alternatives include “Digital Restrictions Management,” and “digital handcuffs.”

Please sign up to support our campaign to abolish DRM.

**“Ecosystem”**

It is inadvisable to describe the free software community, or any human community, as an “ecosystem,” because that word implies the absence of ethical judgment.

The term “ecosystem” implicitly suggests an attitude of nonjudgmental observation: don't ask how what *should* happen, just study and understand what *does* happen. In an ecosystem, some organisms consume other organisms. In ecology, we do not ask whether it is right for an owl to eat a mouse or for a mouse to eat a seed, we only observe that they do so. Species' populations grow or shrink according to the conditions; this is neither right nor wrong, merely an ecological phenomenon, even if it goes so far as the extinction of a species.

By contrast, beings that adopt an ethical stance towards their surroundings can decide to preserve things that, without their intervention, might vanish—such as civil society, democracy, human rights, peace, public health, a stable climate, clean air and water, endangered species, traditional arts...and computer users' freedom.

**“FLOSS”**

The term “FLOSS,” meaning “Free/Libre and Open Source Software,” was coined as a way to be neutral between free software and open source. If neutrality is your goal, “FLOSS” is the best way to be neutral. But if you want to show you stand for freedom, don't use a neutral term.

**“For free”**

If you want to say that a program is free software, please don't say that it is available “for free.” That term specifically means “for zero price.” Free software is a matter of freedom, not price.

Free software copies are often available for free—for example, by downloading via FTP. But free software copies are also available for a price on CD-ROMs; meanwhile, proprietary software copies are occasionally available for free in promotions, and some proprietary packages are normally available at no charge to certain users.

To avoid confusion, you can say that the program is available “as free software.”

**“FOSS”**

The term “FOSS,” meaning “Free and Open Source Software,” was coined as a way to be neutral between free software and open source, but it doesn't really do that. If neutrality is your goal, “FLOSS” is better. But if you want to show you stand for freedom, don't use a neutral term.

Instead of **FOSS**, we say, **free software** or **free (libre) software**.

**“Freely available”**

Don't use “freely available software” as a synonym for “free software.” The terms are not equivalent. Software is “freely available” if anyone can easily get a copy. “Free software” is defined in terms of the freedom of users that have a copy of it. These are answers to different questions.

**“Freemium”**

The confusing term “freemium” is used in marketing to describe *nonfree* software whose standard version is gratis, with paid *nonfree* add-ons available.

Using this term works against the free software movement, because it leads people to think of “free” as meaning “zero price.”

**“Free-to-play”**

The confusing term “free-to-play” (acronym “F2P”) is used in marketing to describe *nonfree* games which don't require a payment before a user starts to play. In many of these games, doing well in the game requires paying later, so the term “gratis-to-start” is a more accurate description.

Using this term works against the free software movement, because it leads people to think of "free" as meaning "zero price."

**"Freeware"**

Please don't use the term "freeware" as a synonym for "free software." The term "freeware" was used often in the 1980s for programs released only as executables, with source code not available. Today it has no particular agreed-on definition.

When using languages other than English, please avoid borrowing English terms such as "free software" or "freeware." It is better to translate the term "free software" into your language.

By using a word in your own language, you show that you are really referring to freedom and not just parroting some mysterious foreign marketing concept. The reference to freedom may at first seem strange or disturbing to your compatriots, but once they see that it means exactly what it says, they will really understand what the issue is.

**"Give away software"**

It's misleading to use the term "give away" to mean "distribute a program as free software." This locution has the same problem as "for free": it implies the issue is price, not freedom. One way to avoid the confusion is to say "release as free software."

**"Google"**

Please avoid using the term "google" as a verb, meaning to search for something on the internet. "Google" is just the name of one particular search engine among others. We suggest to use the term "search the web" or (in some contexts) just "search." Try to use a search engine that respects your privacy; for instance, DuckDuckGo claims not to track its users. (There is no way for outsiders to verify claims of that kind.)

**"Hacker"**

A hacker is someone who enjoys playful cleverness—not necessarily with computers. The programmers in the old MIT free software community of the 60s and 70s referred to themselves as hackers. Around 1980, journalists who discovered the hacker community mistakenly took the term to mean "security breaker."

Please don't spread this mistake. People who break security are "crackers."

**“Intellectual property”**

Publishers and lawyers like to describe copyright as “intellectual property”—a term also applied to patents, trademarks, and other more obscure areas of law. These laws have so little in common, and differ so much, that it is ill-advised to generalize about them. It is best to talk specifically about “copyright,” or about “patents,” or about “trademarks.”

The term “intellectual property” carries a hidden assumption—that the way to think about all these disparate issues is based on an analogy with physical objects, and our conception of them as physical property.

When it comes to copying, this analogy disregards the crucial difference between material objects and information: information can be copied and shared almost effortlessly, while material objects can't be.

To avoid spreading unnecessary bias and confusion, it is best to adopt a firm policy not to speak or even think in terms of “intellectual property”.

The hypocrisy of calling these powers “rights” is starting to make the World “Intellectual Property” Organization embarrassed.

**“Internet of Things”**

When companies decided to make computerized appliances that would connect over the internet to the manufacturer's server, and therefore could easily snoop on their users, they realized that this would not sound very nice. So they came up with a cute, appealing name: the “Internet of Things.”

Experience shows that these products often do spy on their users. They are also tailor-made for giving people biased advice. In addition, the manufacturer can sabotage the product by turning off the server it depends on.

We call them the “Internet of Stings.”

**“LAMP system”**

“LAMP” stands for “Linux, Apache, MySQL and PHP”—a common combination of software to use on a web server, except that “Linux” in this context really refers to the GNU/Linux system. So instead of “LAMP” it should be “GLAMP”: “GNU, Linux, Apache, MySQL and PHP.”

**“Linux system”**

Linux is the name of the kernel that Linus Torvalds developed starting in 1991. The operating system in which Linux is used is basically GNU with Linux added. To call the whole system "Linux" is both unfair and confusing. Please call the complete system GNU/Linux, both to give the GNU Project credit and to distinguish the whole system from the kernel alone.

**"Market"**

It is misleading to describe the users of free software, or the software users in general, as a "market."

This is not to say there is no room for markets in the free software community. If you have a free software support business, then you have clients, and you trade with them in a market. As long as you respect their freedom, we wish you success in your market.

But the free software movement is a social movement, not a business, and the success it aims for is not a market success. We are trying to serve the public by giving it freedom—not competing to draw business away from a rival. To equate this campaign for freedom to a business's efforts for mere success is to deny the importance of freedom and legitimize proprietary software.

**"Modern"**

The term "modern" makes sense from a descriptive perspective—for instance, solely to distinguish newer periods and ways from older ones.

It becomes a problem when it carries the presumption that older ways are "old-fashioned"; that is, presumed to be worse. In technological fields where businesses make the choices and impose them on users, the reverse is often true.

**"Monetize"**

The proper definition of "monetize" is "to use something as currency." For instance, human societies have monetized gold, silver, copper, printed paper, special kinds of seashells, and large rocks. However, we now see a tendency to use the word in another way, meaning "to use something as a basis for profit."

That usage casts the profit as primary, and the thing used to get the profit as secondary. That attitude applied to a software project is objectionable because it would lead the developers to make the program proprietary, if they conclude that making it free/libre isn't sufficiently profitable.

A productive and ethical business can make money, but if it subordinates all else to profit, it is not likely to remain ethical.

**“MP3 Player”**

In the late 1990s it became feasible to make portable, solid-state digital audio players. Most players supported the patented MP3 codec, and that is still the case. Some players also supported the patent-free audio codecs Ogg Vorbis and FLAC, and a few couldn't play MP3-encoded files at all because their developers needed to protect themselves from the patents on MP3 format.

Using the term “MP3 players” for audio players in general has the effect of promoting the MP3 format and discouraging the other formats (some of which are technically superior as well). Even though the MP3 patents have expired, it is still undesirable to do that.

We suggest the term “digital audio player,” or simply “audio player” when that's clear enough, instead of “MP3 player.”

**“Open”**

Please avoid using the term “open” or “open source” as a substitute for “free software.” Those terms refer to a different set of views based on different values. The free software movement campaigns for your freedom in your computing, as a matter of justice. The open source non-movement does not campaign for anything in this way.

When referring to the open source views, it's correct to use that name, but please do not use that term when talking about us, our software, or our views—that leads people to suppose our views are similar to theirs.

Instead of **open source**, we say, **free software** or **free (libre) software**.

**“Opt out”**

When applied to any form of computational mistreatment, “opt out” implies the choice is a minor matter of convenience. We recommend “reject,” “shun” or “escape from.”

**“PC”**

It's OK to use the abbreviation “PC” to refer to a certain kind of computer hardware, but please don't use it with the implication that the computer is running Microsoft Windows. If you install GNU/Linux on the same computer, it is still a PC.

The term “WC” has been suggested for a computer running Windows.

**“Photoshop”**

Please avoid using the term “photoshop” as a verb, meaning any kind of photo manipulation or image editing in general. Photoshop is just the name of one particular image editing program, which should be avoided since it is proprietary. There are plenty of free programs for editing images, such as the GIMP.

**“Players” (said of businesses)**

To describe businesses as “players” presumes that they are motivated purely and simply by “winning” what they treat as a poker-like game—in effect, subordinating all else to profit. Often businesses (and their executives) do act that way, but not always, and we often pressure them to respect other values as well.

The moral cynicism of “players” resonates with a general condemnation of business, which to some extent business in general deserves; at the same time, it tends to dissuade the attempt to judge any business's acts or practices in moral terms. Even to raise the question of whether a certain business treats people unjustly is dissuaded by the “players” metaphor's murmuring, in the background, “Why bother asking?” let's avoid that metaphor.

**“Piracy”**

Publishers often refer to copying they don't approve of as “piracy.” In this way, they imply that it is ethically equivalent to attacking ships on the high seas, kidnapping and murdering the people on them. Based on such propaganda, they have procured laws in most of the world to forbid copying in most (or sometimes all) circumstances. (They are still pressuring to make these prohibitions more complete.)

If you don't believe that copying not approved by the publisher is just like kidnapping and murder, you might prefer not to use the word “piracy” to describe it. Neutral terms such as “unauthorized copying” (or “prohibited copying” for the situation where it is illegal) are available for use instead. Some of us might even prefer to use a positive term such as “sharing information with your neighbor.”

A US judge, presiding over a trial for copyright infringement, recognized that “piracy” and “theft” are smear words.

**“PowerPoint”**

Please avoid using the term “PowerPoint” to mean any kind of slide presentation. “PowerPoint” is just the name of one particular proprietary program to make presentations. For your freedom's sake, you

should use only free software to make your presentations—which means, *not PowerPoint*. Recommended options include LaTeX's beamer class and LibreOffice Impress.

**"Product"**

If you're talking about a product, by all means call it that. However, when referring to a service, please do not call it a "product." If a service provider calls the service a "product," please firmly insist on calling it a "service." If a service provider calls a package deal a "product," please firmly insist on calling it a "deal."

**"Protection"**

Publishers' lawyers love to use the term "protection" to describe copyright. This word carries the implication of preventing destruction or suffering; therefore, it encourages people to identify with the owner and publisher who benefit from copyright, rather than with the users who are restricted by it.

It is easy to avoid "protection" and use neutral terms instead. For example, instead of saying, "Copyright protection lasts a very long time," you can say, "Copyright lasts a very long time."

Likewise, instead of saying, "protected by copyright," you can say, "covered by copyright" or just "copyrighted."

If you want to criticize copyright rather than be neutral, you can use the term "copyright restrictions." Thus, you can say, "Copyright restrictions last a very long time."

The term "protection" is also used to describe malicious features. For instance, "copy protection" is a feature that interferes with copying. From the user's point of view, this is obstruction. So we could call that malicious feature "copy obstruction." More often it is called Digital Restrictions Management (DRM)—see the Defective by Design campaign.

**"RAND (Reasonable and Non-Discriminatory)"**

Standards bodies that promulgate patent-restricted standards that prohibit free software typically have a policy of obtaining patent licenses that require a fixed fee per copy of a conforming program. They often refer to such licenses by the term "RAND," which stands for "reasonable and non-discriminatory."

That term whitewashes a class of patent licenses that are normally neither reasonable nor nondiscriminatory. It is true that these licenses do not discriminate against any specific person, but they do discriminate against the free software community, and that makes them unreasonable. Thus, half of the term "RAND" is deceptive and the other half is prejudiced.

Standards bodies should recognize that these licenses are discriminatory, and drop the use of the term "reasonable and non-discriminatory" or "RAND" to describe them. Until they do so, writers who do not wish to join in the whitewashing would do well to reject that term. To accept and use it merely because patent-wielding companies have made it widespread is to let those companies dictate the views you express.

We suggest the term "uniform fee only," or "UFO" for short, as a replacement. It is accurate because the only condition in these licenses is a uniform royalty fee.

**"SaaS" or "Software as a Service"**

We used to say that SaaS (short for "Software as a Service") is an injustice, but then we found that there was a lot of variation in people's understanding of which activities count as SaaS. So we switched to a new term, "Service as a Software Substitute" or "SaaSS." This term has two advantages: it wasn't used before, so our definition is the only one, and it explains what the injustice consists of.

See Who Does That Server Really Serve? for discussion of this issue.

In Spanish we continue to use the term "software como servicio" because the joke of "software como ser vicio" ("software, as being pernicious") is too good to give up.

**"Sell software"**

The term "sell software" is ambiguous. Strictly speaking, exchanging a copy of a free program for a sum of money is selling the program, and there is nothing wrong with doing that. However, people usually associate the term "selling software" with proprietary restrictions on the subsequent use of the software. You can be clear, and prevent confusion, by saying either "distributing copies of a program for a fee" or "imposing proprietary restrictions on the use of a program."

See Selling Free Software for further discussion of this issue.

**"Sharing (personal data)"**

When companies manipulate or lure people into revealing personal data and thus ceding their privacy, please don't refer to this as "sharing." We use the term "sharing" to refer to noncommercial

cooperation, including noncommercial redistribution of exact copies of published works, and we say this is *good*. Please don't apply that word to a practice which is harmful and dangerous.

When one company redistributes collected personal data to another company, that is even less deserving of the term "sharing."

**"Sharing economy"**

The term "sharing economy" is not a good way to refer to services such as Uber and Airbnb that arrange business transactions between people. We use the term "sharing" to refer to noncommercial cooperation, including noncommercial redistribution of exact copies of published works. Stretching the word "sharing" to include these transactions undermines its meaning, so we don't use it in this context.

A more suitable term for businesses like Uber is the "piecework service economy" or "gig economy."

**"Skype"**

Please avoid using the term "skype" as a verb, meaning any kind of video communication or telephony over the Internet in general. "Skype" is just the name of one particular proprietary program, one that spies on its users. If you want to make video and voice calls over the Internet in a way that respects both your freedom and your privacy, try one of the numerous free Skype replacements.

**"Smart speaker"**

This term is totally absurd. It refers to products that listen and understand voice commands; they also have a speaker for speaking output from those commands. Their primary function is to listen to commands. Let's call them "voice command listeners."

**"Software Industry"**

The term "software industry" encourages people to imagine that software is always developed by a sort of factory and then delivered to "consumers." The free software community shows this is not the case. Software businesses exist, and various businesses develop free and/or nonfree software, but those that develop free software are not run like factories.

The term "industry" is being used as propaganda by advocates of software patents. They call software development "industry" and then try to argue that this means it should be subject to patent monopolies. The European Parliament, rejecting software patents in 2003, voted to define "industry" as "automated production of material goods."

**“Source model”**

Wikipedia uses the term “source model” in a confused and ambiguous way. Ostensibly it refers to how a program's source is distributed, but the text confuses this with the development methodology. It distinguishes “open source” and ”shared source” as answers, but they overlap—Microsoft uses the latter as a marketing term to cover a range of practices, some of which are “open source.” Thus, this term really conveys no coherent information, but it provides an opportunity to say “open source” in pages describing free software programs.

**“Theft”**

The supporters of a too-strict, repressive form of copyright often use words like “stolen” and “theft” to refer to copyright infringement. This is spin, but they would like you to take it for objective truth.

Under the US legal system, copyright infringement is not theft. Laws about theft are not applicable to copyright infringement. The supporters of repressive copyright are making an appeal to authority—and misrepresenting what authority says.

To refute them, you can point to this real case which shows what can properly be described as “copyright theft.”

Unauthorized copying is forbidden by copyright law in many circumstances (not all!), but being forbidden doesn't make it wrong. In general, laws don't define right and wrong. Laws, at their best, attempt to implement justice. If the laws (the implementation) don't fit our ideas of right and wrong (the spec), the laws are what should change.

A US judge, presiding over a trial for copyright infringement, recognized that “piracy” and “theft” are smear-words.

**“Trusted Computing”**

“Trusted computing” is the proponents' name for a scheme to redesign computers so that application developers can trust your computer to obey them instead of you. From their point of view, it is “trusted”; from your point of view, it is “treacherous.”

**“Vendor”**

Please don't use the term “vendor” to refer generally to anyone that develops or packages software. Many programs are developed in order to sell copies, and their developers are therefore their vendors; this even includes some free software packages. However, many programs are developed by volunteers or organizations which do not intend to sell copies. These developers are not vendors. Likewise, only

some of the packagers of GNU/Linux distributions are vendors. We recommend the general term "supplier" instead.

- END OF CHAPTER

# Thank you for reading the book!

[1] https://www.gnu.org/philosophy/selling.html

[2] https://www.gnu.org/philosophy/free-software-even-more-important.html

[3] https://www.gnu.org/philosophy/open-source-misses-the-point.html

[4] https://www.gnu.org/philosophy/free-sw.html#History

[5] https://www.gnu.org/philosophy/free-sw.html#f1

[6] https://www.gnu.org/philosophy/free-sw.html#exportcontrol

[7] https://www.gnu.org/licenses/copyleft.html

[8] https://www.gnu.org/philosophy/pragmatic.html

[9] https://www.gnu.org/philosophy/categories.html#Non-CopyleftedFreeSoftware

[10] https://www.gnu.org/philosophy/categories.html

[11] https://www.gnu.org/licenses/license-list.html

[12] https://www.gnu.org/philosophy/words-to-avoid.html

[13] https://www.gnu.org/philosophy/fs-translations.html

[14] https://www.gnu.org/philosophy/free-doc.html

[15] https://wikipedia.org/

[16] https://freedomdefined.org/

[17] https://web.cvs.savannah.gnu.org/viewvc/www/philosophy/free-sw.html?root=www&view=log

[18] https://www.gnu.org/malware

[19] https://www.gnu.org/philosophy/loyal-computers.html

[20] https://observer.com/2016/06/how-technology-hijacks-peoples-minds-from-a-magician-and-googles-design-ethicist/

[21] https://www.gnu.org/philosophy/free-sw.html

[22] https://www.gnu.org/philosophy/why-call-it-the-swindle.html

[23] https://archive.ieet.org/articles/rinesi20150806.html

[24] https://www.gnu.org/gnu/thegnuproject.html

[25] https://www.gnu.org/gnu/gnu-linux-faq.html

[26] https://www.gnu.org/philosophy/who-does-that-server-really-serve.html

[27] https://arstechnica.com/information-technology/2013/06/nsa-gets-early-access-to-zero-day-data-from-microsoft-others/

[28] https://www.gnu.org/philosophy/government-free-software.html

[29] https://www.gnu.org/education/education.html

[30] https://www.gnu.org/licenses/license-recommendations.html

[31] https://www.gnu.org/help/help.html

[32] https://www.gnu.org/philosophy/saying-no-even-once.html

[33] https://www.gnu.org/philosophy/free-sw.html

[34] https://www.gnu.org/philosophy/categories.html#ProprietarySoftware

[35] https://www.fsf.org/

[36] https://savannah.gnu.org/projects/tasklist

[37] https://www.gnu.org/doc/doc.html

[38] https://www.gnu.org/philosophy/words-to-avoid.html#SellSoftware

[39] https://www.gnu.org/licenses/gpl.html

[40] https://www.gnu.org/licenses/gpl.html#section6

[41] https://badvista.fsf.org/

[42] https://www.gnu.org/proprietary/malware-microsoft.html

[43] https://www.fsf.org/windows

[44] https://www.gnu.org/proprietary/malware-apple.html

[45] https://www.techworm.net/2013/06/nsa-built-back-door-in-microsofts-all.html

[46] https://www.gnu.org/gnu/linux-and-gnu.html

[47] https://www.gnu.org/gnu/gnu.html

[48] https://www.gnu.org/distros/distros.html

[49] https://www.gnu.org/licenses/license-list.html

[50] https://www.gnu.org/philosophy/free-open-overlap.html

[51] https://libreboot.org/faq.html#intelme

[52] https://www.gnu.org/proprietary/proprietary-insecurity.html#uefi-rootkit

[53] https://www.gnu.org/licenses/quick-guide-gplv3.html

[54] https://www.gnu.org/distros/free-system-distribution-guidelines.html

[55] https://www.gnu.org/software/repo-criteria.html

[56] https://sv.gnu.org/

[57] https://www.gnu.org/philosophy/compromise.html

[58] https://www.gnu.org/philosophy/free-doc.html

[59] https://gcc.gnu.org/ml/gcc/2014-01/msg00247.html

[60] https://directory.fsf.org/wiki/IceCat

[61] https://www.gnu.org/distros

[62] https://www.fsf.org/resources/hw/endorsement/criteria

[63] https://www.gnu.org/philosophy/javascript-trap.html

[64] https://www.gnu.org/software/librejs

[65] https://www.gnu.org/gnu/thegnuproject.html

[66] https://www.gnu.org/licenses/gpl-2.0.html

[67] https://www.gnu.org/licenses/license-list.html#apache2

[68] https://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/

[69] https://www.greenbot.com/new-google-play-edition-devices-lack-photo-gallery-app-use-google/

[70] https://arstechnica.com/gadgets/2014/06/android-wear-auto-and-tv-save-you-from-skins-and-oems-from-themselves/

[71] https://blog.grobox.de/2016/the-proprietarization-of-android-google-play-services-and-apps/

[72] https://www.beneaththewaves.net/Projects/Motorola_Is_Listening.html

[73] https://androidsecuritytest.com/features/logs-and-services/loggers/carrieriq/

[74] https://replicant.us/

[75] ftp://ftp.cs.wisc.edu/pub/paradyn/technical_papers/fuzz-revisited.ps

[76] https://shop.fsf.org/category/books/

[77] https://www.gnu.org/doc/doc.html

[78] https://www.gnu.org/licenses/fdl.html

[79] https://www.gnu.org/doc/other-free-books.html

[80] https://www.theverge.com/2019/6/18/18683455/nasa-space-angels-contracts-government-investment-spacex-air-force

[81] https://www.gnu.org/proprietary/proprietary-surveillance.html

[82] https://web.archive.org/web/20211106213411/http://www.clifford.at/icestorm/

[83] https://gothub.projectsegfau.lt/Wolfgang-Spraul/fpgatools/

[84] https://ryf.fsf.org/

[85] https://www.copyright.gov/title17/92chap13.html#1301

[86] https://www.gnu.org/philosophy/not-ipr.html

[87] https://en.wikipedia.org/wiki/STL_(file_format)

[88] https://web.archive.org/web/20211203021432/https://www.publicknowledge.org/assets/uploads/documents/3_Steps_for_Licensing_Your_3D_Printed_Stuff.pdf

[89] https://www.gnu.org/philosophy/words-to-avoid.html#Protection

[90] https://www.gnu.org/licenses/why-affero-gpl.html

[91] en.wikipedia.org/wiki/Gnutella

[92] https://www.gnu.org/philosophy/categories.html#GNUsoftware

[93] https://sourceforge.net/projects/gtk-gnutella/

[94] https://sourceforge.net/projects/mutella/

[95] https://sourceforge.net/projects/gnucleus/

[96] https://www.gnu.org/software/gnunet/

[97] https://web.archive.org/web/20180616130316/https://gnunet.org/compare

[98] https://www.gnu.org/philosophy/essays-and-articles.html#Laws

[99] https://www.gnu.org/philosophy/third-party-ideas.html

[100] https://www.gnu.org/philosophy/reevaluating-copyright.html

[101] https://www.gnu.org/philosophy/java-trap.html

[102] https://www.gnu.org/software/health/

[103] https://www.tryton.org/

[104] https://www.gnu.org/philosophy/surveillance-vs-democracy.html

[105] https://stallman.org/articles/on-hacking.html

[106] https://www.gnu.org/philosophy/saying-no-even-once.html

[107] https://linuxinsider.com/story/Open-Source-Is-Woven-Into-the-Latest-Hottest-Trends-78937.html

[108] https://opensource.org/osd

[109] https://web.archive.org/web/20001011193422/http://da.state.ks.us/ITEC/TechArchPt6ver80.pdf

[110] https://www.nytimes.com/external/gigaom/2009/02/07/07gigaom-the-brave-new-world-of-open-source-game-design-37415.html

[111] https://www.theguardian.com/sustainable-business/2015/aug/27/texas-teenager-water-purifier-toxic-e-waste-pollution

[112] https://www.nytimes.com/2013/03/17/opinion/sunday/morozov-open-and-closed.html

[113] https://www.gnu.org/philosophy/floss-and-foss.html

[114] https://web.archive.org/web/20010618050431/itworld.com/AppDev/350/LWD010523vcontrol4/pfindex.html

[115] https://ocw.mit.edu/courses/sloan-school-of-management/15-352-managing-innovation-emerging-trends-spring-2005/readings/lakhaniwolf.pdf

[116] https://www.defectivebydesign.org/what_is_drm_digital_restrictions_management

[117] https://www.gnu.org/proprietary/proprietary-jails.html

[118] https://www.gnu.org/philosophy/technological-neutrality.html

[119] https://www.fsf.org/blogs/community/better-than-zoom-try-these-free-software-tools-for-staying-in-touch

[120] https://www.gnu.org/philosophy/x.html

[121] https://github.com/w3c/fingerprinting-guidance/issues/8

[122] https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/

[123] https://www.gnu.org/licenses/javascript-labels.html

[124] https://books.google.com/ngrams/graph?content=intellectual+property&year_start=1800&year_end=2008&corpus=15&smoothing=1&share=&direct_url=t1;,intellectual property;,c0

[125] https://www.gnu.org/graphics/seductivemirage.png

[126] https://www.theguardian.com/us-news/2017/mar/11/nebraska-farmers-right-to-repair-bill-stalls-apple

[127] https://fsfe.org/activities/wipo/wiwo.en.html

[128] https://web.archive.org/web/20140217075603/http://bluraysucks.com/

[129] https://www.gnu.org/philosophy/no-word-attachments.html

[130] https://www.cl.cam.ac.uk/~rja14/tcpa-faq.html

[131] https://developer.android.com/privacy-and-security/safetynet/attestation

[132] https://developer.android.com/privacy-and-security/safetynet/deprecation-timeline

[133] https://grapheneos.org/articles/attestation-compatibility-guide

[134] https://www.defectivebydesign.org/

[135] https://www.gnu.org/philosophy/proprietary-surveillance.html

[136] https://slate.com/technology/2013/09/privacy-facebook-kids-dont-post-photos-of-your-kids-on-social-media.html

[137] https://www.theguardian.com/business/2023/nov/05/cloud-service-provider-consumer-prices-netflix-microsoft

[138] https://opendefinition.org/ossd/

[139] https://www.mindprison.cc/p/the-question-that-no-llm-can-answer

[140] https://link.springer.com/article/10.1007/s10676-024-09775-5

[141] https://www.theguardian.com/environment/2024/apr/03/early-warning-system-track-asian-hornets-university-of-exeter

[142] https://www.theguardian.com/society/article/2024/aug/19/ai-may-help-experts-identify-toddlers-at-risk-of-autism-researchers-say

[143] https://www.theguardian.com/technology/article/2024/jul/25/google-deepmind-takes-step-closer-to-cracking-top-level-maths