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

Ian Jackson
Debian Project
*ian@chiark.greenend.org.uk*
PGP2/1024: 5906F687 BD03ACAD 0D8E602E FCF37657

20th November 1998
SANE'98
Maastricht

# Introduction

## Free software

Freedom, not price.

Increasing prominence now, but around for ages. Richard Stallman, GNU Manifesto.

Grown; first diehards, now thousands. Organisations and businesses too. Winning against the monopolist.

# Why use and support free software ?

## Moral and philosophical arguments

Richard Stallman and others: against copyright, proprietary software. ‘Why software should not have owners’. Many free software people agree.

But, not arguments not accessible for some people; even offputting.

(Is a shame; ideas are illuminating, and challenging to current societal deficiencies.)

## Practical

This talk is about practicalities: quality, cost of obtaining, using, maintaing, supporting.

Will examine these good properties, their causes, their implications.

# Mechanisms which benefit users and developers of free software

## Technically-driven control of development

Most free software written by (or canonical version maintained by) people with personal commitment.

Less temptation to cut corners, make do, compromise on key issues.

Working for love, not money: obviously improves motivation; also creativity. (‘Studies Find Reward Often No Motivator’ - Kohn)

Proprietary software contrasts. Often written to deadline (‘first to deploy wins’). Time always short - then cut quality, not scope.

Free software: usual technical focus can produce lower initial takeup. But future brighter: user satisfaction, extensibility, flexibility - depend on taking time.

Technical focus by technical competition: competing developer or team can step in to take over bad job. Others who agree can migrate. Better-maintained version gets more testing and support, can be made to ‘win’.

# Choose your software freely, without lock-in

Proprietary promotes lock-in: producer wants to force you to stay. Lack of openness makes this easy.

Proprietary usually has no (meaningful) try before buy. Competition based on lists of features, sometimes pretty GUIs. Completeness, robustness, reliability, performance not easily measured; bad choice discovered only after committtment.

Free software can be properly tested before use. Change your mind ? Source availability means that the competing product will probably have easy transition route.

# Amicable disagreement between competing authors

Good people can disagree: unknown territory in world of possibilities. Decisions without disagreement often seem poor later. Very best people disagree the most !

When free software, two groups can take same program different ways, or write different programs. But: competing programs can share ideas and even code. Usually compatibility will be good: projects work to share in the benefits of each others' work.

Even so, each team can pursue on goals and views. If issue resolved, can re-merge, producing stronger hybrid.

Proprietary: competing and incompatible versions for 'business reasons'. Migrate away is particularly hard. Oligopolies comfortable for companies, so innovation squashed by lack of penetration.

# Research and innovation with reuse and rapid deployment

Academics / researchers: want to experiment. Free software does not restrict ability to try out modifications, no astronomical source licence charge.

Large body of free software to base innovations on: so can concentrate on new ideas, reuse standard implementations of old ideas.

Researchers want widespread use. How better than to make it free software ? Community will often make production engineering out of research code.

## No monopoly on support

Proprietary: only supplier can give you support. If supplier won't fix, or add feature, you are stuck.

Free software: anyone with skills can support. Can get a standard support contract from 3rd party, or commission ad hoc. Large organisation or individual expert can self-support.

Take over / product discontinued: free software not affected. Free software always has a future if you need it to have one.

# Cooperative development

Usual operation of copyright law makes barriers to cooperation.

After sold licence to proprietary software, developer is often in zero-sum game with customer. Competing developers often in nearly zero-sum game with each other. Risks of cooperation (losing ‘cooperative advantage’) seen as very great.

If two companies sell and support same free software ? One company’s gain of a customer is indirectly the other’s too - extra resources put into shared codebase. Companies can cooperate to improve common market share and common software base, while competing over support, reviews, marketing and packaging.

Free software distributor finds and fixes bug ? Best to send it ‘upstream’, to avoid necessity to re-patch.

Problem-solving with free software is easy: no ‘I can’t show you my source and you can’t show me yours’. Anyone can fix problem in anyone else’s program. With proprietary software, can play ‘pass the buck’ and customer loses out.

Free software can fix problem in ‘right’ place, because it is easy to distribute the fix to the broken code.
With proprietary software, often have to work around problems.

Mutual goodwill very high in free software community. So, bugs or missing features found by developers often fixed by them and passed on to author. Often contributed work is as much or more than original effort.

Common goal: improvement of quality and range of free software. Therefore, better productivity.

# What must we do to achieve these benefits ?

So: practical benefits as well as moral arguments. What kind of freedom is required ?

Consider benefits one at a time, identify requirements.

Will see that requirements all very similar.

# Point-by-point requirements

Technically driven development ? Possible for people other than the current authors/maintainers to create competing version, and eventually displace original. This ultimate sanction prevents mistakes persisting, even when made by the best people.

Prevent lock-in ? People other than the current authors/maintainers must be able to use source of one program to create competing compatible program. Without this, need reverse engineering. If code copying restricted, can make moving target.

Try out different designs ? Competing projects able to use ideas and code freely. Single program forking must be allowed, even if original authors disagree; code must be mixable to ensure most compatibility can be retained.

Researchers ? Modified - highly experimental and perhaps unstable versions - produced and distributed, without prior approval from authors. To incorporate research results into production, third parties must be able to further develop and maintain.

Avoid support monopoly ? Support people must be able to fully investigate and add new features, distribute results - without permission from original authors (who may be unavailable, reluctant, or want to make money).

Cooperation, as if common goal: all developers need effective access to each others programs. Must be able to examine, rebuild and modify them, and generally do all things original authors expect to be able to do.

Maintain community goodwill ? Must be able to engage in technical argument, activity, cut-and-thrust, without offending. Must be able to disagree technically but remain united in purpose, and not be slighted when people disagree with them.

# Simple statement of requirements

Common requirement: anyone can acquire source, 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.

Also necessary to be able to mix code from two different pieces of free software, to achieve best reuse.

Free software usually copyrighted, and licensed to users and developers. But, licences have different intent, to solve different problems.

# Problems addressed by free software licences

## ‘Theft’ of effort

Worry that fruits of labour taken away by proprietary software companies. Company cannot stop original author from using own code, but might prevent author from using company’s improvements. Can feel like unpaid staff. Lack of reciprocity.

Solution: infectious licence; all derived code must be distributed under same licence. Eg, GNU General Public Licence.

But, if two different infectious licences, cannot combine code. Problem only now because subtantial works released under non-GPL infectious licences.

Some feel newcomers should fit in and use GPL. Others are opposed to infectious licences, in particular GPL.

# Problems addressed by licences

## ‘Theft’ of control, and of user base and supporting developers

Some worry that another developer may entice away users and external developers who current help original author’s project. Others worry that independent distribution would lose their control over the program.

Can solve by restricting distribution of modified versions or non-approved binaries.

But, impossible to get most of benefits: most benefits are related to lack of control by original author ! Derive and maintain competing versions is key to many benefits.

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

# Problems addressed by licences

## Reputation tarnished by poor derivative works

Irresponsible developers might make and distribute bad modifications; the original author's and product's good name may be damaged.

Solution: many licences impose requirement to document changes, eg by adding notices to changed files or program's version string or informational text. Can even require different name for modified versions.

These requirement OK, if can still take small portions of program and include in another, or to use modified version as replacement for original.

Again, some authors prohibit derived modified works altogether. No better here.

Some require modified versions distributed as combination of original source patch. On surface, sufficient to get most practical benefits. But, if substantial changes or rearrangements made, breaks down. Eg, if group want to split independent effort, size of patch becomes implausible. Also problematic if original copyright holder unavailable.

# Problems addressed by licences

## Lack of proper attribution and acknowledgement

Most free software requires copyright/authorship information to be retained in source or 'about' dialogues etc.

But: Berkeley Systems Distribution licence, requires advertising to carry notice. Fine if few copyright holders, can get ridiculous now. Current trend is to try to discourage/lift 'obnoxious BSD advertising clause', and to call resulting licence eg 'MIT'.

# Conclusions

Free software practical as well as moral.

Benefits associated with relinquished control by original author: quality, support, flexibility, cooperation.

Freedom required: everyone can inspect, modify, maintain and distribute, just like original authors.

Restrictions usually OK: Infections licences (mostly); requirement to add notices in code and doc messages, change names of modified versions.

Other restrictions bad, eg: restricted distribution of modified works, restricted distribution of binaries. Must distinguish source-available and truly free software.

With luck, recognition for practical benefits will continue to advance. This will make it easier to preach on matters of principle.