GNU ccd2cue

for version 0.5 (March 13, 2015)

Bruno Félix Rezende Ribeiro <oitofelix@gnu.org>

GNU ccd2cue is a CCD sheet to CUE sheet converter. It supports the full extent of CUE sheet format expressiveness, including mixed-mode discs and CD-Text meta-data.

This manual is for GNU ccd2cue version 0.5 (March 13, 2015).

This package is a component of the GNU Operating System and is developed by the GNU Project.

Table of Contents

Preface

> **Manifesto:** On the internet there is a gigantic quantity of optical disc image files in numerous formats. Countless times we need to burn some of them. Some time ago I needed it, but I came across a file format extremely irritating for a Free Software user like me: a CD layout descriptor file, with `.ccd` suffix, generated by a proprietary software called CloneCD. I searched the internet for a way to burn that file on the GNU+Linux-Libre system, but I only found a lot of people asking for a solution on a lot of forums, and getting the unanimous answer: no way! At first I could not believe at that point there was no option. Then, with a little bit of patience and research, I wrote some code to convert that files to a format much more common and accessible, an ad-hoc standard in the GNU operating system: the CUE sheet format. So I could burn a lot of what I wanted! I wondered whether it would be useful for others... and here is the result!
>
> —*Bruno Félix Rezende Ribeiro (oitofelix)*

There is a well known and widely employed proprietary optical disc authoring software called CloneCD[1] which is intended to make a nearly exact copy of audio and/or data optical discs for archival and/or reproduction purposes. When making a copy of an optical disc to hard disk that program can output three files: audio/data raw image, sub-channel data and layout description. The GNU ccd2cue program operates exclusively on the latter to produce a hopefully equivalent[2] layout description in the CUE sheet format and possibly a companion CD-Text meta-data file. Since there are free software packages that can easily and fully handle CUE sheet files, this frees the user from the temptation to use a proprietary program or from the burden of searching for the desired disc data into a supposedly more free software friendly format.

One such program capable of burning a raw image file laid out by a CUE sheet file, as well as converting a CUE sheet file to the TOC sheet format, is *cdrdao*[3].

Notably, GNU ccd2cue can handle:

- Mixed-mode CCD sheet file[4];
- CD-Text meta-data[5];
- Track special sub-code flags[6];

---

[1] Designed to run on an equally freedom-restricting and user-subjugating platform known as Microsoft Windows.

[2] As far as possible to the full extent of CUE sheet format expressiveness.

[3] See http://cdrdao.sourceforge.net/.

[4] A sheet describing the layout of a disc which contains simultaneously audio and data tracks.

[5] A method for storing disc and track relevant information (like album, song and artist names) on a standard compliant audio CD.

[6] In-disc information that determines copy restriction (very sad!) and audio track characteristics like 4 channel audio and track pre-emphasis.

This manual is split in two major logical segments: instruction and reference. In the former resides the manual content targeted to instructional, reasonably tutorial, use. It is composed of the following chapters:

Overview General information about purpose, distribution, copying, community, contribution and donation for this package.

Release Presents information specifically relevant for the current release or subject to change between releases.

Operation Teach how the program works from the user's perspective;

Tutorial Guides the reader step-by-step in using the program for practical common use cases.

Invocation Concerns `ccd2cue` invocation syntax, accepted options and their meanings.

Burning Gives an *important warning* and advice about burning the resulting CUE sheet. **READ THAT!**

In the latter, one can find the reference material comprised of the following appendices:

Compact Disc fields
Describes some Compact Disc information that are common concepts to CCD and CUE sheet formats.

CCD sheet format
Describes the CCD sheet format (at least the part we know about — it is a reverse engineering process, and you are encouraged to help).

CUE sheet format
Describes the CUE sheet format.

Now that you are acquainted with the program and documentation, **HAPPY HACKING!**

1 Overview

This chapter briefly explains for what this package is intended, gives relevant considerations regarding dependencies, configuration, build, installation and use, describes which are the rules for its distribution, how to get a copy of it, how to contact the community, how to fill bug reports, how to contribute to the package, how to make donations to support its development and how to get started hacking the code. Bear in mind that instructions in this chapter are package-specific; for general and in-depth configuration, build and installation instructions refer to the file `INSTALL` present in the top-level directory of the source distribution. If you have checked out the source tree from the VCS repository see Section 1.8 [Hacking], page 6.

For more information about this program you can visit its home page at `https://www.gnu.org/software/ccd2cue/`. If you want to receive notifications about new releases of this program or important issues related to it, subscribe to its mailing list, as described in Section 1.4 [Contact], page 4, or subscribe to the package atom feed `https://savannah.gnu.org/news/atom.php?group=ccd2cue`.

1.1 Description

GNU ccd2cue is a CCD sheet to CUE sheet converter. It supports the full extent of CUE sheet format expressiveness, including mixed-mode discs and CD-Text meta-data. It plays an important role for those who need to use optical disc data which is only available in the proprietary sheet format CCD, but don't want to surrender their freedom. It fills an important gap in the free software world because before its conception it was impossible to use complex forms of optical disc data laid out by CCD sheets in a whole/holy free operating system.

The GNU ccd2cue documentation is also intended to be a reference documentation for both sheet format specifications. That way we can reverse engineer the secret CCD sheet proprietary format only once and then make the information available for developers in order to benefit all free software users that want their software to be interoperable. The CUE sheet format is not a secret, but with this package we take the opportunity to ensure that its specification is available under a free documentation license for the sake of the whole free software community.

1.2 Distribution

This program is *free software*; this means that everyone is free to use it and free to redistribute it under certain conditions. This program is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of this program that they might get from you. The precise conditions are found in the GNU GPL (General Public Licence). The program manual is covered by the GFDL (Appendix D [GNU Free Documentation License], page 56). This license is similar in spirit to the GNU General Public License, but is more suitable for documentation.

1.3 Getting a copy

One way to get a copy of this program is from someone else who has it. You need not ask for our permission to do so, or tell any one else; just copy it. You may also receive this program when you buy a computer. Computer manufacturers are free to distribute copies on the same terms that apply to everyone else. These terms require them to give you the full sources, including whatever changes they may have made, and to permit you to redistribute the program received from them under the usual terms of the GNU General Public License. In other words, the program must be free for you when you get it, not just free for the manufacturer.

If you have access to the Internet, you can get the latest distribution version of this program at `https://ftp.gnu.org/gnu/ccd2cue/`. Please, use a mirror if possible; you will be automatically redirected to the nearest mirror at `http://ftpmirror.gnu.org/ccd2cue/`.

A VCS repository, where the development takes place, is also available. It maintains the full history of modifications of every single source file. You can checkout any given revision of any file or get a snapshot of the entire source tree in a particular desired state. Special build tools, as described in Section 1.8 [Hacking], page 6, are required to build from those checkouts, though. Notably, to stay up to date with the latest developments in the source tree, you can anonymously checkout the repository with the following command:

```
git clone git://git.savannah.gnu.org/ccd2cue.git
```

1.4 Contact

You can get in touch with other users and the developers of this program by subscribing to its mailing list. Anyone is welcome to join the list; to do so, visit ccd2cue's help and support mailing list web interface. To post a message to all the list members, send email to `ccd2cue@gnu.org`. To see the collection of prior postings to the list, visit its archive. You can use this list for all discussion, including asking for help and bug reporting, although the preferred method for reporting bugs is sending a mail to `bug-ccd2cue@gnu.org`, the ccd2cue's bug reporting mailing list. See Section 1.5 [Bug reporting], page 4.

If you feel somewhat chatty, eager for a somewhat more instantaneous response from community, you can join us on our friendly IRC channel: '`irc://irc.freenode.net/ccd2cue`'.

1.5 Bug reporting

If you came across some problem and need help you can contact the community as described in Section 1.4 [Contact], page 4. If you think you found a bug, but is not quite sure about it, you can ask for support sending a mail to `ccd2cue@gnu.org`. We will revise your post, advise you and take the appropriate measures. If you are confident you have found a bug, you can submit a bug report directly to `bug-ccd2cue@gnu.org`. You can subscribe to this ccd2cue's bug reporting mailing list at its web interface. To see the collection of prior reported bugs, visit its archive. Please, when reporting a bug include enough information for the maintainers to reproduce the problem. Generally speaking, that means:

- The contents of any input files necessary to reproduce the bug and command line invocations of the program(s) involved (crucial!).
- A description of the problem and any samples of the erroneous output.
- The version number of the program(s) involved (use `--version`).

- Hardware, operating system, and compiler versions (`uname -a`).
- Unusual options you gave to configure, if any (see `config.status`).
- Anything else that you think would be helpful.

1.6 Contributing

This program is a collaborative effort and we encourage contributions from anyone and everyone — your help is very much appreciated. You can help in many ways:

- Donate to developers in order to support their work. See Section 1.7 [Donating], page 5.
- Write documentation. We are specially in need to complete the CCD sheet format specification.
- Help users in the mailing list and IRC channel.
- Find and report bugs. See Section 1.5 [Bug reporting], page 4.
- Fix reported bugs.
- Implement new feature ideas.
- Write test cases.
- Check the documentation against the implementation.
- Translate the program strings to other languages.

You can join the development team to contribute code and documentation at the development page. Patches are most welcome, but contributed code should follow the *GNU Coding Standards*. If it doesn't, we'll need to find someone to fix the code before we can use it. It is also necessary that the contributor be willing to assign their copyright to the FSF, since the developers plan to make it officially part of the GNU operating system and they want FSF to enforce the program's license. To get started hacking see Section 1.8 [Hacking], page 6.

1.7 Donating

If you find this program useful, please **send a donation** to its developers to support their work. If you use this program at your workplace, please suggest that the company make a donation. We appreciate contributions of any size – donations enable us to spend more time working on this package, and help cover our infrastructure expenses.

If you'd like to make a donation of any value, please send it to the following Bitcoin address:

```
12sKDaBNYekQuRPdrpnbUL4YRDKrzMnY62
```

Since we aren't a tax-exempt organization, we can't offer you a tax deduction, but for all donations over 0.05 BTC, we'd be happy to recognize your contribution on the donors page and on `DONORS` file for the next release.

We are also happy to consider making particular improvements or changes, or giving specific technical assistance, in return for a substantial donation over 0.5 BTC. If you would like to discuss this possibility, write to me at `oitofelix@gnu.org`.

Another possibility is to pay a software maintenance fee. Again, write to me about this at `oitofelix@gnu.org` to discuss how much you want to pay and how much maintenance we can offer in return.

Thanks for your support!

1.8 Hacking

The development sources are available through VCS at Savannah:

https://savannah.gnu.org/git/?group=ccd2cue

If you are getting the sources from a VCS (or change `configure.ac`), you'll need to have Automake, Autoconf and Gettext installed to (re)build. You'll also need help2man. All of these programs are available from `https://ftp.gnu.org/gnu/`.

After getting the VCS sources, and installing the tools above, you can run `./bootstrap && ./configure && make` to do a fresh build. After that first time, running `make` should suffice to rebuild the program with your changes. See file `INSTALL`.

When modifying the sources, or making a distribution, more is needed, as follows:

- This distribution also uses Gnulib to share common files, stored as a submodule in git.
- When updating gettext, besides the normal installation on the system, it is necessary to run `gettextize -f` in this hierarchy to update the `po/` infrastructure. After doing so, rerun `gnulib-tool --import` since otherwise older files will have been imported. See *Gnulib Manual*, for more information.

When committing changes to the repository always create an entry in the `doc/release/latest-news.texi` file for any user-visible changes or additions made. This file is intended to provide the latest release news for the `NEWS.texi` and `ANNOUNCEMENT.texi` files to avoid duplication of information and syncing work. After a release is made the news items should be moved to the `NEWS.texi` file and another news list should be built from scratch in the `ANNOUNCEMENT.texi` file.

2 Release

In this chapter one can find information that are specific to the current release or subject to change between releases. It presents the features this package acquired over time, problems that affect its overall usage and what possibly could happen to it — by our wishes — in the future. Last but not least, people whose contribution for this package are noteworthy are acknowledged.

2.1 Announcement

GNU ccd2cue is a CCD sheet to CUE sheet converter. It supports the full extent of CUE sheet format expressiveness, including mixed-mode discs and CD-Text meta-data. It plays an important role for those who need to use optical disc data which is only available in the proprietary sheet format CCD, but don't want to surrender their freedom. It fills an important gap in the free software world because before its conception it was impossible to use complex forms of optical disc data laid out by CCD sheets in a whole/holy free operating system.

The GNU ccd2cue documentation is also intended to be a reference documentation for both sheet format specifications. That way we can reverse engineer the secret CCD sheet proprietary format only once and then make the information available for developers in order to benefit all free software users that want their software to be interoperable. The CUE sheet format is not a secret, but with this package we take the opportunity to ensure that its specification is available under a free documentation license for the sake of the whole free software community.

News

- Danish, German, Ukrainian and Vietnamese translations.
- Support for localized Unix manual pages. Feature suggested by Mario Blättermann (German translator).
- Respectable Unix manual pages.
- Localizable `--help` meta-variables. Bug reported by Mario Blättermann.
- Numerous grammatical and markup corrections to the user's manual. Patch submitted by Karl Berry.
- Make target `announcegnu` which automatically sends a signed announcement message to `info-gnu@gnu.org`, `ccd2cue@gnu.org` and `coordinator@translationproject.org` mailing lists when a release is ready. This target can only be made in VCS checkouts.
- Make target `fetchpo` which fetches from the Translation Project the latest PO files available . This target can only be made in VCS checkouts.
- Latest release news, manifesto, package's description and version in a single point of maintenance.
- Package meta-information, as release date and build system version, automatically generated at configuration time.
- Announcement message in user's manual and homepage.
- Make target `distdir` can be made when top-level documentation files are missing.

Download

Here are the compressed sources and a GPG detached signature:

```
https://ftp.gnu.org/gnu/ccd2cue/ccd2cue-0.5.tar.gz
https://ftp.gnu.org/gnu/ccd2cue/ccd2cue-0.5.tar.gz.sig
```

Use a mirror for higher download bandwidth:

```
http://ftpmirror.gnu.org/ccd2cue/ccd2cue-0.5.tar.gz
http://ftpmirror.gnu.org/ccd2cue/ccd2cue-0.5.tar.gz.sig
```

Use a `.sig` file to verify that the corresponding file (without the `.sig` suffix) is intact. First, be sure to download both the `.sig` file and the corresponding tarball. Then, run a command like this:

```
gpg --verify ccd2cue-0.5.tar.gz.sig
```

If that command fails because you don't have the required public key, then run this command to import it:

```
gpg --recv-keys 0x28D618AF --keyserver hkp://keys.gnupg.net
```

and rerun the `gpg --verify` command.

This release is signed by *Bruno Félix Rezende Ribeiro*. His key fingerprint is `7CB1 208C 7336 56B7 5962 2500 27B9 C6FD 28D6 18AF`.

This release was bootstrapped with the following tools:

- GNU Autoconf 2.69
- GNU Automake 1.14.1
- GNU Texinfo 5.2

Links

Homepage https://www.gnu.org/software/ccd2cue/

Atom feed https://savannah.gnu.org/news/atom.php?group=ccd2cue

Help and support mailing list
ccd2cue@gnu.org

Bug reporting mailing list
bug-ccd2cue@gnu.org

IRC channel
irc://irc.freenode.net/ccd2cue

Support Tracker
https://savannah.gnu.org/support/?group=ccd2cue

Bug Tracker
https://savannah.gnu.org/bugs/?group=ccd2cue

Development page
https://savannah.gnu.org/projects/ccd2cue/

2.2 News

This document contains a list of user-visible changes worth mentioning. The changes are split and ordered by version in reverse chronological order. If you want to receive notifications about new releases of this program or important issues related to it, subscribe to the mailing list, as described in Section 1.4 [Contact], page 4, or subscribe to the package atom feed `https://savannah.gnu.org/news/atom.php?group=ccd2cue`.

0.5

- Danish, German, Ukrainian and Vietnamese translations.
- Support for localized Unix manual pages. Feature suggested by Mario Blättermann (German translator).
- Respectable Unix manual pages.
- Localizable `--help` meta-variables. Bug reported by Mario Blättermann.
- Numerous grammatical and markup corrections to the user's manual. Patch submitted by Karl Berry.
- Make target `announcegnu` which automatically sends a signed announcement message to `info-gnu@gnu.org`, `ccd2cue@gnu.org` and `coordinator@translationproject.org` mailing lists when a release is ready. This target can only be made in VCS checkouts.
- Make target `fetchpo` which fetches from the Translation Project the latest PO files available . This target can only be made in VCS checkouts.
- Latest release news, manifesto, package's description and version in a single point of maintenance.
- Package meta-information, as release date and build system version, automatically generated at configuration time.
- Announcement message in user's manual and homepage.
- Make target `distdir` can be made when top-level documentation files are missing.

0.4

- The code repository moved from CVS to Git.
- Mailing lists were re-purposed. Now `bug-ccd2cue@gnu.org` is exclusively used for bug reporting and `ccd2cue@gnu.org` for user support.
- Fixed bad wording in `--help` text. Bug reported by Benno Schulenberg of the Translation Project.
- Fixed potential crash caused by calling `error` function with wrong arguments inside error handling routines. Bug first noticed by Rosa (GNU/)Linux maintainers.
- Fixed documentation distribution bug in which the `INSTALL` file didn't get redistributed. Bug reported by Darren S..
- Added `distgnu` Make target which aids the maintainer in uploading release tarballs to the GNU ftp site. It only can be made from VCS checkouts.
- Likewise, the `homepage` Make target now can only be made from VCS checkouts. Therefore the exclusive infrastructure for building the package's homepage doesn't get distributed.

- Fixed build system bug in which `configure` didn't detect the absence of `help2man` for maintainer builds.
- Improved `maintainer-clean` Make target effectiveness.

0.3

- The program ccd2cue has become a GNU package, and therefore is now dubbed GNU ccd2cue. Its code and documentation have been updated to reflect such remarkable occurrence.
- The target `homepage` generates the package homepage primarily from release documentation.
- Some interesting GNU philosophy essays were added to documentation.
- Various bugs in the program and documentation were fixed.

0.2

- Support for all features of the CUE sheet format, prominently CD-Text meta-data and track sub-code flags.
- Internationalization support using gettext.
- Compliance with the GNU Coding Standards and the GNU Maintaining Standards. See *GNU Coding Standards* and *GNU Maintaining Standards*. Noteworthy changes are the use of the GNU build system for the ease and consistency of build and installation (see `INSTALL` file), documentation in the Texinfo format, notably an user manual that can be outputted in numerous formats including, but not limited to, printable pdf, info, html and plain text.
- Traditional Unix man page automatically generated by help2man.
- Doxygen for in-depth source code documentation. That can be used to output pdf, html and numerous other formats documenting the inner workings of the code.
- GNU compliant command line parsing provided by Argp. It is no more necessary to provide arguments in the fixed order of command's synopsis; the arguments can be given in any order and long options are accepted.
- The command line options suffered the following changes to comply with GNU Coding Standards:

  `-o` Has companion long option `--output`.

  `-i` Has companion long option `--image`.

  `-h` Changed to `-?` with companion long option `--help`.

  `-v` Changed to `-V` with companion long option `--version`.

  The following command line options were added to cover new features:

  `-c` *`cdt-file`*
  `--cd-text=`*`cdt-file`*
  Writes CD-Text data to *`cdt-file`*.

`-a`
`--absolute-file-name`
: Uses absolute file name deduction.

`--usage`
: Gives a short usage message.

- It is no longer required to specify `-i` option in every invocation. It is enough to specify only one file name on the command invocation, as an option or non-option argument, since the remaining needed file names are now deduced automatically.

0.1
: Initial release;

2.3 To do

This chapter contains a list of ideas and features which would be nice to see implemented some day. If you are ready to start working on any of these TODO items, we appreciate your help; please write to `bug-ccd2cue@gnu.org` so we can be aware that the problem is being addressed, and talk with you how to do it best. It is best to consult the latest version of this file in the program source code repository. See Section 1.3 [Getting a copy], page 4 and Section 1.8 [Hacking], page 6 for instruction on how to access it.

Since the developers of this program are willing to assign the copyright of this package to FSF, please be prepared to sign legal papers to transfer the copyright on your work to the FSF too. For more details on this and some more practical details about getting involved, see Section 1.6 [Contributing], page 5.

As well as the issues listed here, there are bug reports, which are effectively to-do items too — that can use your help — at the bug tracker web interface. See Section 1.5 [Bug reporting], page 4 for more information.

CCD sheet format specification
: There is a specification of the CCD sheet format in the program manual. That specification is the result of a reverse engineering process and is incomplete; pretty much only the fields that affects the CUE sheet generation are known. It would be very nice to decipher the entirety of the format, and document it, so we can bring into light whether there is room for improvement in the conversion process, and to open new possibilities for more expressive formats like the TOC sheet.

TOC sheet format
: As the name implies the program GNU ccd2cue is designed to convert CCD sheets only to the CUE sheet format. This format has some well know limitations regarding the sub-channel data, which apparently can be mitigated by more expressive formats. There is one sheet format called “TOC” that is very common, well documented and accessible to free software in the GNU system. It seems that format was designed to handle the sub-channel data that is the major flaw of the CUE sheet format. In fact it seems like a super-set of the CUE sheet format. For that reason we would like to have a ccd2toc in complement of the GNU ccd2cue. But is that another project? If not, we have to change the name. ;-)

2.4 Authors

This chapter contains a list of people that made sufficiently large contributions to development that they can be regarded as truly authors of the program. This program would not be what it is today without the invaluable help of these people, to whom we would like to say:

THANK YOU VERY MUCH!

The names are listed in chronological order of contribution. Each person's contact information, usually email, and a summary of that person's contribution is put in per correspondent section.

For contributions that does not matter for copyright purposes, e.g., minor or non-source code contributions, see Section 2.5 [Thanks], page 15. If you would appreciate your own name listed here, **please contribute!** See Section 1.6 [Contributing], page 5.

Bruno Félix Rezende Ribeiro (oitofelix) `oitofelix@gnu.org`
He is the original and only author so far.

2.5 Thanks

This chapter contains a list of people that reported problems, suggested improvements or submitted relatively small portions of actual code. This program would not be what it is today without the invaluable help of these people, to whom we would like to say:

THANK YOU!

The names are listed in chronological order of contribution. Each person's contact information, usually email, and a summary of that person's contribution is put in per correspondent section.

For contributions that does matter for copyright purposes, e.g., major source code contributions, see Section 2.4 [Authors], page 14. If you would appreciate your own name listed here, **please contribute!** See Section 1.6 [Contributing], page 5.

Lucas Sköldqvist (frusen) `frusen@gungre.ch`

> He pointed out a syntactically incorrect expression in the manifesto, kindly tested pre-release versions and reported bugs.

2.6 Donors

This chapter contains a list of people that supported development by means of donation. This program would not be what it is today without the invaluable help of these people, to whom we would like to say:

THANK YOU VERY MUCH!

If you would appreciate your own name listed here, **please donate!** See Section 1.7 [Donating], page 5.

Name	**Email**	**BTC**
Lucas Sköldqvist (frusen)	frusen@gungre.ch	0.28833329

3 Operation

First, it is important to distinguish between the various files involved in the conversion operation accomplished by GNU ccd2cue. The whole process basically consists of a conversion of the CCD set of files into a CUE set of files, whose meaning, by suffix, is described below:

In the *CCD set* all files have the same basename and the respective suffixes are mandatory. It is composed of:

`.img` Audio/data raw image (hereafter called *image file*);

`.sub` Sub-channel data (hereafter called *sub-channel file*);

`.ccd` Layout description (hereafter called *CCD sheet file*);

In the *CUE set* all files need not have the same basename and the respective suffixes are just an optional default. It is composed of:

`.bin` The same image file as above with the traditional suffix;

`.cdt` CD-Text meta-data file (hereafter called *CD-Text file*);

`.cue` Layout description (hereafter called *CUE sheet file*);

The GNU ccd2cue program never touches any of the files in the CCD set; not even to rename the image file to use the traditional `.bin` suffix. However it will still work with other software because the image file is explicitly referenced inside the CUE sheet file. The idea is to take the least intrusive approach, e.g., to have both sets of files simultaneously without any interfering with each other. Although we are against the use and development of any proprietary program, including those which can handle CCD sheets[1], it can be useful to have the original CCD sheet for reference, in case the conversion process is improved in a subsequent release; or more expressive destination formats[2] become supported; or just for debugging.

The GNU ccd2cue program has just one non-trivial operation: the conversion. And this only relies on the information contained inside the CCD sheet file to generate the equivalent CUE sheet file. Thus, GNU ccd2cue does not enforce, and in fact does not even check, if you have the sub-channel file, or even more important, the actual image file. Therefore, although it is possible to generate seamlessly a CUE sheet file from a lone CCD sheet file, it is a must to have also, at least, the corresponding image file in order to burn the disc.

While the image file is not used by GNU ccd2cue but is referenced in the produced CUE sheet file, the sub-channel file is not used nor referenced at all in the CUE sheet file. The CUE sheet format was not designed to describe the information found in the sub-channel

[1] Actually I'm really sorry those programs came into existence in the first place. I wish I never had to write a program like GNU ccd2cue. I would then be contributing to society in a way that directly advances the social good, not just alleviating the wrong doing of others that immersed in their own egoism and blindness desire to subjugate their neighbors for their own benefit.

[2] The TOC sheet format is an example.

file and most optical disc authoring software seems to ignore it when guided by a CUE sheet file. Therefore, do not expect to obtain an exact copy, at the sub-channel level, using the CUE sheet file to guide your burning software. Nevertheless, for the vast majority of practical uses the extra sub-channel data contained in the sub-channel file can be fully ignored producing yet usable discs for the intended application. Notice however that that does not mean CUE sheet format is incapable of describing any sub-channel data; in fact it describes the most important ones. Hence all that data is available for GNU ccd2cue not inside the sub-channel file but rather in the CCD sheet file itself; the CD-Text meta-data is such a case — the CD-Text file is produced from decoded sub-channel information available inside the CCD sheet file.

4 Tutorial

The most ordinary use case is when you have a CCD set of files and just want to generate a CUE sheet file in order to burn or otherwise access the data inside the image file. Supposing your CCD sheet file is called free-as-in-freedom.ccd, all you need is the command:

```
ccd2cue -o free-as-in-freedom.cue free-as-in-freedom.ccd
```

Remember that if your CCD sheet file name has spaces or unusual characters, like '$', you are better quoting the whole name with ''. Naturally the CUE sheet file don't need to have the same base name as the CCD sheet file. You can give an adequate CUE sheet output file name, at your discretion, to the `-o` (`--output`) option. If you omit the `-o` option entirely as in

```
ccd2cue free-as-in-freedom.ccd
```

the result is that the yielding CUE sheet is written to the standard output, giving you the chance to pipe it out to another command or to use the shell to redirect it to a file, as you would do without it using the `-o` option. The same thing is accomplished by passing `-` as an argument to option `-o`.

Can you guess what happens if we omit the CCD sheet input file specification as well? If you answered that GNU ccd2cue will read the CCD sheet input from standard input and write the correspondent CUE sheet output to standard output, unfortunately you are wrong. It would be nice if it behaved that way, but there is an inherent design drawback: the CUE sheet needs to reference an image file and possibly a CD-Text file; in that way, how could it have any clue about those file names? For that reason it is always necessary to supply at least one file name in an option or non-option argument, so GNU ccd2cue can deduce the remaining file names needed. The deduction algorithm is very simple: get the file name supplied, in the following preference order:

1. CCD sheet input file name (non-option argument);
2. CUE sheet output file name (`-o`, `--output`);
3. Image file name (`-i`, `--image`);
4. CD-Text file name (`-c`, `--cd-text`);

conserve only its base name and concatenate the canonical suffix. Therefore, if you want to read the CCD sheet input from standard input and write the CCD sheet output to standard output, it is enough to just make explicit the name of a image name, possibly with a nonstandard suffix, regardless of the actual existence of the image file, as in

```
ccd2cue -i free-as-in-free-speech.img
```

where `free-as-in-free-speech.img` is the image file name. On the other hand, you can use the `-i` (`--image`) in conjunction with the CCD sheet file name and/or CUE sheet file name just to override the image file name deduction. As expected, the argument `-` in place of the CCD sheet file name, will also make GNU ccd2cue read its CCD sheet input from standard input.

If, for some weird reason, you want to move the CUE sheet file around while keeping the image file in its original place and yet be able to normally use this pair as if they were in the same directory, you can use the option `-a` (`--absolute-file-name`). That option will make the file name deduction algorithm explained above retain the absolute directory name

for each file; however, as a consequence of this approach, it will only work for file names not directly supplied, but deduced by the program.

When GNU ccd2cue detects CD-Text data information inside the CCD sheet, it outputs a CD-Text file, whose name is determined by the same file name deduction rules just described, unless overridden by the option `-c` (`--cd-text`). It is not possible to output the binary CD-Text data to standard output, however.

5 Invocation

The command `ccd2cue` is used to convert CCD sheet files in CUE sheet files. It is invoked as follow:

```
ccd2cue [OPTION...] [ccd-file]
```

The input file, referred as *`ccd-file`*, must exist. If *`ccd-file`* is '`-`', or omitted, standard input is used.

Output options:

`-c`
`--cd-text=`*`cdt-file`*
: Write CD-Text data to *`cdt-file`*.

`-o`
`--output=`*`cue-file`*
: Write output to *`cue-file`*.

While the main output file *`cue-file`* is always generated, the *`cdt-file`* is created only when there is CD-Text info. If *`cue-file`* is '`-`', or `--output` is omitted, standard output is used.

Conversion options:

`-a`
`--absolute-file-name`
: Use absolute file name deduction.

`-i`
`--image=`*`img-file`*
: Reference *`img-file`* as the image file.

The *`img-file`* is a reference to a data file required only at burning time and thus its existence is not enforced at the conversion stage.

Help options:

`-?`
`--help` Give a help list.

`--usage` Give a short usage message.

`-V`
`--version`
: Print program version.

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

6 Burning

READ THIS!

If you have burned a CD from a CUE sheet produced by this program and all audio tracks became only senseless static noise, you may need to tell your burning software to swap the byte order of all samples sent to the CD-recorder. This can be accomplished with (for example) the `--swap` option when using the `cdrdao` program. Experience has shown that at least for mixed-mode discs it is necessary to use that option when burning, otherwise you will almost certainly waste a CD.

Supposing you want to burn a CD using the cdrdao program and a CUE sheet file named `gnu.cue`, and wisely want to ensure the correct behavior of your burnt disc, use the command:

```
cdrdao write --swap --speed 1 --eject gnu.cue
```

That way `cdrdao` will swap the byte order of audio samples, cautiously burning in the smallest possible speed and will eject your CD when it is done.

Appendix A Compact Disc fields

This appendix describes some Compact Disc information that are common concepts to both sheet formats. Each following manual section corresponds to a Compact Disc field or characteristic. For each of them is explained its meaning, the values it can assume and is given an example where appropriate. Cross references to both CCD and CUE sheets applicable elements follow each description.

A.1 CATALOG

Description
: MCN (Media Catalog Number);

Value
: *UPC (Universal Product Code)*[1]/*EAN (European Article Number)*[2] encoded; exactly 13 decimal digits;

Example
: `7954234679231`

CCD sheet
: See Section B.2.5 [CATALOG (Disc CCD Section)], page 31.

CUE sheet
: See Section C.1 [CATALOG (CUE Command)], page 41.

This field is usually used on CD-ROM's mastered for commercial production.

[1] See https://en.wikipedia.org/wiki/Universal_Product_Code.

[2] See https://en.wikipedia.org/wiki/European_Article_Number.

A.2 MODE

Description

Track data type;

Value The following modes are defined:

Value	Description
AUDIO	Audio/Music (2352 — 588 samples)
CDG	Karaoke CD+G (2448)
MODE1/2048	CD-ROM Mode 1 Data (cooked)
MODE1/2352	CD-ROM Mode 1 Data (raw)
MODE2/2048	CD-ROM XA Mode 2 Data (form 1) *
MODE2/2324	CD-ROM XA Mode 2 Data (form 2) *
MODE2/2336	CD-ROM XA Mode 2 Data (form mix)
MODE2/2352	CD-ROM XA Mode 2 Data (raw)
CDI/2336	CDI Mode 2 Data
CDI/2352	CDI Mode 2 Data

The modes marked with '*' are not defined in the original CUE sheet format specification.

CCD sheet

See Section B.6.1 [MODE (TRACK CCD Section)], page 38

CUE sheet

See Section C.7 [TRACK (CUE Command)], page 48.

A.3 FLAGS

Description

Track special sub-code flags;

Value The following flags (possibly more than one):

Value	Description
DCP	**D**igital **C**opy **P**ermitted
4CH	**4 CH**annel audio
PRE	**PRE**-emphasis enabled (audio tracks only)
SCMS	**S**erial **C**opy **M**anagement **S**ystem

For non-audio tracks the special flag DATA is used implicitly.

CCD sheet

See Section B.6.4 [FLAGS (TRACK CCD Section)], page 39.

CUE sheet

See Section C.8 [FLAGS (CUE Command)], page 49.

A.4 ISRC

Description

ISRC (International Standard Recording Code)[3];

Value 12 characters: the first five characters are upper case alphanumeric; the last seven are numeric only. The code has the format: *CCOOOYYSSSSS*, where:

`C`	Country code;
`O`	Owner code;
`Y`	Year;
`S`	Serial number;

Example `USRMS8371421`

CCD sheet

See Section B.6.2 [ISRC (TRACK CCD Section)], page 38.

CUE sheet

See Section C.9 [ISRC (CUE Command)], page 50.

This field is usually used on CD-ROM's mastered for commercial production.

[3] See https://en.wikipedia.org/wiki/ISRC.

Appendix B CCD sheet format

Warning: This appendix describes the CCD sheet format. This description is the result of observation, deduction and tests. Although there is quite some uncertainty about the semantics of a number of elements, it is expected that every applicable entry, or otherwise relevant element — for the production of the corresponding CUE sheet — is well understood. Due to its proprietary nature, no authoritative, free and public available reference of the CCD sheet format do exist. Whereas the information contained herein is thought to be accurate, there is no guarantee of its correctness.

The CCD sheet format is an instance of the *INI format*[1]; a simple text file with a basic structure composed of *sections* and *properties*. The following syntactic rules apply to CCD sheets:

1. Every section has a name in square brackets that appears on a line by itself (as in `[CDText]`).
2. Every property has a name and a value (in the form *`name`*`=`*`value`* — there is no space around '`=`') and belongs exclusively to one section: the nearest upward section.
3. There is no explicit end-of-section delimiter (sections end at the next section declaration or end-of-file) and sections cannot be nested.
4. A ' ' (space) can occur in a *indexed* section (resp. property) name followed by a number (like in `[Session 3]` (resp. `INDEX 0=17524`)).
5. Each non-indexed section (resp. property) come out only once per file (resp. section).
6. Indexes of sections and properties starts at the minimum value allowed and increment sequentially.

Each following manual section corresponds to a CCD sheet *section*, and each subsection within to a valid *property* of its parent section. The sequence in which sections should be declared in the CCD sheet is the same given below. For each section and property is explained its meaning, and for properties is made explicit its value type. A '`(?)`' indicates the correspondent property value is not more than believed to be right or is a guess against which no counter example do exist so far. In the end of each section description is presented a code example of that section using each available property just described. Indexed sections and properties are those whose syntax references *`index`*. Unless otherwise noted, each section and property are mandatory. The correlation with CUE sheet format is expressed by means of cross references to the relevant CUE commands.

[1] See https://en.wikipedia.org/wiki/Ini_file.

B.1 `CloneCD`

Syntax `[CloneCD]`

Description
Properties of the disc authoring software;

Example

```
[CloneCD]
Version=3
```

B.1.1 `Version`

Syntax `Version=`*`value`*

Description
The major version of CloneCD which made the CCD sheet;

Value Positive integer;

B.2 Disc

Syntax `[Disc]`

Description
Properties that apply to the entire disc;

Example

```
[Disc]
TocEntries=18
Sessions=1
DataTracksScrambled=0
CDTextLength=810
CATALOG=5479154328763
```

B.2.1 TocEntries

Syntax `TocEntries=`*value*

Description
Number of `Entry` sections;

Value Strictly positive integer;

B.2.2 Sessions

Syntax `Sessions=`*value*

Description
Number of `Session` sections;

Value Strictly positive integer;

B.2.3 DataTracksScrambled

Syntax `DataTracksScrambled=`*value*

Description
Unknown meaning;

Value Non-negative integer (?);

B.2.4 `CDTextLength`

Syntax `CDTextLength=value`

Description

The size in bytes of the corresponding CD-Text file discarding one byte for the terminating null character; `0` if there is no CD-Text meta-data;

See Section C.2 [CDTEXTFILE (CUE Command)], page 42, for more information on the CD-Text file.

Value Given by the formula:

```
(16 + 2) * [CDText].Entries
```

Where:

- `16` → CD-Text entry length (in bytes);
- `2` → CRC code length (in bytes);
- `[CDText].Entries` → Value of `Entries` property of `CDText` section;

CUE sheet

See Section C.2 [CDTEXTFILE (CUE Command)], page 42.

B.2.5 `CATALOG`

Syntax `CATALOG=value`

Description

MCN (Media Catalog Number);

Value UPC/EAN encoded; exactly 13 decimal digits; see Section A.1 [CATALOG (Compact Disc fields)], page 24.

CUE sheet

See Section C.1 [CATALOG (CUE Command)], page 41.

B.3 `CDText`

Syntax `[CDText]`

Description

CD-Text meta-data; this section is entirely omitted if there is no CD-Text meta-data.

Example

```
[CDText]
Entries=3
Entry 0=80 00 00 00 54 72 69 6f 20 41 63 6f 75 73 74 69
Entry 1=81 00 16 00 54 72 69 6f 20 41 63 6f 75 73 74 69
Entry 2=8f 00 2a 00 00 01 0f 00 16 14 00 00 00 00 00 00
```

CUE sheet

See Section C.2 [CDTEXTFILE (CUE Command)], page 42.
See Section C.3 [TITLE (CUE Command)], page 43.
See Section C.4 [PERFORMER (CUE Command)], page 44.
See Section C.5 [SONGWRITER (CUE Command)], page 45.

B.3.1 `Entries`

Syntax `Entries=`*`value`*

Description

Number of `Entry` properties inside `CDText` section;

Value Strictly positive integer;

B.3.2 `Entry`

Syntax `Entry` *`index=value`*

Description

CD-Text entry;

Value The nth index is exactly the nth CD-Text file entry, omitting the 2 byte CRC code and additionally — in the case of the last entry — the final null character. It is encoded as a space-separated sequence of 16 hexadecimal numbers of two digits *without* '`0x`' base indicator. Each pair of digits represents one byte.

Index Non-negative integer;

B.4 Session

Syntax `[Session index]`

Description
Unknown meaning;

Index Strictly positive integer;

Example

```
[Session 1]
PreGapMode=0
PreGapSubC=0
```

B.4.1 PreGapMode

Syntax `PreGapMode=value`

Description
Unknown meaning;

Value Non-negative integer (?);

B.4.2 PreGapSubC

Syntax `PreGapSubC=value`

Description
Unknown meaning;

Value Non-negative integer (?);

B.5 `Entry`

Syntax `[Entry index]`

Description
Table of Contents entry;

Index Non-negative integer;

Example

```
[Entry 0]
Session=1
Point=0xa0
ADR=0x01
Control=0x00
TrackNo=0
AMin=0
ASec=0
AFrame=0
ALBA=-150
Zero=0
PMin=1
PSec=0
PFrame=0
PLBA=4350
```

Every property whose value is an integer in hexadecimal base uses '`0x`' notation.

B.5.1 `Session`

Syntax `Session=value`

Description
Unknown meaning;

Value Strictly positive integer;

B.5.2 `Point`

Syntax `Point=value`

Description
Unknown meaning;

Value Hexadecimal non-negative integer;

B.5.3 ADR

Syntax `ADR=value`

Description
Unknown meaning;

Value Hexadecimal non-negative integer;

B.5.4 Control

Syntax `Control=value`

Description
Unknown meaning;

Value Hexadecimal non-negative integer;

B.5.5 TrackNo

Syntax `TrackNo=value`

Description
Unknown meaning;

Value Non-negative integer;

B.5.6 AMin

Syntax `AMin=value`

Description
Unknown meaning;

Value Non-negative integer;

B.5.7 ASec

Syntax `ASec=value`

Description
Unknown meaning;

Value Non-negative integer;

B.5.8 `AFrame`

Syntax `AFrame=`*`value`*

Description
Unknown meaning;

Value Non-negative integer;

B.5.9 `ALBA`

Syntax `ALBA=`*`value`*

Description
Unknown meaning;

Value Negative integer;

B.5.10 `Zero`

Syntax `Zero=`*`value`*

Description
Unknown meaning;

Value Non-negative integer;

B.5.11 `PMin`

Syntax `PMin=`*`value`*

Description
Unknown meaning;

Value Non-negative integer;

B.5.12 `PSec`

Syntax `PSec=`*`value`*

Description
Unknown meaning;

Value Non-negative integer;

B.5.13 `PFrame`

Syntax `PFrame=value`

Description
Unknown meaning;

Value Non-negative integer;

B.5.14 `PLBA`

Syntax `PLBA=value`

Description
Unknown meaning;

Value Non-negative integer;

B.6 TRACK

Syntax `[TRACK index]`

Description
Track layout and meta-data;

Index Strictly positive integer;

Example

```
[TRACK 1]
MODE=0
ISRC=DKKH50800101
INDEX 1=0
FLAGS=DCP 4CH PRE SCMS
```

CUE sheet
See Section C.7 [TRACK (CUE Command)], page 48.

B.6.1 MODE

Syntax `MODE=value`

Description
Track mode; see Section A.2 [MODE (Compact Disc fields)], page 25.

Value Non-negative integer;

Value	Mode
0	AUDIO
1	MODE1/2352 (?)
2	MODE2/2352

CUE sheet
See Section C.7 [TRACK (CUE Command)], page 48.

B.6.2 ISRC

Syntax `ISRC=value`

Description
ISRC (International Standard Recording Code); see Section A.4 [ISRC (Compact Disc fields)], page 27.

Value 12 characters: the first five characters are alphanumeric; the last seven are numeric only.

CUE sheet
See Section C.9 [ISRC (CUE Command)], page 50.

B.6.3 `INDEX`

Syntax `INDEX` *`index=value`*

Description
Index within a track;

Index Non-negative integer;

Value Non-negative integer indicating the number of frames (1/75th of a second).

CUE sheet
See Section C.11 [INDEX (CUE Command)], page 52.

B.6.4 `FLAGS`

Syntax `FLAGS=`*`value`*

Description
Special sub-code flags for this track;

Value A space-separated string of the flags described in Section A.3 [FLAGS (Compact Disc fields)], page 26;

CUE sheet
See Section C.8 [FLAGS (CUE Command)], page 49.

Appendix C CUE sheet format

The CUE sheet format is a simple text file composed solely of CUE commands, its respective arguments, blank spaces and lines. The following syntactic rules apply to CCD sheets:

1. Each command take one, and only one, entire line.
2. Commands are case insensitive.
3. Commands can be indented at will.
4. Blank lines are ignored.

There are three contexts in a CUE sheet file: *None*, *File* and *Track*. The first of them is not introduced by any command, and the last two are introduced by `FILE` and `TRACK` commands, respectively. The only commands that pertain to more than one context, namely None and Track, are the CD-Text related commands `TITLE`, `PERFORMER` and `SONGWRITER`. The only commands that can appear multiple times within its contexts are `FILE`, `TRACK`, `INDEX` and `REM`.

The valid positions of a command are determined by the following rules:

1. A command can only appear after the introduction of the context it pertains to.
2. Within that context the position of a command is determined by its specific positioning rules.

Each following manual section corresponds to a CUE command. For each of them is explained its meaning, context structure and positioning rules. At the end of each section is an example of use and a reference to the correlated CCD sheet format element, if there is any.

The meaning of the context tables is:

Parents — The contexts to which this command pertains; '`None`' here names the context introduced by no command.

Multiple — Whether multiple instances of this command within a single instance of its context is allows; value is '`Yes`' or '`No`'.

Children — The commands that pertains to the context introduced by this command; '`None`' if there is not one.

The meaning of the positioning tables is:

After — This command must appear after the commands listed here, if they are present. The expression '`None`' means nothing.

Before — This command must appear before the commands listed here, if they are present. The expression '`None`' means nothing.

C.1 CATALOG

`CATALOG` *mcn* [CUE Command]

Define MCN (Media Catalog Number) of the disc as *mcn*. The argument *mcn* is a number composed of 13 decimal digits in UPC/EAN encoding as described in Section A.1 [CATALOG (Compact Disc fields)], page 24.

Context

Parents None

Multiple No

Children None

Position

After None

Before

`CDTEXTFILE`
`TITLE`
`PERFORMER`
`SONGWRITER`
`FILE`

Example

```
CATALOG 3203040601052
```

CCD sheet

See Section B.2.5 [CATALOG (Disc CCD Section)], page 31.

C.2 CDTEXTFILE

`CDTEXTFILE` *file-name* [CUE Command]

Specify file *file-name* as the one containing the CD-Text meta-data of the disc. If *file-name* contains any spaces it must be enclosed in quotation marks.

The *CD Text Encoding and Decoding Guide*[1] documents the CD-Text file format.

Context

Parents None

Multiple No

Children None

Position

After

`CATALOG`

Before

`TITLE`
`PERFORMER`
`SONGWRITER`
`FILE`

Example

```
CDTEXTFILE músicas-com-nomes-de-garotas.cdt
CDTEXTFILE "EngHaw - Longe Demais das Capitais (1986).cdt"
```

CCD sheet

See Section B.3 [CDText (CCD Section)], page 32.

[1] See https://gnu.org/software/libcdio/cd-text-format.html.

C.3 TITLE

`TITLE` *title-string* [CUE Command]

If this command is inside a `TRACK` command context, make *title-string* the title of the correspondent track; otherwise make it the title of the entire disc. The value *title-string* should not contain more than 80 characters. If *title-string* contains any space, it must be enclosed in quotation marks.

This command is meant for CD-Text enhanced discs. The same data can be retrieved from the CD-Text file using `CDTEXTFILE` command.

Context

Parents

None

`TRACK`

Multiple No

Children None

Position

After

`CATALOG`

`CDTEXTFILE`

Before

`FILE`

Position (`TRACK`)

After None

Before

`INDEX`

Example

```
TITLE "Por Quem os Sinos Dobram"
TITLE S.O.S.
```

CCD sheet

The CCD sheet format support this feature only by means of a general mechanism designed to specify indiscriminately any CD-Text encoded entry. See Section B.3 [CDText (CCD Section)], page 32.

C.4 PERFORMER

`PERFORMER` *performer-string* [CUE Command]

If this command is inside a `TRACK` command context, write *performer-string* as the name of the performer of the correspondent track; otherwise write it as the name of the performer of the entire disc. The value *title-string* should not contain more than 80 characters. If *title-string* contains any space, it must be enclosed in quotation marks.

This command is meant for CD-Text enhanced discs. The same data can be retrieved from the CD-Text file using `CDTEXTFILE` command.

Context

Parents

None

`TRACK`

Multiple No

Children None

Position

After

`CATALOG`

`CDTEXTFILE`

Before

`FILE`

Position (`TRACK`)

After None

Before

`INDEX`

Example

```
PERFORMER "Detonator e as Musas do Metal"
PERFORMER Metrô
```

CCD sheet

The CCD sheet format support this feature only by means of a general mechanism designed to specify indiscriminately any CD-Text encoded entry. See Section B.3 [CDText (CCD Section)], page 32.

C.5 SONGWRITER

`SONGWRITER` *song-writer-string* [CUE Command]

If this command is inside a `TRACK` command context, write *song-writer-string* as the name of the song writer of the correspondent track; otherwise write it as the name of the song writer of the entire disc. The value *song-writer-string* should not contain more than 80 characters. If *song-writer-string* contains any space, it must be enclosed in quotation marks.

This command is meant for CD-Text enhanced discs. The same data can be retrieved from the CD-Text file using `CDTEXTFILE` command.

Context

Parents

None

`TRACK`

Multiple No

Children None

Position

After

`CATALOG`

`CDTEXTFILE`

Before

`FILE`

Position (`TRACK`)

After None

Before

`INDEX`

Example

```
SONGWRITER "Stefani Joanne Angelina Germanotta"
SONGWRITER Getz
```

CCD sheet

The CCD sheet format support this feature only by means of a general mechanism designed to specify indiscriminately any CD-Text encoded entry. See Section B.3 [CDText (CCD Section)], page 32.

C.6 FILE

`FILE` *file-name file-type* [CUE Command]

Declare file *file-name* of type *file-type* as the one whose data will be used in the following `TRACK` command contexts by `INDEX` commands. If *file-name* contains any space, it must be enclosed in quotation marks.

There are a total of 5 types *file-type* can take: two types for raw data and three for audio. The raw data types are *BINARY* and *MOTOROLA*, and they differ only in endianness — little and big endian respectively. The audio types are *AIFF*, *WAVE* and *MP3* and the correspondent files must be setup to 44.1 KHz, 16 bits and stereo. The following table summarizes it:

`BINARY`	Intel binary raw data *Little-endian*[2] — least significant byte first).
`MOTOROLA`	Motorola binary raw data *Big-endian*[3] — most significant byte first).
`AIFF`	Audio data in *AIFF (Audio Interchange File Format)*[4].
`WAVE`	Audio data in *WAVE (Waveform Audio File Format)*[5].
`MP3`	Audio data in *MP3 (MPEG-1 or MPEG-2 Audio Layer III)*[6].

Context

Parents None

Multiple Yes

Children

`TRACK`

Position

After

`CATALOG`

`CDTEXTFILE`

Before None

Example

```
FILE guix-1.0.iso BINARY
FILE "The Free Software Song.wav" WAVE
```

[2] See https://en.wikipedia.org/wiki/Little_endian#Little-endian.

[3] See https://en.wikipedia.org/wiki/Big_endian#Big-endian.

[4] See https://en.wikipedia.org/wiki/Aiff.

[5] See https://en.wikipedia.org/wiki/Waveform_Audio_File_Format.

[6] See https://en.wikipedia.org/wiki/Mp3.

CCD sheet

Since the CCD sheet file is not meant to be composed from scratch by an user or authoring software, but rather created to mimic an existing disc layout, it is assumed that the source of data are the CCD image and sub-channel files, which have predefined formats and names. Thus there is no related explicit element in the CCD sheet format.

Among the CCD set files, is presumed that the source of audio/data is a file with the same base name of the CCD sheet file, but with a different extension, to indicate its specific purpose, namely `.img`. This file is always assumed as having type `BINARY`.

C.7 TRACK

`TRACK` *number data-type* [CUE Command]

Define the *number*th track, whose type is *data-type*. The value *number* is a strictly positive integer and must be one greater than the last one supplied to a previous `TRACK` command, notwithstanding *number* can be greater than `1` in the first occurrence of `TRACK` command; however, it cannot exceed `99` in any case. Usually *number* is padded with a `0` on the left when smaller than `10`, in order to keep track numbers two digit wide uniformly throughout the CUE sheet. The *data-type* argument is one of those described at Section A.2 [MODE (Compact Disc fields)], page 25.

This command should appear at least once in the CUE sheet file.

Context

Parents

`FILE`

Multiple Yes

Children

`FLAGS`
`ISRC`
`TITLE`
`PERFORMER`
`SONGWRITER`
`PREGAP`
`INDEX`
`POSTGAP`

Position

After None

Before None

Example

```
TRACK 1 MODE2/2352
TRACK 37 AUDIO
```

CCD sheet

See Section B.6 [TRACK (CCD Section)], page 38.

C.8 FLAGS

`FLAGS` *flag_1 flag_2 . . . flag_n* [CUE Command]

Set special sub-code flags `flag_1`...`flag_n` for the current track. Flags are separated by space. For the allowed flags and its respective meanings, see Section A.3 [FLAGS (Compact Disc fields)], page 26.

Context

Parents

`TRACK`

Multiple No

Children None

Position

After None

Before

`INDEX`

Example

```
FLAGS DCP
FLAGS 4CH PRE
```

CCD sheet

See Section B.6.4 [FLAGS (TRACK CCD Section)], page 39.

C.9 ISRC

`ISRC` *code* [CUE Command]

Specify the ISRC (International Standard Recording Code) to the current track. The value *code* is described at Section A.4 [ISRC (Compact Disc fields)], page 27.

This command only apply to audio tracks.

Context

Parents

`TRACK`

Multiple No

Children None

Position

After None

Before

`INDEX`

Example

```
ISRC DKKH50800101
```

CCD sheet

See Section B.6.2 [ISRC (TRACK CCD Section)], page 38.

C.10 PREGAP

`PREGAP` *mm:ss:ff* [CUE Command]

Set the length of the current track pre-gap to *mm* minutes plus *ss* seconds plus *ff* frames.

The values *mm*, *ss* and *ff* must be non-negative integers.

A frame is the 75th part of a second.

No data is consumed from the file specified in the current `FILE` command context.

Context

Parents

`TRACK`

Multiple No

Children None

Position

After None

Before

`INDEX`

Example

```
PREGAP 00:02:00
```

CCD sheet

No CCD sheet element analogous or directly related to this command is known.

C.11 INDEX

`INDEX` *number mm:ss:ff* [CUE Command]

If *number* is `0`, then consider the time specified by the next `INDEX` command as the track pre-gap length present in the file declared by the current `FILE` command context.

If *number* is `1`, then define the starting time of the index of this track as *mm* minutes plus *ss* seconds plus *ff* frames. This index specify the starting time of track data and is the only stored in the table-of-contents of the disc.

If *number* is `2` or more, then define the starting time of the $(number - 1)$th sub-index within this track as *mm* minutes plus *ss* seconds plus *ff* frames.

If this is the first `INDEX` command of the current `TRACK` command context, then it must start at `00:00:00` and *number* must be either `0` or `1`.

If this is not the first `INDEX` command of the current `TRACK` command context, then *number* must be one greater than that of the last `TRACK` command.

The value *number* cannot be greater than `99` and is usually padded with a `0` on the left when smaller than `10`, in order to keep index and sub-index numbers two digit wide uniformly throughout the CUE sheet.

The time `mm:ss:ff` is an offset relative to the beginning of the file specified by the current `FILE` command context.

The values *mm*, *ss* and *ff* must be non-negative integers.

There are 75 frames per second.

Context

Parents

`TRACK`

Multiple Yes

Children None

Position

After

`FLAGS`

`ISRC`

`TITLE`

`PERFORMER`

`SONGWRITER`

`PREGAP`

Before

`POSTGAP`

Example

```
INDEX 00 00:00:00
INDEX 01 00:02:00
INDEX 02 03:23:54
```

CCD sheet

See Section B.6.3 [INDEX (TRACK CCD Section)], page 39.

C.12 POSTGAP

`POSTGAP` *mm:ss:ff* [CUE Command]

Set the length of the current track post-gap to *mm* minutes plus *ss* seconds plus *ff* frames.

The values *mm*, *ss* and *ff* must be non-negative integers.

A frame is the 75th part of a second.

No data is consumed from the file specified in the current `FILE` command context.

Context

Parents

`TRACK`

Multiple No

Children None

Position

After

`INDEX`

Before None

Example

```
POSTGAP 00:02:00
```

CCD sheet

No CCD sheet element analogous or directly related to this command is known.

C.13 `REM`

`REM` *comment* [CUE Command]

Ignore *comment*.

The value *comment* can be any string of arbitrary length, but it ends at end-of-line.

Its practical use is to comment the CUE sheet for the sake of human readers.

There is no way to make multi-line comments, but one can simply start consecutive lines with this command.

Context

Parents None

Multiple Yes

Children None

Position

After None

Before None

Example

```
REM There is no system but GNU
REM and Linux-Libre is just one of its kernels.
```

CCD sheet

No CCD sheet element analogous or directly related to this command is known.

Appendix D GNU Free Documentation License

Version 1.3, 3 November 2008

FSF (Free Software Foundation) (http://fsf.org/)
51 Franklin St., Floor 5
Boston, MA 02110-1335
USA

This document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of

mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other

implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your

option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3 or
any later version published by the Free Software Foundation; with
no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled
``GNU Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix E The GNU Manifesto

The GNU Manifesto was written by Richard Stallman at the beginning of the GNU Project, to ask for participation and support. For the first few years, it was updated in minor ways to account for developments, but now it seems best to leave it unchanged as most people have seen it. Since that time, we have learned about certain common misunderstandings that different wording could help avoid. Footnotes added since 1993 help clarify these points. For up-to-date information about the available GNU software, please see the information available on our web server, in particular our list of software. For how to contribute, see http://gnu.org/help.

"The GNU Manifesto" was originally published in *Dr. Dobb's Journal,* vol. 10, n. 3 (March 1985).

This document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

What's GNU? Gnu's Not Unix!

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system which I am writing so that I can give it away free to everyone who can use it.[1] Several other volunteers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have an Emacs text editor with Lisp for writing editor commands, a source level debugger, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. A new portable optimizing C compiler has compiled itself and may be released this year. An initial kernel exists but many more features are needed to emulate Unix. When the kernel and compiler are finished, it will be possible to distribute a GNU system suitable for program development. We will use TEX as our text formatter, but an nroff is being worked on. We will use the free, portable X window system as well. After this we will add a portable Common Lisp, an Empire game, a spreadsheet, and hundreds of other things, plus online documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical to Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer file names, file version numbers, a crashproof file system, file name completion perhaps, terminal-independent display support, and perhaps

[1] The wording here was careless. The intention was that nobody would have to pay for *permission* to use the GNU system. But the words don't make this clear, and people often interpret them as saying that copies of GNU should always be distributed at little or no charge. That was never the intent; later on, the manifesto mentions the possibility of companies providing the service of distribution for a profit. Subsequently I have learned to distinguish carefully between "free" in the sense of freedom and "free" in the sense of price. Free software is software that users have the freedom to distribute and change. Some users may obtain copies at no charge, while others pay to obtain copies—and if the funds help support improving the software, so much the better. The important thing is that everyone who has a copy has the freedom to cooperate with others in using it.

eventually a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen. Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaosnet, and Internet protocols for communication.

GNU is aimed initially at machines in the 68000/16000 class with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on smaller machines will be left to someone who wants to use it on them.

To avoid horrible confusion, please pronounce the *g* in the word "GNU" when it is the name of this project.

Why I Must Write GNU

I consider that the Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away.[2]

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. And a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to Help

I have found many other programmers who are excited about GNU and want to help. Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more

[2] The expression "give away" is another indication that I had not yet clearly separated the issue of price from that of freedom. We now recommend avoiding this expression when talking about free software. See *Words to Avoid (or Use with Care) Because They Are Loaded or Confusing* for more explanation.

important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony which is impossible if we use software that is not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.[3]

One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machines should be complete, ready to use systems, approved for use in a residential area, and not in need of sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work for GNU. For most projects, such part-time distributed work would be very hard to coordinate; the independently written parts would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, then these utilities will work right when put together. Even allowing for Murphy to create a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part time. The salary won't be high by programmers' standards, but I'm looking for people for whom building community spirit is as important as making money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air.[4]

This means much more than just saving everyone the price of a Unix license. It means that much wasteful duplication of system programming effort will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available

[3] Nowadays, for software tasks to work on, see the High Priority Projects list, at http://fsf.org/campaigns/priority-projects/, and the GNU Help Wanted list, the general task list for GNU software packages, at http://savannah.gnu.org/people/?type_id=1. For other ways to help, see http://gnu.org/help/help.html.

[4] This is another place I failed to distinguish carefully between the two different meanings of "free." The statement as it stands is not false—you can get copies of GNU software at no charge, from your friends or over the net. But it does suggest the wrong idea.

programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company which owns the sources and is in sole position to make changes.

Schools will be able to provide a much more educational environment by encouraging all students to study and improve the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do with it will be lifted.

Arrangements to make people pay for using a program, including licensing of copies, always incur a tremendous cost to society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay for. And only a police state can force everyone to obey them. Consider a space station where air must be manufactured at great cost: charging each breather per liter of air may be fair, but wearing the metered gas mask all day and all night is intolerable even if everyone can afford to pay the air bill. And the TV cameras everywhere to see if you ever take the mask off are outrageous. It's better to support the air plant with a head tax and chuck the masks.

Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free.

Some Easily Rebutted Objections to GNU's Goals

"Nobody will use it if it is free, because that means they can't rely on any support."

"You have to charge for the program to pay for providing the support."

If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable.[5]

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost.

If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem; you are not at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible for there to be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them.

Meanwhile, the users who know nothing about computers need handholding: doing things for them which they could easily do themselves but don't know how.

Such services could be provided by companies that sell just handholding and repair service. If it is true that users would rather spend money and get a product with service, they will also be willing to buy the service having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

[5] Several such companies now exist.

"You cannot reach many people without advertising, and you must charge for the program to support that."
"It's no use advertising a program people can get free."

There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it.

On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really necessary to spread GNU. Why is it that free market advocates don't want to let the free market decide this?[6]

"My company needs a proprietary operating system to get a competitive edge."

GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems.

I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each.[7]

"Don't programmers deserve a reward for their creativity?"

If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only in so far as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

"Shouldn't a programmer be able to ask for a reward for his creativity?"

There is nothing wrong with wanting pay for work, or seeking to maximize one's income, as long as one does not use means that are destructive. But the means customary in the field of software today are based on destruction.

Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount and the ways that the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction.

The reason a good citizen does not use such destructive means to become wealthier is that, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics; or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one to do so.

---

[6] Although it is a charity rather than a company, the Free Software Foundation for 10 years raised most of its funds from its distribution service. You can order things from the FSF to support its work.

[7] A group of computer companies pooled funds around 1991 to support maintenance of the GNU C Compiler.

Specifically, the desire to be rewarded for one's creativity does not justify depriving the world in general of all or part of that creativity.

"Won't programmers starve?"

I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else.

But that is the wrong answer because it accepts the questioner's implicit assumption: that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing.

The real reason programmers will not starve is that it will still be possible for them to get paid for programming; just not paid as much as now.

Restricting copying is not the only basis for business in software. It is the most common basis[8] because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other bases of organization which are now used less often. There are always numerous ways to organize any kind of business.

Probably programming will not be as lucrative on the new basis as it is now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

"Don't people have a right to control how their creativity is used?"

"Control over the use of one's ideas" really constitutes control over other people's lives; and it is usually used to make their lives more difficult.

People who have studied the issue of intellectual property rights[9] carefully (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes.

For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for whom the cost and effort of a license agreement are small compared with setting up production,

[8] I think I was mistaken in saying that proprietary software was the most common basis for making money in software. It seems that actually the most common business model was and is development of custom software. That does not offer the possibility of collecting rents, so the business has to keep doing real work in order to keep getting income. The custom software business would continue to exist, more or less unchanged, in a free software world. Therefore, I no longer expect that most paid programmers would earn less in a free software world.

[9] In the 1980s I had not yet realized how confusing it was to speak of "the issue" of "intellectual property." That term is obviously biased; more subtle is the fact that it lumps together various disparate laws which raise very different issues. Nowadays I urge people to reject the term "intellectual property" entirely, lest it lead others to suppose that those laws form one coherent issue. The way to be clear is to discuss patents, copyrights, and trademarks separately. See *Did You Say "Intellectual Property"? It's a Seductive Mirage* for further explanation of how this term spreads confusion and bias.

the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied other authors at length in works of nonfiction. This practice was useful, and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: are we really better off granting such license? What kind of act are we licensing a person to do?

The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object code which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to.

"Competition makes things get done better."

The paradigm of competition is a race: by rewarding the winner, we encourage everyone to run faster. When capitalism really works this way, it does a good job; but its defenders are wrong in assuming it always works this way. If the runners forget why the reward is offered and become intent on winning, no matter how, they may find other strategies—such as, attacking other runners. If the runners get into a fist fight, they will all finish late.

Proprietary and secret software is the moral equivalent of runners in a fist fight. Sad to say, the only referee we've got does not seem to object to fights; he just regulates them ("For every ten yards you run, you can fire one shot"). He really ought to break them up, and penalize runners for even trying to fight.

"Won't everyone stop programming without a monetary incentive?"

Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way.

But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will not disappear, only become less. So the right question is, will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of nonmonetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself.

Then most of them left when offered a chance to do the same interesting work for a lot of money.

What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are banned.

"We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey."

You're never so desperate that you have to obey this sort of demand. Remember: millions for defense, but not a cent for tribute!

"Programmers need to make a living somehow."

In the short run, this is true. However, there are plenty of ways that programmers could make a living without selling the right to use a program. This way is customary now because it brings programmers and businessmen the most money, not because it is the only way to make a living. It is easy to find other ways if you want to find them. Here are a number of examples.

A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.

The sale of teaching, handholding and maintenance services could also employ programmers.

People with new ideas could distribute programs as freeware,[10] asking for donations from satisfied users, or selling handholding services. I have met people who are already working this way successfully.

Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a Software Tax:

Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.

But if the computer buyer makes a donation to software development himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

The total tax rate could be decided by a vote of the payers of the tax, weighted according to the amount they will be taxed on.

The consequences:

- The computer-using community supports software development.
- This community decides what level of support is needed.
- Users who care which projects their share is spent on can choose this for themselves.

[10] Subsequently we learned to distinguish between "free software" and "freeware." The term "freeware" means software you are free to redistribute, but usually you are not free to study and change the source code, so most of it is not free software. See *Words to Avoid (or Use with Care) Because They Are Loaded or Confusing* for more explanation.

In the long run, making programs free is a step toward the postscarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

We have already greatly reduced the amount of work that the whole society must do for its actual productivity, but only a little of this has translated itself into leisure for workers because much nonproductive activity is required to accompany productive activity. The main causes of this are bureaucracy and isometric struggles against competition. Free software will greatly reduce these drains in the area of software production. We must do this, in order for technical gains in productivity to translate into less work for us.

Appendix F 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 document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

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.

[1] The use of "hacker" to mean "security breaker" is a confusion on the part of the mass media. We hackers refuse to recognize that meaning, and continue using the word to mean someone who loves to program, someone who enjoys playful cleverness, or the combination of the two. See my article, "On Hacking," at http://stallman.org/articles/on-hacking.html.

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*.

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

[2] As an Atheist, I don't follow any religious leaders, but I sometimes find I admire something one of them has said.

- 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 compiler.

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. This 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 question 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 software. Most X users were running the proprietary versions that came with Unix systems, not the free version.

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 a 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 user 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, a crucial requirement for copyleft is to plug this hole: anything added to or combined with a copylefted 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

[3] In 1984 or 1985, Don Hopkins (a very imaginative fellow) mailed me a letter. On the envelope he had written several amusing sayings, including this one: "Copyleft—all rights reversed." I used the word "copyleft" to name the distribution concept I was developing at the time.

[4] We now use the *GNU Free Documentation License* for documentation.

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 manuals 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 over 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 the user's freedom.

[5] "Bourne Again Shell" is a play on the name "Bourne Shell," which was the usual shell on Unix.

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, we 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.

[6] That was written in 1998. In 2009 we no longer maintain a long task list. The community develops free software so fast that we can't even keep track of it all. Instead, we have a list of High Priority Projects, a much shorter list of projects we really want to encourage people to write.

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 C 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 do 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 them 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 the amount Readline is used, but that is no loss for us. Meanwhile, at least one useful application has been 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.

---

[7] This license is now called the GNU Lesser General Public License, to avoid giving the idea that all libraries ought to use it.

[8] Eric Raymond is a prominent open source advocate; see *Why Open Source Misses the Point of Free Software*.

[9] Eric S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, rev. ed. (Sebastopol, Calif.: O'Reilly, 2001), p. 23.

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 why 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. But it took a long time to make that possible, and the multithreaded servers that send messages to each other have turned out to be very hard to debug. Making the Hurd work solidly has stretched on for many years.

Alix

The GNU kernel was not originally supposed to be called the Hurd. Its original name was Alix—named after the woman who was my sweetheart at the time. She, a Unix system

administrator, had pointed out how her name would fit a common naming pattern for Unix system versions; as a joke, she told her friends, "Someone should name a kernel after me." I said nothing, but decided to surprise her with a kernel named Alix.

It did not stay that way. Michael (now Thomas) Bushnell, the main developer of the kernel, preferred the name Hurd, and redefined Alix to refer to a certain part of the kernel—the part that would trap system calls and handle them by sending messages to Hurd servers.

Later, Alix and I broke up, and she changed her name; independently, the Hurd design was changed so that the C library would send messages directly to servers, and this made the Alix component disappear from the design.

But before these things happened, a friend of hers came across the name Alix in the Hurd source code, and mentioned it to her. So she did have the chance to find a kernel named after her.

Linux and GNU/Linux

The GNU Hurd is not suitable for production use, and we don't know if it ever will be. The capability-based design has problems that result directly from the flexibility of the design, and it is not clear solutions exist.

Fortunately, another kernel is available. In 1991, Linus Torvalds developed a Unix-compatible kernel and called it Linux. In 1992, he made Linux free software; combining Linux with the not-quite-complete GNU system resulted in a complete free operating system. (Combining them was a substantial job in itself, of course.) It is due to Linux that we can actually run a version of the GNU system today.

We call this system version GNU/Linux, to express its composition as a combination of the GNU system with Linux as the kernel.

Challenges in Our Future

We have proved our ability to develop a broad spectrum of free software. This does not mean we are invincible and unstoppable. Several challenges make the future of free software uncertain; meeting them will require steadfast effort and endurance, sometimes lasting for years. It will require the kind of determination that people display when they value their freedom and will not let anyone take it away.

The following four sections discuss these challenges.

Secret Hardware

Hardware manufacturers increasingly tend to keep hardware specifications secret. This makes it difficult to write free drivers so that Linux and XFree86 can support new hardware. We have complete free systems today, but we will not have them tomorrow if we cannot support tomorrow's computers.

There are two ways to cope with this problem. Programmers can do reverse engineering to figure out how to support the hardware. The rest of us can choose the hardware that is supported by free software; as our numbers increase, secrecy of specifications will become a self-defeating policy.

Reverse engineering is a big job; will we have programmers with sufficient determination to undertake it? Yes—if we have built up a strong feeling that free software is a matter of principle, and nonfree drivers are intolerable. And will large numbers of us spend extra

money, or even a little extra time, so we can use free drivers? Yes, if the determination to have freedom is widespread.

[2008 note: this issue extends to the BIOS as well. There is a free BIOS, coreboot; the problem is getting specs for machines so that coreboot can support them.]

Nonfree Libraries

A nonfree library that runs on free operating systems acts as a trap for free software developers. The library's attractive features are the bait; if you use the library, you fall into the trap, because your program cannot usefully be part of a free operating system. (Strictly speaking, we could include your program, but it won't *run* with the library missing.) Even worse, if a program that uses the proprietary library becomes popular, it can lure other unsuspecting programmers into the trap.

The first instance of this problem was the Motif toolkit, back in the 80s. Although there were as yet no free operating systems, it was clear what problem Motif would cause for them later on. The GNU Project responded in two ways: by asking individual free software projects to support the free X Toolkit widgets as well as Motif, and by asking for someone to write a free replacement for Motif. The job took many years; LessTif, developed by the Hungry Programmers, became powerful enough to support most Motif applications only in 1997.

Between 1996 and 1998, another nonfree GUI toolkit library, called Qt, was used in a substantial collection of free software, the desktop KDE.

Free GNU/Linux systems were unable to use KDE, because we could not use the library. However, some commercial distributors of GNU/Linux systems who were not strict about sticking with free software added KDE to their systems—producing a system with more capabilities, but less freedom. The KDE group was actively encouraging more programmers to use Qt, and millions of new "Linux users" had never been exposed to the idea that there was a problem in this. The situation appeared grim.

The free software community responded to the problem in two ways: GNOME and Harmony.

GNOME, the GNU Network Object Model Environment, is GNU's desktop project. Started in 1997 by Miguel de Icaza, and developed with the support of Red Hat Software, GNOME set out to provide similar desktop facilities, but using free software exclusively. It has technical advantages as well, such as supporting a variety of languages, not just C++. But its main purpose was freedom: not to require the use of any nonfree software.

Harmony is a compatible replacement library, designed to make it possible to run KDE software without using Qt.

In November 1998, the developers of Qt announced a change of license which, when carried out, should make Qt free software. There is no way to be sure, but I think that this was partly due to the community's firm response to the problem that Qt posed when it was nonfree. (The new license is inconvenient and inequitable, so it remains desirable to avoid using Qt.)

[Subsequent note: in September 2000, Qt was rereleased under the GNU GPL, which essentially solved this problem.]

How will we respond to the next tempting nonfree library? Will the whole community understand the need to stay out of the trap? Or will many of us give up freedom for convenience, and produce a major problem? Our future depends on our philosophy.

Software Patents

The worst threat we face comes from software patents, which can put algorithms and features off limits to free software for up to 20 years. The LZW compression algorithm patents were applied for in 1983, and we still cannot release free software to produce proper compressed GIFs. [As of 2009 they have expired.] In 1998, a free program to produce MP3 compressed audio was removed from distribution under threat of a patent suit.

There are ways to cope with patents: we can search for evidence that a patent is invalid, and we can look for alternative ways to do a job. But each of these methods works only sometimes; when both fail, a patent may force all free software to lack some feature that users want. What will we do when this happens?

Those of us who value free software for freedom's sake will stay with free software anyway. We will manage to get work done without the patented features. But those who value free software because they expect it to be technically superior are likely to call it a failure when a patent holds it back. Thus, while it is useful to talk about the practical effectiveness of the "bazaar" model of development, and the reliability and power of some free software, we must not stop there. We must talk about freedom and principle.

Free Documentation

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

Free documentation, like free software, is a matter of freedom, not price. The criterion for a free manual is pretty much the same as for free software: it is a matter of giving all users certain freedoms. Redistribution (including commercial sale) must be permitted, online and on paper, so that the manual can accompany every copy of the program.

Permission for modification is crucial too. As a general rule, I don't believe that it is essential for people to have permission to modify all sorts of articles and books. For example, I don't think you or I are obliged to give permission to modify articles like this one, which describe our actions and our views.

But there is a particular reason why the freedom to modify is crucial for documentation for free software. When people exercise their right to modify the software, and add or change its features, if they are conscientious they will change the manual, too—so they can provide accurate and usable documentation with the modified program. A nonfree manual, which does not allow programmers to be conscientious and finish the job, does not fill our community's needs.

Some kinds of limits on how modifications are done pose no problem. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are OK. It is also no problem to require modified versions to include notice that they were modified, even to have entire sections that may not be deleted or changed, as long as these sections deal with nontechnical topics. These kinds of restrictions

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

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

Will free software developers have the awareness and determination to produce a full spectrum of free manuals? Once again, our future depends on philosophy.

We Must Talk about Freedom

Estimates today are that there are ten million users of GNU/Linux systems such as Debian GNU/Linux and Red Hat "Linux." Free software has developed such practical advantages that users are flocking to it for purely practical reasons.

The good consequences of this are evident: more interest in developing free software, more customers for free software businesses, and more ability to encourage companies to develop commercial free software instead of proprietary software products.

But interest in the software is growing faster than awareness of the philosophy it is based on, and this leads to trouble. Our ability to meet the challenges and threats described above depends on the will to stand firm for freedom. To make sure our community has this will, we need to spread the idea to the new users as they come into the community.

But we are failing to do so: the efforts to attract new users into our community are far outstripping the efforts to teach them the civics of our community. We need to do both, and we need to keep the two efforts in balance.

"Open Source"

Teaching new users about freedom became more difficult in 1998, when a part of the community decided to stop using the term "free software" and say "open source software" instead.

Some who favored this term aimed to avoid the confusion of "free" with "gratis"—a valid goal. Others, however, aimed to set aside the spirit of principle that had motivated the free software movement and the GNU Project, and to appeal instead to executives and business users, many of whom hold an ideology that places profit above freedom, above community, above principle. Thus, the rhetoric of "open source" focuses on the potential to make high-quality, powerful software, but shuns the ideas of freedom, community, and principle.

The "Linux" magazines are a clear example of this—they are filled with advertisements for proprietary software that works with GNU/Linux. When the next Motif or Qt appears, will these magazines warn programmers to stay away from it, or will they run ads for it?

The support of business can contribute to the community in many ways; all else being equal, it is useful. But winning their support by speaking even less about freedom and principle can be disastrous; it makes the previous imbalance between outreach and civics education even worse.

"Free software" and "open source" describe the same category of software, more or less, but say different things about the software, and about values. The GNU Project continues

to use the term "free software," to express the idea that freedom, not just technology, is important.

Try!

Yoda's aphorism ("There is no 'try'") sounds neat, but it doesn't work for me. I have done most of my work while anxious about whether I could do the job, and unsure that it would be enough to achieve the goal if I did. But I tried anyway, because there was no one but me between the enemy and my city. Surprising myself, I have sometimes succeeded.

Sometimes I failed; some of my cities have fallen. Then I found another threatened city, and got ready for another battle. Over time, I've learned to look for threats and put myself between them and my city, calling on other hackers to come and join me.

Nowadays, often I'm not the only one. It is a relief and a joy when I see a regiment of hackers digging in to hold the line, and I realize, this city may survive—for now. But the dangers are greater each year, and now Microsoft has explicitly targeted our community. We can't take the future of freedom for granted. Don't take it for granted! If you want to keep your freedom, you must be prepared to defend it.

Appendix G Linux and the GNU system

The original version of this essay was published as the file `etc/LINUX-GNU` in the GNU Emacs distribution.

This document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

The GNU project started in 1984 with the goal of developing a complete free Unix-like operating system: GNU. "Free" refers to freedom, not price; it means you are free to run, copy, distribute, study, change, and improve the software.

A Unix-like system consists of many different programs. We found some components already available as free software—for example, X Windows and TEX. We obtained other components by helping to convince their developers to make them free—for example, the Berkeley network utilities. This left many missing components that we had to write in order to produce GNU—for example, GNU Emacs, the GNU C compiler, the GNU C library, Bash, and Ghostscript. The GNU system consists of all these components together.

The GNU project is not just about developing and distributing some useful free software. The heart of the GNU project is an idea: that software should be free, that software users should have freedom to participate in a community. To run your computer, you need an operating system; if it is not free, your freedom has been denied. To have freedom, you need a free operating system. We therefore set out to write one.

In the long run, though, we cannot expect to keep the free operating system free unless the users are aware of the freedom it gives them, and value that freedom. People who do not appreciate their freedom will not keep it long. If we want to make freedom last, we need to spread awareness of the freedoms they have in free software.

The GNU project's method is that free software and the idea of users' freedom support each other. We develop GNU software, and as people encounter GNU programs or the GNU system and start to use them, they also think about the GNU idea. The software shows that the idea can work in practice. Some of these people come to agree with the idea, and then they are more likely to write additional free software. Thus, the software embodies the idea, spreads the idea, and grows from the idea.

Early on in the development of GNU, various parts of it became popular even though users needed proprietary systems to run them on. Porting the system to many systems and maintaining them required a lot of work. After that work, most GNU software is easily configured for a variety of different platforms.

By 1991, we had found or written all of the essential major components of the system except the kernel, which we were writing.[1]

That was the situation when Linux came into being. Linux is a kernel, like the kernel of Unix; it was written by Linus Torvalds, who released it under the GNU General Public

[1] This kernel consists of the Mach microkernel plus the GNU HURD. The first test release was made in 1996. Now, in 2002, it is running well, and Hurd-based GNU systems are starting to be used.

License. He did not write this kernel for GNU, but it fit into the gap in GNU. The combination of GNU and Linux included all the major essential components of a Unix-compatible operating system. Other people, with some work made the combination into a usable system. The principal use of Linux, the kernel, is as part of this combination.

The popularity of the GNU/Linux combination is success, in the sense of popularity, for GNU. Ironically, the popularity of GNU/Linux undermines our method of communicating the ideas of GNU to people who use GNU.

When GNU programs were only usable individually on top of another operating system, installing and using them meant knowing and appreciating these programs, and thus being aware of GNU, which led people to think about the philosophical base of GNU. Now users can install a unified operating system which is basically GNU, but they usually think these are "Linux systems". At first impression, a "Linux system" sounds like something completely distinct from the "GNU system," and that is what most users think.

This leads many users to identify themselves as a separate community of "Linux users", distinct from the GNU user community. They use more than just some GNU programs, they use almost all of the GNU system, but they don't think of themselves as GNU users. Often they never hear about the GNU idea; if they do, they may not think it relates to them.

Most introductions to the "Linux system" acknowledge that GNU software components play a role in it, but they don't say that the system as a whole is a modified version of the GNU system that the GNU project has been developing and compiling since Linus Torvalds was in junior high school. They don't say that the main reason this free operating exists is that the GNU Project worked persistently to achieve its goal of freedom.

As a result, most users don't know these things. They believe that the "Linux system" was developed by Linus Torvalds "just for fun", and that their freedom is a matter of good fortune rather than the dedicated pursuit of freedom. This creates a danger that they will leave the survival of free software to fortune as well.

Since human beings tend to correct their first impressions less than called for by additional information they learn later, these users will tend to continue to underestimate their connection to GNU even if they do learn the facts.

When we began trying to support the GNU/Linux system, we found this widespread misinformation led to a practical problem—it hampered cooperation on software maintenance. Normally when users change a GNU program to make it work better on a particular system, they send the change to the maintainer of that program; then they work with the maintainer, explaining the change, arguing for it, and sometimes rewriting it for the sake of the overall coherence and maintainability of the package, to get the patch installed. But people who thought of themselves as "Linux users" showed a tendency to release a forked "Linux-only" version of the GNU program and consider the job done. In some cases we had to redo their work in order to make GNU programs run as released in GNU/Linux systems.

How should the GNU project encourage its users to cooperate? How should we spread the idea that freedom for computer users is important?

We must continue to talk about the freedom to share and change software—and to teach other users to value these freedoms. If we value having a free operating system, it makes sense to think about preserving those freedoms for the long term. If we value having a

variety of free software, it makes sense to think about encouraging others to write free software, instead of proprietary software.

However, it is not enough just to talk about freedom; we must also make sure people know the reasons it is worth listening to what we say.

Long explanations such as our philosophical articles are one way of informing the public, but you may not want to spend so much time on the matter. The most effective way you can help with a small amount of work is simply by using the terms "Linux-based GNU system" or "GNU/Linux system", instead of "Linux system," when you write about or mention such a system. Seeing these terms will show many people the reason to pay attention to our philosophical articles.

The system as a whole is more GNU than Linux; the name "GNU/Linux" is fair. When you are choosing the name of a distribution or a user group, a name with "GNU/Linux" will reflect both roots of the combined system, and will bring users into connection with both—including the spirit of freedom and community that is the basis and purpose of GNU.

Appendix H Why Software Should Not Have Owners

This essay was originally published in *Technos: Quarterly for Education and Technology*, vol. 3, n. 2, pp. 24–26, Summer 1994.

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

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

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

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

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

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

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

[1] The charges were subsequently dismissed.

Name Calling

Owners use smear words such as "piracy" and "theft," as well as expert terminology such as "intellectual property" and "damage," to suggest a certain line of thinking to the public—a simplistic analogy between programs and physical objects.

Our ideas and intuitions about property for material objects are about whether it is right to *take an object away* from someone else. They don't directly apply to *making a copy* of something. But the owners ask us to apply them anyway.

Exaggeration

Owners say that they suffer "harm" or "economic loss" when users copy programs themselves. But the copying has no direct effect on the owner, and it harms no one. The owner can lose only if the person who made the copy would otherwise have paid for one from the owner.

A little thought shows that most such people would not have bought copies. Yet the owners compute their "losses" as if each and every one would have bought a copy. That is exaggeration—to put it kindly.

The Law

Owners often describe the current state of the law, and the harsh penalties they can threaten us with. Implicit in this approach is the suggestion that today's law reflects an unquestionable view of morality—yet at the same time, we are urged to regard these penalties as facts of nature that can't be blamed on anyone.

This line of persuasion isn't designed to stand up to critical thinking; it's intended to reinforce a habitual mental pathway.

It's elementary that laws don't decide right and wrong. Every American should know that, in the 1950s, it was against the law in many states for a black person to sit in the front of a bus; but only racists would say sitting there was wrong.

Natural Rights

Authors often claim a special connection with programs they have written, and go on to assert that, as a result, their desires and interests concerning the program simply outweigh those of anyone else—or even those of the whole rest of the world. (Typically companies, not authors, hold the copyrights on software, but we are expected to ignore this discrepancy.)

To those who propose this as an ethical axiom—the author is more important than you—I can only say that I, a notable software author myself, call it bunk.

But people in general are only likely to feel any sympathy with the natural rights claims for two reasons.

One reason is an overstretched analogy with material objects. When I cook spaghetti, I do object if someone else eats it, because then I cannot eat it. His action hurts me exactly as much as it benefits him; only one of us can eat the spaghetti, so the question is, which one? The smallest distinction between us is enough to tip the ethical balance.

But whether you run or change a program I wrote affects you directly and me only indirectly. Whether you give a copy to your friend affects you and your friend much more than it affects me. I shouldn't have the power to tell you not to do these things. No one should.

The second reason is that people have been told that natural rights for authors is the accepted and unquestioned tradition of our society.

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

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

Economics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

You deserve free software.

Appendix I Why Free Software Needs Free Documentation

This essay was originally published on http://gnu.org, in 1996.

This document is part of GNU philosophy, the GNU Project's exhaustive collection of articles and essays about free software and related matters.

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

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

Why was this? The authors of the good manuals had written them for O'Reilly Associates, which published them with restrictive terms—no copying, no modification, source files not available—which exclude them from the free software community.

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

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

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

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

As a general rule, I don't believe that it is essential for people to have permission to modify all sorts of articles and books. The issues for writings are not necessarily the same as those for software. For example, I don't think you or I are obliged to give permission to modify articles like this one, which describe our actions and our views.

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

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

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

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

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

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

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

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

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

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

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z