Richard Stallman is the prophet of the free software movement. He understood the dangers of softwar patents years ago. Now that this has become a crucial issue in the world, buy this book and read wha he said.

—**Tim Berners-Lee,** inventor of the World Wide Web

Richard Stallman is the philosopher king of software. He single-handedly ignited what has become a world-wide movement to create software that is Free, with a capital F. He has toiled for years at a project that many once considered a fool's errand, and now that is widely seen as "inevitable."

—**Simon L. Garfinkel,** computer science author and columnist

By his hugely successful efforts to establish the idea of "Free Software," Stallman has made a massiv contribution to the human condition. His contribution combines elements that have technical, social, political, and economic consequences.

—**Gerald Jay Sussman,** Matsushita Professor of Electrical Engineering, MIT

RMS is the leading philosopher of software. You may dislike some of his attitudes, but you cannot avoid his ideas. This slim volume will make those ideas readily accessible to those who are confused by the buzzwords of rampant commercialism. This book needs to be widely circulated and widely rea

—**Peter Salus,** computer science writer, book reviewer, and UNIX historian

Richard is the leading force of the free software movement. This book is very important to spread the key concepts of free software world-wide, so everyone can understand it. Free software gives people freedom to use their creativity.

—**Masayuki Ida,** professor, Graduate School of International Management, Aoyama Gakuin University

Free Software, Free Society

Selected Essays of Richard M. Stallman
Second Edition

Richard M. Stallman

This is the second edition of *Free Software, Free Society: Selected Essays of Richard M. Stallman.*

Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1335

ISBN 978-0-9831592-0-9

Cover design by Rob Myers.
Cover photograph by Peter Hinely.

Contents

Foreword

This foreword was originally published, in 2002, as the introduction to the first edition. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman,* 2nd ed. (Boston: GNU Press, 2010).

Every generation has its philosopher—a writer or an artist who captures the imagination of a time Sometimes these philosophers are recognized as such; often it takes generations before the connectio is made real. But recognized or not, a time gets marked by the people who speak its ideals, whether i the whisper of a poem, or the blast of a political movement.

Our generation has a philosopher. He is not an artist, or a professional writer. He is a programmer Richard Stallman began his work in the labs of MIT, as a programmer and architect building operatin system software. He has built his career on a stage of public life, as a programmer and an architect founding a movement for freedom in a world increasingly defined by "code."

"Code" is the technology that makes computers run. Whether inscribed in software or burned in hardware, it is the collection of instructions, first written in words, that directs the functionality of machines. These machines—computers—increasingly define and control our life. They determine how phones connect, and what runs on TV. They decide whether video can be streamed across a broadband link to a computer. They control what a computer reports back to its manufacturer. These machines run us. Code runs these machines.

What control should we have over this code? What understanding? What freedom should there be to match the control it enables? What power?

These questions have been the challenge of Stallman's life. Through his works and his words, he has pushed us to see the importance of keeping code "free." Not free in the sense that code writers don't get paid, but free in the sense that the control coders build be transparent to all, and that anyone have the right to take that control, and modify it as he or she sees fit. This is "free software"; "free software" is one answer to a world built in code.

"Free." Stallman laments the ambiguity in his own term. There's nothing to lament. Puzzles force people to think, and this term "free" does this puzzling work quite well. To modern American ears, "free software" sounds utopian, impossible. Nothing, not even lunch, is free. How could the most important words running the most critical machines running the world be "free." How could a sane society aspire to such an ideal?

Yet the odd clink of the word "free" is a function of us, not of the term. "Free" has different senses, only one of which refers to "price." A much more fundamental sense of "free" is the "free," Stallman says, in the term "free speech," or perhaps better in the term "free labor." Not free as in costless, but free as in limited in its control by others. Free software is control that is transparent, and open to change, just as free laws, or the laws of a "free society," are free when they make their contro knowable, and open to change. The aim of Stallman's "free software movement" is to make as much

code as it can transparent, and subject to change, by rendering it “free.”

The mechanism of this rendering is an extraordinarily clever device called “copyleft” implemented through a license called GPL. Using the power of copyright law, “free software” not on assures that it remains open, and subject to change, but that other software that takes and uses “free software” (and that technically counts as a “derivative”) must also itself be free. If you use and adapt free software program, and then release that adapted version to the public, the released version must be as free as the version it was adapted from. It must, or the law of copyright will be violated.

“Free software,” like free societies, has its enemies. Microsoft has waged a war against the GPL, warning whoever will listen that the GPL is a “dangerous” license. The dangers it names, however, ar largely illusory. Others object to the “coercion” in GPL’s insistence that modified versions are also free. But a condition is not coercion. If it is not coercion for Microsoft to refuse to permit users to distribute modified versions of its product Office without paying it (presumably) millions, then it is not coercion when the GPL insists that modified versions of free software be free too.

And then there are those who call Stallman’s message too extreme. But extreme it is not. Indeed, in an obvious sense, Stallman’s work is a simple translation of the freedoms that our tradition crafted in the world before code. “Free software” would assure that the world governed by code is as “free” a our tradition that built the world before code.

For example: A “free society” is regulated by law. But there are limits that any free society place on this regulation through law: No society that kept its laws secret could ever be called free. No government that hid its regulations from the regulated could ever stand in our tradition. Law controls But it does so justly only when visibly. And law is visible only when its terms are knowable and controllable by those it regulates, or by the agents of those it regulates (lawyers, legislatures).

This condition on law extends beyond the work of a legislature. Think about the practice of law in American courts. Lawyers are hired by their clients to advance their clients’ interests. Sometimes tha interest is advanced through litigation. In the course of this litigation, lawyers write briefs. These briefs in turn affect opinions written by judges. These opinions decide who wins a particular case, or whether a certain law can stand consistently with a constitution.

All the material in this process is free in the sense that Stallman means. Legal briefs are open and free for others to use. The arguments are transparent (which is different from saying they are good) and the reasoning can be taken without the permission of the original lawyers. The opinions they produce can be quoted in later briefs. They can be copied and integrated into another brief or opinion The “source code” for American law is by design, and by principle, open and free for anyone to take. And take lawyers do—for it is a measure of a great brief that it achieves its creativity through the reuse of what happened before. The source is free; creativity and an economy is built upon it.

This economy of free code (and here I mean free legal code) doesn’t starve lawyers. Law firms have enough incentive to produce great briefs even though the stuff they build can be taken and copie by anyone else. The lawyer is a craftsman; his or her product is public. Yet the crafting is not charity Lawyers get paid; the public doesn’t demand such work without price. Instead this economy flourishes, with later work added to the earlier.

We could imagine a legal practice that was different—briefs and arguments that were kept secret

rulings that announced a result but not the reasoning. Laws that were kept by the police but published to no one else. Regulation that operated without explaining its rule.

We could imagine this society, but we could not imagine calling it “free.” Whether or not the incentives in such a society would be better or more efficiently allocated, such a society could not be known as free. The ideals of freedom, of life within a free society, demand more than efficient application. Instead, openness and transparency are the constraints within which a legal system gets built, not options to be added if convenient to the leaders. Life governed by software code should be no less.

Code writing is not litigation. It is better, richer, more productive. But the law is an obvious instance of how creativity and incentives do not depend upon perfect control over the products created. Like jazz, or novels, or architecture, the law gets built upon the work that went before. This adding and changing is what creativity always is. And a free society is one that assures that its most important resources remain free in just this sense.

This book collects the writing of Richard Stallman in a manner that will make its subtlety and power clear. The essays span a wide range, from copyright to the history of the free software movement. They include many arguments not well known, and among these, an especially insightful account of the changed circumstances that render copyright in the digital world suspect. They will serve as a resource for those who seek to understand the thought of this most powerful man—powerf in his ideas, his passion, and his integrity, even if powerless in every other way. They will inspire others who would take these ideas, and build upon them.

I don’t know Stallman well. I know him well enough to know he is a hard man to like. He is driven, often impatient. His anger can flare at friend as easily as foe. He is uncompromising and persistent; patient in both.

Yet when our world finally comes to understand the power and danger of code—when it finally sees that code, like laws, or like government, must be transparent to be free—then we will look back at this uncompromising and persistent programmer and recognize the vision he has fought to make real: the vision of a world where freedom and knowledge survives the compiler. And we will come to see that no man, through his deeds or words, has done as much to make possible the freedom that this next society could have.

We have not earned that freedom yet. We may well fail in securing it. But whether we succeed or fail, in these essays is a picture of what that freedom could be. And in the life that produced these words and works, there is inspiration for anyone who would, like Stallman, fight to create this freedom.

LAWRENCE LESSI

Lawrence Lessig is a Professor of Law at Harvard Law School, the director of the Edmond J. Safra Foundation Center for Ethic and the founder of Stanford Law School’s Center for Internet and Society. For much of his career, he focused his work on law and technology, especially as it affects copyright. He is the author of numerous books and has served as a board member of many organizations, including the Free Software Foundation.

Preface to the Second Edition

The second edition of *Free Software, Free Society* holds updated versions of most of the essays from the first edition, as well as many new essays published since the first edition.

The essays about software patents are now in one section and those about copyright in another, to set an example of not grouping together these two laws, whose workings and effects on software are totally different.

Another section presents the GNU licenses, with a new introduction written with Brett Smith giving their history and the motives for each of them. One of the essays explains why software projects should upgrade to version 3 of the GNU General Public License.

There is now a section on issues of terminology, since the way we describe an issue affects how people think about it.

The last two sections describe some of the traps free software developers and users face—new ways to lose your freedom, and how to avoid them.

We have also added an index, to complement the appendix on software.

We would like to thank Jeanne Rasata for managing the project, editing the book, formatting the text, and creating the index. Thanks also to Karl Berry for technical assistance with Texinfo, Brett Smith for all other technical help and for valuable feedback, and Rob Myers for formatting the cover

Part I
The GNU Project and Free Software

Chapter 1
The Free Software Definition

The free software definition was first published in 1996, on http://gnu.org. This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

We maintain this free software definition to show clearly what must be true about a particular software program for it to be considered free software. From time to time we revise this definition to clarify it. If you would like to review the changes we've made, please see the History section, following the definition, at http://gnu.org/philosophy/free-sw.html.

"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."

Free software is a matter of the users' freedom to run, copy, distribute, study, change and improv the software. More precisely, it means that the program's users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (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 users have all of these freedoms. Thus, you should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission to do so.

You should also have the freedom to make modifications and use them privately in your own wor 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.

The freedom to run the program means the freedom for any kind of person or organization to use 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 someone else, she is then free to run it for her purposes, but you are not entitled to impose your purposes on her.

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.

In order for freedoms 1 and 3 (the freedom to make changes and the freedom to publish improved versions) to be meaningful, you must 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 n real source code and does not count as source code.

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 (in its practitioners' perverse terminology) as "secure boot"—freedom 1 becomes a theoretical fiction rather than a practical freedom. This is not sufficient In other words, 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.

Freedom 3 includes the freedom to release your modified versions as free software. A free licens may also permit other ways of releasing them; in other words, it does not have to be a copyleft licens However, a license that requires modified versions to be nonfree does not qualify as a free license.

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 change its terms, without your doing anything wrong to give cause, the software is not free.

However, 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 centra freedoms. This rule does not conflict with the central freedoms; rather it protects them.

"Free software" does not mean "noncommercial." A free program must be available for commercial use, commercial development, and commercial distribution. Commercial development o free software is no longer unusual; such free commercial software is very important. You may have paid money to get copies of free software, 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.

Whether a change constitutes an improvement is a subjective matter. If your modifications are limited, in substance, to changes that someone else considers an improvement, that is not freedom.

However, 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.

In the GNU Project, we use copyleft to protect these freedoms legally for everyone. But noncopylefted free software also exists. We believe there are important reasons why it is better to use copyleft, but if your program is noncopylefted free software, it is still basically ethical. (See "Categories of Free and Nonfree Software" (119) for a description of how "free software," "copylefted software" and other categories of software relate to each other.)

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 export regulations as a condition of any of the essential freedoms.

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.

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 "Words to Avoid (or Use with Care)" (143) for a discussion of these terms. We also have a list of proper translations of "free software" into various languages (393).

Finally, 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.

If you are interested in whether a specific license qualifies as a free software license, see our list of licenses, at http://gnu.org/licenses/license-list.html. 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 licens really is a free software license and avoid various practical problems.

Beyond Software

Software manuals must be free, 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 is the best-known example.

Any kind of work *can* be free, and the definition of free software has been extended to a definitio of free cultural works [1] applicable to any kind of works.

Endnotes

1 See http://freedomdefined.org.

Chapter 2
The GNU Project

The original version of this essay was published in *Open Sources: Voices from the Open Source Revolution*, by Chris DiBona and others (Sebastopol: O'Reilly Media, 1999), under the title "The GNU Operating System and the Free Software Movement." This version is part of *Free Software, Free Society: Selected Essays of Richard M. Stallman*, 2nd ed. (Boston: GNU Press, 2010).

The First Software-Sharing Community

When I started working at the MIT Artificial Intelligence Lab in 1971, I became part of a software-sharing community that had existed for many years. Sharing of software was not limited to our particular community; it is as old as computers, just as sharing of recipes is as old as cooking. But we did it more than most.

The AI Lab used a timesharing operating system called ITS (the Incompatible Timesharing System) that the lab's staff hackers [1] had designed and written in assembler language for the Digital PDP-10, one of the large computers of the era. As a member of this community, an AI Lab staff system hacker, my job was to improve this system.

We did not call our software "free software," because that term did not yet exist; but that is what it was. Whenever people from another university or a company wanted to port and use a program, we gladly let them. If you saw someone using an unfamiliar and interesting program, you could always ask to see the source code, so that you could read it, change it, or cannibalize parts of it to make a new program.

The Collapse of the Community

The situation changed drastically in the early 1980s when Digital discontinued the PDP-10 series. Its architecture, elegant and powerful in the 60s, could not extend naturally to the larger address spaces that were becoming feasible in the 80s. This meant that nearly all of the programs composing ITS were obsolete.

The AI Lab hacker community had already collapsed, not long before. In 1981, the spin-off company Symbolics had hired away nearly all of the hackers from the AI Lab, and the depopulated community was unable to maintain itself. (The book *Hackers*, by Steve Levy, describes these events, as well as giving a clear picture of this community in its prime.) When the AI Lab bought a new PDP-10 in 1982, its administrators decided to use Digital's nonfree timesharing system instead of ITS.

The modern computers of the era, such as the VAX or the 68020, had their own operating systems, but none of them were free software: you had to sign a nondisclosure agreement even to get an executable copy.

This meant that the first step in using a computer was to promise not to help your neighbor. A cooperating community was forbidden. The rule made by the owners of proprietary software was, "If you share with your neighbor, you are a pirate. If you want any changes, beg us to make them."

The idea that the proprietary software social system—the system that says you are not allowed to share or change software—is antisocial, that it is unethical, that it is simply wrong, may come as a surprise to some readers. But what else could we say about a system based on dividing the public and keeping users helpless? Readers who find the idea surprising may have taken the proprietary software social system as a given, or judged it on the terms suggested by proprietary software businesses. Software publishers have worked long and hard to convince people that there is only one way to look at the issue.

When software publishers talk about "enforcing" their "rights" or "stopping piracy," what they actually *say* is secondary. The real message of these statements is in the unstated assumptions they take for granted, which the public is asked to accept without examination. Let's therefore examine them.

One assumption is that software companies have an unquestionable natural right to own software and thus have power over all its users. (If this were a natural right, then no matter how much harm it does to the public, we could not object.) Interestingly, the US Constitution and legal tradition reject this view; copyright is not a natural right, but an artificial government-imposed monopoly that limits the users' natural right to copy.

Another unstated assumption is that the only important thing about software is what jobs it allows you to do—that we computer users should not care what kind of society we are allowed to have.

A third assumption is that we would have no usable software (or would never have a program to do this or that particular job) if we did not offer a company power over the users of the program. This assumption may have seemed plausible, before the free software movement demonstrated that we can make plenty of useful software without putting chains on it.

If we decline to accept these assumptions, and judge these issues based on ordinary commonsense morality while placing the users first, we arrive at very different conclusions. Computer users should be free to modify programs to fit their needs, and free to share software, because helping other people is the basis of society.

There is no room here for an extensive statement of the reasoning behind this conclusion, so I refer the reader to the article "Why Software Should Not Have Owners" (55).

A Stark Moral Choice

With my community gone, to continue as before was impossible. Instead, I faced a stark moral choice.

The easy choice was to join the proprietary software world, signing nondisclosure agreements and promising not to help my fellow hacker. Most likely I would also be developing software that was released under nondisclosure agreements, thus adding to the pressure on other people to betray their fellows too.

I could have made money this way, and perhaps amused myself writing code. But I knew that at the end of my career, I would look back on years of building walls to divide people, and feel I had spent my life making the world a worse place.

I had already experienced being on the receiving end of a nondisclosure agreement, when someon refused to give me and the MIT AI Lab the source code for the control program for our printer. (The lack of certain features in this program made use of the printer extremely frustrating.) So I could not tell myself that nondisclosure agreements were innocent. I was very angry when he refused to share with us; I could not turn around and do the same thing to everyone else.

Another choice, straightforward but unpleasant, was to leave the computer field. That way my skills would not be misused, but they would still be wasted. I would not be culpable for dividing and restricting computer users, but it would happen nonetheless.

So I looked for a way that a programmer could do something for the good. I asked myself, was there a program or programs that I could write, so as to make a community possible once again?

The answer was clear: what was needed first was an operating system. That is the crucial softwar for starting to use a computer. With an operating system, you can do many things; without one, you cannot run the computer at all. With a free operating system, we could again have a community of cooperating hackers—and invite anyone to join. And anyone would be able to use a computer withou starting out by conspiring to deprive his or her friends.

As an operating system developer, I had the right skills for this job. So even though I could not take success for granted, I realized that I was elected to do the job. I chose to make the system compatible with Unix so that it would be portable, and so that Unix users could easily switch to it. Th name GNU was chosen, following a hacker tradition, as a recursive acronym for “GNU’s Not Unix.”

An operating system does not mean just a kernel, barely enough to run other programs. In the 1970s, every operating system worthy of the name included command processors, assemblers, compilers, interpreters, debuggers, text editors, mailers, and much more. ITS had them, Multics had them, VMS had them, and Unix had them. The GNU operating system would include them too.

Later I heard these words, attributed to Hillel: [2]

> If I am not for myself, who will be for me? If I am only for myself, what am I? If not now, when?

The decision to start the GNU Project was based on a similar spirit.

Free as in Freedom

The term “free software” is sometimes misunderstood—it has nothing to do with price. It is about freedom. Here, therefore, is the definition of free software.

A program is free software, for you, a particular user, if:

- You have the freedom to run the program as you wish, for any purpose.
- You have the freedom to modify the program to suit your needs. (To make this freedom effectiv

in practice, you must have access to the source code, since making changes in a program without having the source code is exceedingly difficult.)

- You have the freedom to redistribute copies, either gratis or for a fee.
- You have the freedom to distribute modified versions of the program, so that the community can benefit from your improvements.

Since "free" refers to freedom, not to price, there is no contradiction between selling copies and free software. In fact, the freedom to sell copies is crucial: collections of free software sold on CD-ROMs are important for the community, and selling them is an important way to raise funds for free software development. Therefore, a program which people are not free to include on these collections is not free software.

Because of the ambiguity of "free," people have long looked for alternatives, but no one has found a better term. The English language has more words and nuances than any other, but it lacks a simple, unambiguous, word that means "free," as in freedom—"unfettered" being the word that comes closest in meaning. Such alternatives as "liberated," "freedom," and "open" have either the wrong meaning or some other disadvantage.

GNU Software and the GNU System

Developing a whole system is a very large project. To bring it into reach, I decided to adapt and use existing pieces of free software wherever that was possible. For example, I decided at the very beginning to use TEX as the principal text formatter; a few years later, I decided to use the X Window System rather than writing another window system for GNU.

Because of this decision, the GNU system is not the same as the collection of all GNU software. The GNU system includes programs that are not GNU software, programs that were developed by other people and projects for their own purposes, but which we can use because they are free software.

Commencing the Project

In January 1984 I quit my job at MIT and began writing GNU software. Leaving MIT was necessary so that MIT would not be able to interfere with distributing GNU as free software. If I had remained on the staff, MIT could have claimed to own the work, and could have imposed their own distribution terms, or even turned the work into a proprietary software package. I had no intention of doing a large amount of work only to see it become useless for its intended purpose: creating a new software-sharing community.

However, Professor Winston, then the head of the MIT AI Lab, kindly invited me to keep using the lab's facilities.

The First Steps

Shortly before beginning the GNU Project, I heard about the Free University Compiler Kit, also known as VUCK. (The Dutch word for "free" is written with a *v*.) This was a compiler designed to handle multiple languages, including C and Pascal, and to support multiple target machines. I wrote to its author asking if GNU could use it.

He responded derisively, stating that the university was free but the compiler was not. I therefore decided that my first program for the GNU Project would be a multilanguage, multiplatform compile

Hoping to avoid the need to write the whole compiler myself, I obtained the source code for the Pastel compiler, which was a multiplatform compiler developed at Lawrence Livermore Lab. It supported, and was written in, an extended version of Pascal, designed to be a system-programming language. I added a C front end, and began porting it to the Motorola 68000 computer. But I had to give that up when I discovered that the compiler needed many megabytes of stack space, while the available 68000 Unix system would only allow 64k.

I then realized that the Pastel compiler functioned by parsing the entire input file into a syntax tree, converting the whole syntax tree into a chain of "instructions," and then generating the whole output file, without ever freeing any storage. At this point, I concluded I would have to write a new compiler from scratch. That new compiler is now known as GCC; none of the Pastel compiler is used in it, but I managed to adapt and use the C front end that I had written. But that was some years later; first, I worked on GNU Emacs.

GNU Emacs

I began work on GNU Emacs in September 1984, and in early 1985 it was beginning to be usable. Th enabled me to begin using Unix systems to do editing; having no interest in learning to use vi or ed, I had done my editing on other kinds of machines until then.

At this point, people began wanting to use GNU Emacs, which raised the question of how to distribute it. Of course, I put it on the anonymous ftp server on the MIT computer that I used. (This computer, `prep.ai.mit.edu`, thus became the principal GNU ftp distribution site; when it was decommissioned a few years later, we transferred the name to our new ftp server.) But at that time, many of the interested people were not on the Internet and could not get a copy by ftp. So the questio was, what would I say to them?

I could have said, "Find a friend who is on the net and who will make a copy for you." Or I could have done what I did with the original PDP-10 Emacs: tell them, "Mail me a tape and a SASE (self-addressed stamped envelope), and I will mail it back with Emacs on it." But I had no job, and I was looking for ways to make money from free software. So I announced that I would mail a tape to whoever wanted one, for a fee of $150. In this way, I started a free software distribution business, the precursor of the companies that today distribute entire Linux-based GNU systems.

Is a Program Free for Every User?

If a program is free software when it leaves the hands of its author, this does not necessarily mean it will be free software for everyone who has a copy of it. For example, public domain software (software that is not copyrighted) is free software; but anyone can make a proprietary modified version of it. Likewise, many free programs are copyrighted but distributed under simple permissive licenses which allow proprietary modified versions.

The paradigmatic example of this problem is the X Window System. Developed at MIT, and released as free software with a permissive license, it was soon adopted by various computer

companies. They added X to their proprietary Unix systems, in binary form only, and covered by the same nondisclosure agreement. These copies of X were no more free software than Unix was.

The developers of the X Window System did not consider this a problem—they expected and intended this to happen. Their goal was not freedom, just "success," defined as "having many users." They did not care whether these users had freedom, only about having many of them.

This led to a paradoxical situation where two different ways of counting the amount of freedom gave different answers to the question, "Is this program free?" If you judged based on the freedom provided by the distribution terms of the MIT release, you would say that X was free software. But if you measured the freedom of the average user of X, you would have to say it was proprietary softwar Most X users were running the proprietary versions that came with Unix systems, not the free versio

Copyleft and the GNU GPL

The goal of GNU was to give users freedom, not just to be popular. So we needed to use distribution terms that would prevent GNU software from being turned into proprietary software. The method we use is called "copyleft." [3]

Copyleft uses copyright law, but flips it over to serve the opposite of its usual purpose: instead of means for restricting a program, it becomes a means for keeping the program free.

The central idea of copyleft is that we give everyone permission to run the program, copy the program, modify the program, and distribute modified versions—but not permission to add restrictions of their own. Thus, the crucial freedoms that define "free software" are guaranteed to everyone who has a copy; they become inalienable rights.

For an effective copyleft, modified versions must also be free. This ensures that work based on ours becomes available to our community if it is published. When programmers who have jobs as programmers volunteer to improve GNU software, it is copyleft that prevents their employers from saying, "You can't share those changes, because we are going to use them to make our proprietary version of the program."

The requirement that changes must be free is essential if we want to ensure freedom for every use of the program. The companies that privatized the X Window System usually made some changes to port it to their systems and hardware. These changes were small compared with the great extent of X, but they were not trivial. If making changes were an excuse to deny the users freedom, it would be easy for anyone to take advantage of the excuse.

A related issue concerns combining a free program with nonfree code. Such a combination would inevitably be nonfree; whichever freedoms are lacking for the nonfree part would be lacking for the whole as well. To permit such combinations would open a hole big enough to sink a ship. Therefore, crucial requirement for copyleft is to plug this hole: anything added to or combined with a copylefte program must be such that the larger combined version is also free and copylefted.

The specific implementation of copyleft that we use for most GNU software is the GNU General Public License, or GNU GPL for short. We have other kinds of copyleft that are used in specific circumstances. GNU manuals are copylefted also, but use a much simpler kind of copyleft, because

the complexity of the GNU GPL is not necessary for manuals. [4]

The Free Software Foundation

As interest in using Emacs was growing, other people became involved in the GNU Project, and we decided that it was time to seek funding once again. So in 1985 we created the Free Software Foundation (FSF), a tax-exempt charity for free software development. The FSF also took over the Emacs tape distribution business; later it extended this by adding other free software (both GNU and non-GNU) to the tape, and by selling free manuals as well.

Most of the FSF's income used to come from sales of copies of free software and of other related services (CD-ROMs of source code, CD-ROMs with binaries, nicely printed manuals, all with the freedom to redistribute and modify), and Deluxe Distributions (distributions for which we built the whole collection of software for the customer's choice of platform). Today the FSF still sells manual and other gear, but it gets the bulk of its funding from members' dues. You can join the FSF at `http://fsf.org/join`.

Free Software Foundation employees have written and maintained a number of GNU software packages. Two notable ones are the C library and the shell. The GNU C library is what every program running on a GNU/Linux system uses to communicate with Linux. It was developed by a member of the Free Software Foundation staff, Roland McGrath. The shell used on most GNU/Linux systems is BASH, the Bourne Again Shell, [5] which was developed by FSF employee Brian Fox.

We funded development of these programs because the GNU Project was not just about tools or a development environment. Our goal was a complete operating system, and these programs were needed for that goal.

Free Software Support

The free software philosophy rejects a specific widespread business practice, but it is not against business. When businesses respect the users' freedom, we wish them success.

Selling copies of Emacs demonstrates one kind of free software business. When the FSF took ove that business, I needed another way to make a living. I found it in selling services relating to the free software I had developed. This included teaching, for subjects such as how to program GNU Emacs and how to customize GCC, and software development, mostly porting GCC to new platforms.

Today each of these kinds of free software business is practiced by a number of corporations. Some distribute free software collections on CD-ROM; others sell support at levels ranging from answering user questions, to fixing bugs, to adding major new features. We are even beginning to see free software companies based on launching new free software products.

Watch out, though—a number of companies that associate themselves with the term "open source actually base their business on nonfree software that works with free software. These are not free software companies, they are proprietary software companies whose products tempt users away from freedom. They call these programs "value-added packages," which shows the values they would like us to adopt: convenience above freedom. If we value freedom more, we should call them "freedom-

subtracted" packages.

Technical Goals

The principal goal of GNU is to be free software. Even if GNU had no technical advantage over Unix it would have a social advantage, allowing users to cooperate, and an ethical advantage, respecting th user's freedom.

But it was natural to apply the known standards of good practice to the work—for example, dynamically allocating data structures to avoid arbitrary fixed size limits, and handling all the possible 8-bit codes wherever that made sense.

In addition, we rejected the Unix focus on small memory size, by deciding not to support 16-bit machines (it was clear that 32-bit machines would be the norm by the time the GNU system was finished), and to make no effort to reduce memory usage unless it exceeded a megabyte. In programs for which handling very large files was not crucial, we encouraged programmers to read an entire input file into core, then scan its contents without having to worry about I/O.

These decisions enabled many GNU programs to surpass their Unix counterparts in reliability and speed.

Donated Computers

As the GNU Project's reputation grew, people began offering to donate machines running Unix to the project. These were very useful, because the easiest way to develop components of GNU was to do it on a Unix system, and replace the components of that system one by one. But they raised an ethical issue: whether it was right for us to have a copy of Unix at all.

Unix was (and is) proprietary software, and the GNU Project's philosophy said that we should not use proprietary software. But, applying the same reasoning that leads to the conclusion that violence in self defense is justified, I concluded that it was legitimate to use a proprietary package when that was crucial for developing a free replacement that would help others stop using the proprietary package.

But, even if this was a justifiable evil, it was still an evil. Today we no longer have any copies of Unix, because we have replaced them with free operating systems. If we could not replace a machine's operating system with a free one, we replaced the machine instead.

The GNU Task List

As the GNU Project proceeded, and increasing numbers of system components were found or developed, eventually it became useful to make a list of the remaining gaps. We used it to recruit developers to write the missing pieces. This list became known as the GNU Task List. In addition to missing Unix components, we listed various other useful software and documentation projects that, w thought, a truly complete system ought to have.

Today, [6] hardly any Unix components are left in the GNU Task List—those jobs had been done

aside from a few inessential ones. But the list is full of projects that some might call “applications.” Any program that appeals to more than a narrow class of users would be a useful thing to add to an operating system.

Even games are included in the task list—and have been since the beginning. Unix included games, so naturally GNU should too. But compatibility was not an issue for games, so we did not follow the list of games that Unix had. Instead, we listed a spectrum of different kinds of games that users might like.

The GNU Library GPL

The GNU C library uses a special kind of copyleft called the GNU Library General Public License, [7] which gives permission to link proprietary software with the library. Why make this exception?

It is not a matter of principle; there is no principle that says proprietary software products are entitled to include our code. (Why contribute to a project predicated on refusing to share with us?) Using the LGPL for the C library, or for any library, is a matter of strategy.

The C library does a generic job; every proprietary system or compiler comes with a C library. Therefore, to make our C library available only to free software would not have given free software any advantage—it would only have discouraged use of our library.

One system is an exception to this: on the GNU system (and this includes GNU/Linux), the GNU library is the only C library. So the distribution terms of the GNU C library determine whether it is possible to compile a proprietary program for the GNU system. There is no ethical reason to allow proprietary applications on the GNU system, but strategically it seems that disallowing them would d more to discourage use of the GNU system than to encourage development of free applications. That is why using the Library GPL is a good strategy for the C library.

For other libraries, the strategic decision needs to be considered on a case-by-case basis. When a library does a special job that can help write certain kinds of programs, then releasing it under the GPL, limiting it to free programs only, is a way of helping other free software developers, giving the an advantage against proprietary software.

Consider GNU Readline, a library that was developed to provide command-line editing for BASH Readline is released under the ordinary GNU GPL, not the Library GPL. This probably does reduce th amount Readline is used, but that is no loss for us. Meanwhile, at least one useful application has bee made free software specifically so it could use Readline, and that is a real gain for the community.

Proprietary software developers have the advantages money provides; free software developers need to make advantages for each other. I hope some day we will have a large collection of GPL-covered libraries that have no parallel available to proprietary software, providing useful modules to serve as building blocks in new free software, and adding up to a major advantage for further free software development.

Scratching an Itch?

Eric Raymond [8] says that “Every good work of software starts by scratching a developer’s personal

itch." [9] Maybe that happens sometimes, but many essential pieces of GNU software were developed in order to have a complete free operating system. They come from a vision and a plan, not from impulse.

For example, we developed the GNU C library because a Unix-like system needs a C library, BASH because a Unix-like system needs a shell, and GNU tar because a Unix-like system needs a tar program. The same is true for my own programs—the GNU C compiler, GNU Emacs, GDB and GNU Make.

Some GNU programs were developed to cope with specific threats to our freedom. Thus, we developed gzip to replace the Compress program, which had been lost to the community because of the LZW patents. We found people to develop LessTif, and more recently started GNOME and Harmony, to address the problems caused by certain proprietary libraries (see below). We are developing the GNU Privacy Guard to replace popular nonfree encryption software, because users should not have to choose between privacy and freedom.

Of course, the people writing these programs became interested in the work, and many features were added to them by various people for the sake of their own needs and interests. But that is not wh the programs exist.

Unexpected Developments

At the beginning of the GNU Project, I imagined that we would develop the whole GNU system, then release it as a whole. That is not how it happened.

Since each component of the GNU system was implemented on a Unix system, each component could run on Unix systems long before a complete GNU system existed. Some of these programs became popular, and users began extending them and porting them—to the various incompatible versions of Unix, and sometimes to other systems as well.

The process made these programs much more powerful, and attracted both funds and contributors to the GNU Project. But it probably also delayed completion of a minimal working system by several years, as GNU developers' time was put into maintaining these ports and adding features to the existing components, rather than moving on to write one missing component after another.

The GNU Hurd

By 1990, the GNU system was almost complete; the only major missing component was the kernel. We had decided to implement our kernel as a collection of server processes running on top of Mach. Mach is a microkernel developed at Carnegie Mellon University and then at the University of Utah; the GNU Hurd is a collection of servers (i.e., a herd of GNUs) that run on top of Mach, and do the various jobs of the Unix kernel. The start of development was delayed as we waited for Mach to be released as free software, as had been promised.

One reason for choosing this design was to avoid what seemed to be the hardest part of the job: debugging a kernel program without a source-level debugger to do it with. This part of the job had been done already, in Mach, and we expected to debug the Hurd servers as user programs, with GDB.