Why is software freedom useful, and what does it mean ?

Ian Jackson <ian@chiark.greenend.org.uk>

Debian Project

SANE '98 — 18-20 November 1998

Abstract

Besides the moral and philosophical reasons for preferring free software, free software enjoys many practical advantages in its design, development, deployment, support, maintenance and reuse. Most of these benefits are already recognised and used by the free software community. These, and the resulting quality benefit, are now becoming very important for the spread of free software into the nonspecialist and business world. In this paper I shall describe these benefits and explain how they flow from the freedom of the software. I'll then examine what kinds of freedoms are necessary for these advantages to operate, and what the effects of various common restrictions are on them. Finally, I shall present a set of loose criteria which can be applied to a particular piece of software to see if it is sufficiently free to obtain these practical benefits (if not necessarily the moral ones).

1. Introduction

Free software[1] has become increasingly prominent in recent years. However, it has been around for quite a long time now; perhaps the clearest statement of the original goals of at least part of the free software movement was made in 1983 by Richard Stallman, one of the community's founding fathers, in the GNU Manifesto [GNU85m].

Since then, the free software movement has grown from a few diehards to thousands of participants - individuals, organisations and businesses - and our software is more widely used than the flagship operating system being touted by the current major monopolist supplier of general purpose computer system software [Young98].

2. Why use and support free software ?

Richard Stallman and many others have argued very cogently against the current system of copyright, proprietary software, and so forth [GNU98o].

Many of these arguments are based on moral and philosophical grounds. While many people in the free software community agree with them, it is obvious that not everyone finds these reasons for using and supporting free software very accessible, and some people even find them offputting. It is a shame that this is the case, for these arguments for supporting free software and free information contain concepts which are some of the most illuminating, and most challenging to some of current society's deficiencies, to come out of computing.

However, free software is also very often of higher quality, cheaper to obtain, use and maintain, easier to support and obtain support for, and more flexible, than proprietary software. These form very sound practical grounds for preferring to use and support free software, and it is these good properties of free software, and their causes and implications, that are discussed in this paper.

3. Mechanisms which benefit users and developers of free software

3.1. Technically-driven control of development

Most free software is initially written and/or has a canonical version maintained by people who have a great deal of personal commitment to the particular software. This commitment greatly reduces the temptation to cut corners, make do with nearly good enough, and to compromise on key technical issues. Working for love rather than money on complex creative tasks generally improves creativity and motivation [Kohn87].

This technically-driven control contrasts clearly with many of the problems found by proprietary and commercially-written software. One of its main problems is that it's often written to a deadline (following the

[1] Note that the 'free' in free software refers to freedom, not to price.

frequently-quoted maxim that the first to deploy wins), and when time runs short it is quality rather than scope which is made to suffer.

Because of its technically-driven control, free software usually does not suffer from these problems. The technical focus sometimes makes for a lower short-term takeup by the potential user base, but the long-term future is much brighter, both because of the higher level of satisfaction of users who do take it up and because of the extensibility and flexibility which can be built into software if the time is taken to do so.

Crucial to the maintenance of this technical focus is the ability of a competing developer or team to step in if the existing author or maintainer is doing a bad job for any reason. When this happens, developers who agree with the new team will usually choose to base their work on the new version as it will make their own work easier. In this way the better-maintained version gets better testing and support from the general free software community, and thus the correct technical decision can be made to stick.

3.2. Choose your software freely, without lock-in

Proprietary software frequently promotes lock-in: it is in the interests of the producers of proprietary programs to make it hard to change your mind and go for a competing product instead. The lack of openness makes it possible for the supplier to prevent their competitors from making it easy to switch.

Because users cannot try out proprietary software before they buy, the developers of proprietary software are forced to compete with one another on the basis of lists of features and sometimes pretty user interfaces. Factors such as completeness, robustness, reliability and performance are not as easily measurable and it's often not possible to find out that a mistake has been made choosing proprietary software until it is too late and you are committed.

On the other hand, free software can be tested before use, without as much commitment. If you find that a piece of free software doesn't meet your needs, or your needs change, the availability of source code to the authors of competing programs will make it easy for a transition route to be provided.

3.3. Amicable disagreement between competing authors

In software even experienced and knowledgeable programmers sometimes disagree about the best way to do something; writing software is often a case of striking out into the unknown world of possible solutions. Experienced software developers are aware of the phenomenon that a technical decision made without enough disagreement and argument often turns out later to have been wrong; furthermore, the very best system architects and software engineers are even more likely to disagree with each other about the correct way forward.

Sometimes this happens with free software, and two groups of developers will decide either to take development of the same program in two different directions, or to write a different program to achieve the same purpose.

However, when it does happen, the openness of the specifications and source code mean that the two competing programs can often share new ideas and even new code. Usually the level of compatibility between the competing systems will be high, as each of the competing communities works to share in the benefits of the work done by the other(s).

Despite this, each program is still free to pursue the goals and technical decisions which set it apart from the other, whether this is a question of trying to satisfy a different set of requirements, having a different answer to the same technical problem, or interest in experimenting with, developing and deploying novel technology.

If and when the developers of such competing programs decide that the issue has been resolved, they can and often do re-merge their development paths, producing a hybrid strengthened by its time as two separate systems.

In contrast, the proprietary software world frequently produces two competing and completely incompatible versions merely to satisfy the conventional requirements for a proprietary software business. Furthermore, it is in the suppliers' interests to make their version different from their competitors', and in particular hard to migrate away from. With closed systems they have the means to do so. Often companies in such oligopolies are quite happy with their situation and are not very interested in innovation. Innovative products from new players in the market find it difficult to achieve penetration because their potential users are locked into the existing solutions and their control by the dominant players.

3.4. Research and innovation with reuse and rapid deployment

When academic and independent researchers wish to experiment with new ideas in computing, they often prefer to use free software to do so. Free software does not come with restrictions which limit the researchers' ability to try out modifications, and source licences do not cost small fortunes.

The availability of a large body of software to base innovations on is invaluable in this context; the researcher can concentrate on the novel features in their experiment.

Furthermore, most researchers cherish the hope that the fruits of their research will become widely used. What better way to achieve this than to distribute their experiments and prototypes, and allow the free software community to transform their ideas into production-quality software engineering and distribute it to the world ?

3.5. No monopoly on support

When you use proprietary software, only its supplier can give you support. If you discover a bug that they are not interested in fixing, or want a feature that they do not plan to add, there is little you can do unless you have enormous purchasing power. In today's world where a single company dominates much of the operating system market users have little real choice, and therefore little real leverage. Sometimes proprietary software companies will offer a level of support adequate to your needs, but the price will often be exorbitant.

On the other hand, as a user of free software, support can be provided by any person or organisation with the relevant skills and experience. You can negotiate for a support contract with an independent software support house (of which several now exist), or you can commission work on an ad hoc basis. Alternatively, a large organisation or technically expert individual may decide to acquire the relevant expertise and can provide support to itself.

If the company which supplied your proprietary software is taken over or decides to withdraw support for it you usually have no effective recourse or solution; sometimes organisations enter into complicated agreements regarding 'source code escrow' to solve this problem. With free software this is unnecessary; the original supplier cannot prevent you from seeking support elsewhere.

3.6. Cooperative development

Free software does not suffer from the usual barriers and obstacles to cooperation created by the operation of copyright law and by the necessities of proprietary software development. After a company has sold a licence to use proprietary software it is in a zero-sum game with its customer; in many cases, competing developers of proprietary software are very nearly in a zero-sum game with each other. The risks of cooperation and of losing 'competitive advantage' are seen as very great.

When two competing companies sell and support the same free software, one company's gain of a customer is also indirectly the other's gain, because the resulting extra resources which the first company puts into the software development will also indirectly benefit the second. Both companies would like to see the market share of their common software increase, even as they compete with each other for the best support, reviews, marketing and packaging.

When a distributor of free software finds and fixes a bug it is in their interests to get their bugfix included in the standard version sent to everyone, to ensure that their version and the standard version stay in sync: this means that the distributor will easily be able to pick up new and improved standard versions.

When different pieces of free software interact with each other, any problems are usually resolved in the 'right' place by fixing the program that was faulty. This is so because the person who's experiencing the problem will find it easier to fix it in the right place and get the fix distributed, even if it's not their own program; fixing bugs by working around them is messy and usually harder. With proprietary software it is usually not possible for the interoperation problems to be fixed except by the one supplier to whom the customer is complaining, and sometimes the customer has noone to complain to at all.

The lack of openness also makes it hard to identify the real causes of such problems - people who have worked on proprietary device drivers and network protocols are all too familiar with the ‘I can’t show you my source code and you can’t show me yours’ problem.

On another level, the general amount of mutual goodwill in the free software community is much higher than between organisations supplying proprietary software. This ‘feel good factor’ means that when an individual developer finds a bug or missing feature in a piece of free software, they often choose to add it. This results in the phenomenon often observed by free software developers, that they often get a contribution of development effort from the community; the contributed effort is often as much as the work the original authors put into the program themselves.

In general, the common goal shared by all the participants on the free software movement - improvement in the quality and range of free software available - engender much more cooperation than is usual in proprietary circles. The result is usually much greater productivity.

4. What must we do to achieve these benefits ?

As we’ve seen, free software has major practical benefits as well as the well-known moral arguments in its favour. So, what kind of freedom must software have to acquire these benefits ? Firstly, I’ll go through the practical benefits one at a time, and see what the requirements are for each. As we’ll see, they are very similar.

4.1. Point-by-point requirements

To have the technically driven development discussed in section 3.1, it must be possible for people other than the current authors/maintainers to create a competing version of the original program and, if the situation persists, eventually displace the original. Even the best software developers are sometimes wrong, and without this ultimate sanction they can never be shown to be, so their mistakes will persist.

In order to prevent lock-in (section 3.2), it must be possible for people other than the current authors/maintainers to use information in the source code of one program, and take pieces of the source code itself, to create a different competing program which is nevertheless compatible. Without at least the ability to read the source code to find out what a program is doing, making a compatible program is often very difficult, particularly if the program’s authors and the authors of its documentation want to make it so. If they can restrict the ability to copy parts of the source code the authors of the original program will find it easy to maintain a constantly moving target, if they wish, and those writing the compatible program will find it difficult to catch up.

In order to allow developers to disagree about how to design or develop a program (section 3.3), and to each try out their own approach without acrimony and legal problems, it must be possible for competing projects to use each other’s ideas and code freely so far as the technical arrangements permit. In the case of a single program going two different ways, there must be no legal or social barriers to each group distributing their own version, possibly even when the original authors disagree about the correctness of their approach(es), and code from the two versions must be freely mixable to ensure that compatibility can be maintained as far as possible (as noted above).

To allow researchers to work with the software (section 3.4), they will need to make modified and possibly highly experimental and unstable versions of the software, without prior approval from the authors, and they may need to distribute these to other researchers. In order to allow their experiments to be developed and deployed widely, it must be possible for third parties to continue the development and maintenance where the researcher leaves off.

To avoid the monopoly on support (section 3.5), the people doing the support work must be able to fully investigate any problems and add new features. They must then be able to distribute the results to their users, in whatever form the users require. This must be possible without permission from the original authors or copyright holders, who may be unavailable, reluctant, or simply want to make money out of the situation.

To allow cooperation between software developers within the whole community as if they were working together in the same organisation on the same project (section 3.6), all the developers need equal access to each other's programs. They all need to be able to examine and modify them, distribute their modifications, and generally do all the things the original authors would expect to do.

And finally, to maintain the shared community spirit and goodwill between developers, it must be possible for developers to engage in the usual cut-and-thrust of technical activity and discussion without offending one another. They must be able to disagree on technical matters and still remain united in purpose, and not to be slighted when others disagree with their technical decisions and exercise the options I've mentioned above.

4.2. Simple statement of requirements

The common thread running through the requirements list above is that it's necessary for anyone to be able to acquire the source code for free software, modify it, distribute those modifications to people and organisations they have a relationship with or to the world in general, and to distribute the resulting executable programs, just as if they were the original author.

It is also necessary for it to be possible to mix code from two different pieces of free software, to achieve the best levels of reuse.

More detailed discussion of the properties of various licensing restrictions follows.

5. Problems addressed by free software licences

Free software is usually protected by copyright, and licensed to users and developers by its authors. However, free software licences have a very different intent to those for proprietary software. In this section I'll explain some of the key problems that they are intended to solve and discuss some of the typical solutions which have been adopted.

5.1. 'Theft' of effort

Many people who write free software (whether as part of an organisation or as individuals) are worried that the fruits of their labour will be 'taken away' from them by proprietary software companies.

Of course, a proprietary software company cannot really take software away in the sense that they prevent the original author from using it, but unless stopped somehow they can prevent the original author from benefitting from any improvements the company makes. In the worst case this can sometimes make free software authors feel like they're actually unpaid and unrecognised staff for these companies.

Particularly bothersome for some is the lack of reciprocity in such arrangements; the original developers share their system with the world, and yet a particular company then takes it, makes it proprietary, perhaps improves it, and hoards the results.

To prevent this, many people use an 'infectious' licence such as the GNU General Public Licence [GNU91l] from the Free Software Foundation. Such licences typically require any derived software to be distributed with source code under the same licence. The GNU GPL was the first such licence, and there is now a very considerable body of code distributed under it. The copyrights are sometimes owned by the FSF, but more recently many programs (such as the Linux operating system kernel) have copyright shared by many authors, which makes it in practice impossible ever to distribute the program under any licence other than the GPL or one of its successors.

However, there is a problem with infectious licences, which has become relevant recently. If two programs are each released under a different infectious licence it is often impossible to combine code from both programs in a single work and still legally distribute the result. This has only become a problem now because several substantial works been released under infectious licences not compatible with the GNU GPL.

Some in the free software community feel that the solution to this is for the newcomers to fit in with existing practice, and ensure that their licences are GPL-compatible; others are strongly opposed to infectious licences in general and the GPL in particular.

5.2. 'Theft' of control, and of user base and supporting developers

Some free software authors are worried that another developer might make a modified version of their program and by fair means or foul entice away their users, and hence the external developers who help their project.

Others are worried in principle by the idea that another developer might start independently distributing modified versions of their program, and that this would lose what they perceive as their control over the program as the original author.

Authors who are worried about these things often include clauses in their licences which restrict or prevent other people from distributing modified versions, or non-approved executable binaries.

Unfortunately, such restrictions make it impossible to get most of the practical benefits of free software, most of which are directly related to lack of control by the original author. Indeed, the ability of other developers to derive and maintain competing versions of a program even against the wishes of the original authors is key to many of these benefits, and on the rare occasions where it has actually happened the community as a whole has usually benefitted. Furthermore, these restrictions fly against the philosophical and moral arguments for free software.

Consequently, software with such restrictions is usually not classed as free software by most members of the community.

5.3. Reputation tarnished by poor derivative works

Many free software authors are worried that irresponsible developers might make unwise modifications to their software and then distribute the result. They feel that poor software which still bears their name and that of their original product will tarnish the good reputation they hope they have built up for themselves.

Many free software licences therefore impose some kind of requirement to document any changes you make in a modified, derived work, for example by adding prominent notices to source files or a program's version string and help output. Authors who are particularly concerned by this issue sometimes insert clauses in the licence which require any modified version to carry a different name to the original. These requirements are not onerous, provided they do not interfere with the ability to take small portions of a program and include them in another, or to use the modified version as a replacement for the original.

Some authors go further, and (as in the previous section) prohibit derived modified works altogether. This approach is no better when used to solve this problem than the previous one; it takes away nearly all of the benefits of using free software.

As a compromise, a few authors have required that modified versions of the source code be distributed as a combination of the original source and a patch file giving the modifications made. This appears on the surface to be sufficient to get most of the practical benefits, but it breaks down if substantial changes or rearrangements need to be made: for example, if a group wish to split off an independent development effort and go their own way, the size of the patch would fairly rapidly approach the size of the original source code. Such licences can also cause problems if the original copyright holder becomes unavailable or uncooperative for some reason.

5.4. Lack of proper attribution and acknowledgement

Most free software licences have clauses saying that the copyright notices and authorship information may not be removed from the source files. Such restrictions are clearly harmless. Some licences insist that program startup messages or 'about' dialogue boxes contain this information, which is also clearly appropriate and nonintrusive.

However, for many years much software was released under the Berkeley Systems Distribution licences, which required that advertising material relating to the software carry a notice about the software's origin. This may have been appropriate when free software licences were rare and there were few copyright holders of such software. However, modern free operating systems contain components written by many thousands of people, and it would be quite impractical to mention all of their names or organisations in advertising. There is therefore a current trend in the free software community to encourage authors not to impose this restriction on new software, and to try to have it lifted for existing software.

6. Conclusions

We've seen that free software has many practical benefits, as well as the moral and philosophical ones with which the community is acquainted.

These benefits, almost all associated with the deliberate relinquishing of control by the software's authors, include higher quality, better ability to get support, generally greater flexibility on the part of users and developers, and a greater sense of cooperation across different groups and organisations.

The nature of the freedom required to achieve these benefits is that everyone — not just the original authors — has all the necessary abilities to inspect, modify, maintain and distribute the software.

Certain restrictions, such as infectious licences which require derivative works to be released under the same licence, requirements to preserve and/or add notices in the source code and documentation messages, and to distribute modified versions only under a different name, do not necessarily interfere with these benefits.

Other restrictions, particularly restrictions on distribution of modified works and executable binaries (including restrictions on the format for distribution of modified versions), make it hard to realise these benefits. The growing success of free software will be enhanced if software developers and users are aware of the distinction between software which is truly free and that for which the source code may be available but which is encumbered by these onerous restrictions.

With luck and judgement on our part, recognition in the world of the practical benefits of free software will continue to advance. With the continuing spread of free software, not only will we gain and share these practical benefits, but the principle that software and information should be freely available will also become more accepted.

Bibliography

[GNU85m]
Free Software Foundation. *The GNU Manifesto.*
http://www.gnu.org/gnu/manifesto.html, 1985, revised 1993.

[GNU91l]
Free Software Foundation. *GNU General Public License, version 2.*
http://www.gnu.org/copyleft/gpl.html, June 1991.

[GNU98o]
Free Software Foundation. *Why software should not have owners.*
http://www.gnu.org/philosophy/why-free.html, 1998.

[Kohn87]
Alfie Kohn. *Studies Find Reward Often No Motivator*. Boston Globe, 19th January 1987. URL
http://www.gnu.org/philosophy/motivation.html.

[Young98]
Robert F. Young. *Sizing the Linux Market*.
http://www.redhat.com/redhat/linuxmarket.html, March 1998.