Information For Maintainers of GNU Software

Richard Stallman
last updated 08 November 1994

1 About This Document

This file contains guidelines and advice for someone who is the maintainer of a GNU program on behalf of the GNU project. Anyone can change GNU software, but there's no need to pay attention to these guidelines unless you are maintaining a version for widespread distribution.

Corrections or suggestions regarding this document should be sent to `gnu@prep.ai.mit.edu`. If you make a suggestion, please include a suggested new wording for it; our time is limited. We prefer a context diff to the `maintain.texi` file, but if you don't have that file, please mail your suggestion anyway.

This release of the GNU Maintenance Instructions was last updated 08 November 1994.

2 Legal Matters

When incorporating changes from other people, make sure to follow the correct procedures. Doing this ensures that the FSF has the legal right to distribute and defend GNU software.

2.1 Copyrights

For the sake of registering the copyright on later versions of the software, you need to keep track of each person who makes significant changes. A change of ten lines or so, or a few such changes, in a large program is not significant.

Before incorporating significant changes, make sure that the person has signed copyright papers and that the Foundation has received and signed them.

You can tell the person what papers to sign by email. For large changes, ask for an assignment. Send the person a copy of `/gd/gnuorg/assign.changes`, but first, go to the second page and insert the person's name and the name of the program involved in place of '`NAME OF PERSON`' and '`NAME OF PROGRAM`'. Do this before sending, because otherwise the person might sign without noticing them. Then the papers would be useless.

For medium to small changes, ask for a disclaimer. Use the file `/gd/gnuorg/disclaim.changes`.

To check whether papers have been received, look in `/gd/gnuorg/copyright.list`. If you can't look there directly, `fsf-records@prep.ai.mit.edu` can check for you, and can also check for papers that are waiting to be entered and inform you when expected papers arrive.

You can also send the person `/gd/gnuorg/conditions.text`, which explains his options (assign vs. disclaim) and their consequences.

2.2 Recording Changes

Keep records of which portions were written by whom.

These records don't need to be as detailed as a change log. They don't need to distinguish work done at different times, only different people.

They should say which files or functions were written by each person, and which files or functions were revised by each person. They don't need to say what the purpose of the change was. The Register of Copyrights doesn't care what the program does.

For example, this would describe an early version of GAS:

Dean Elsner first version of all files except gdb-lines.c and m68k.c.
Jay Fenlason entire files gdb-lines.c and m68k.c, most of app.c,
extensive changes in messages.c, input-file.c, write.c,
revisions elsewhere.

Please keep these records in a file named `AUTHORS` in the source directory for the program itself.

3 Cleaning Up Changes

If someone sends you changes which are ugly and will make the program harder to understand and maintain in the future, such as a port to another operating system containing ad hoc conditionals, don't hesitate to ask the person to clean up his changes before you merge them.

Since the amount of work we do is constant in any case, the more work we get other people to do, the faster GNU will advance.

If the person will not or can not make the changes clean enough, then say that you can't afford to merge them. Invite him to distribute his changes himself, or to find other people who can make them clean enough for us to maintain.

The only reason to do these cleanups yourself is if (1) it is easy enough that it is less work than telling the author what to clean up, or (2) users will greatly appreciate the improvement, and you would almost write it yourself if you had time.

The GNU Coding Standards are a good thing to send people who have to clean up C programs (see Section "Contents" in *GNU Coding Standards*). The Emacs Lisp manual contains an appendix that gives coding standards for Emacs Lisp programs; it is good to urge authors to read it (see Section "Tips and Standards" in *The GNU Emacs Lisp Reference Manual*).

4 Dealing With Mail

Once a program is in use, you will start getting bug reports. Some GNU programs have their own special lists for sending bug reports. For miscellaneous programs that don't have their own lists, we use a catch-all list, `bug-gnu-utils@prep.ai.mit.edu`. Talk with `gnu@prep.ai.mit.edu` to arrange to be added to the proper list for your program.

When you receive bug reports, keep in mind that the main purpose of the bug reports is to enable you to improve the next version of the program. Helping individuals is only secondary. So you don't need to give a substantial response unless you have a reason to (for example, to ask for more information, or to ask the user to test a fix). But it is good to respond to the rest of the bug reports with just "Thanks." That is quick and tells the user that the bug report was useful.

As a practical matter, any time you spend helping individuals beyond what is necessary for you, takes time away from maintaining the program. While this may seem "friendly" and "helpful," actually it is counterproductive. If you help one person when you could instead have helped thousands, the world is worse off.

5 Recording Old Versions

It is very important to keep backup files of all source files of GNU. You can do this using RCS if you like. The easiest way to use RCS is via the Version Control library in Emacs; Section "Concepts of Version Control" in *The GNU Emacs Manual*.

Alternatively, you can keep backup files.

5.1 Backup Files

Emacs makes a backup file by renaming the old source file to a new name. This means that if the file is also pointed to by a distribution directory, the distribution directory continues to point to the same version it always did—the right thing.

We want to keep more than one backup for all GNU sources. So, if you are going to edit GNU sources, *make certain* to put

```
(setq version-control t)
```

into your `.emacs` file, so that Emacs always creates numbered backup files.

Using Emacs backup files works as long as people always make changes with Emacs. If you change the file in some other way, and use `cp`, `ftp`, or `tar` to install it, you will *overwrite* the old version and fail to make a backup. Don't do that!

If you want to make a change to a source file with something other than Emacs, you can write the changed file to another name, and use *C-x C-w* in Emacs to write it under the real name. This makes the backup file properly.

You can use GNU `cp` or `mv` to install changed files if you give them the '`--backup`' (or, equivalently, '`-b`') option; then they make backup copies the same way that Emacs does. You should also use the '`--version-control=t`' option, or, alternately, first

```
export VERSION_CONTROL=t
```

(or the `csh` equivalent); this makes GNU `cp` and `mv` create numbered backup files instead of a single backup file with a '`~`' appended to the filename. For installing many changed files, you can use shell wildcards and also give GNU `cp` or `mv` the '`--update`' ('`-u`') option, which only copies (or moves) files that have been modified more recently than the existing destination files, and the '`--verbose`' ('`-v`') option, which prints the names of the files that are actually copied (or moved).

Before you use `mv` or `cp` in this way, *make sure it is the GNU version.* Do '`which mv`' (in `csh`) or '`type mv`' (in `bash`) to verify you are not getting `/bin/mv` (or likewise for `cp`). Or just type '`cp`' or '`mv`' and look at the usage message.

5.2 Deleting Backup Files

Always answer no when Emacs offers to delete backup files automatically. The way to delete them is by hand, using *M-x dired*.

When you decide which backup files to delete, it is good to keep one every couple of weeks or so, going back about 2 months.

If there is a long gap between versions, because that file did not change during a long period of time, then keep the version before the gap even if it is 2 months old. Pretend it

is just 2 weeks older than the next kept version, so delete it when the next version is >= 6 weeks old.

If the changes in a program have been simple, then you don't need to keep as many backup files. Just one a month for 2 months is enough.

If you have made big changes, keep the versions before and after the big change, until they are old enough.

If you made several changes the same day, usually the last version written that day is best to keep.

It is almost always right to keep the most recent backup version.

6 Archives

For each program, you should keep a special magtape or cartridge as an archive. Each time you release a new version, `dd` the tar file onto the end of the tape. Keep a list of versions on the tape's paper label, and add to it each time you add to the tape.

For cartridges, you can type

```
mt -f /dev/nrst8 eom
```

to go straight to the end of the data on the tape.

For reel-to-reel tapes, there is no automated way to go to the end of the data on the tape. You have to count the number of files (based on the written label), and space forward over them with '`mt fsf`'.

To be safe, it is important to check your count. If the count is *n*, then do:

```
mt -f /dev/nrmt8 fsf n-1
```

This puts you at the beginning of the last existing tar file.

Then do

```
tar tf /dev/nrmt8
```

to list that tar file. If the version number appears in a directory name, which is a good idea, you can use this to verify that you have reached the tar file you wanted to reach.

C-c the `tar` before it finishes; then do

```
mt -f /dev/nrmt8 fsf
```

to skip past it and its end-of-file marker.

To copy the new distribution file onto cartridge tape, do:

```
dd if=tar-file-name of=/dev/nrst8 bs=102400
```

(This specifies a blocking factor of 200.)

For reel-to-reel tape, do:

```
dd if=tar-file-name of=/dev/nrmt8 bs=10240
```

(This specifies a blocking factor of 20.)

When the tape gets full, put it aside permanently and start writing another.

7 Distributions

It is important to follow the GNU conventions when making GNU software distributions.

7.1 Distribution tar Files

The tar file for version *m.n* of program `foo` should be named `foo-m.n.tar`. It should unpack into a subdirectory named `foo-m.n`. Tar files should not unpack into files in the current directory, because this is inconvenient if the user happens to unpack into a directory with other files in it.

Here is how the `Makefile` for Bison creates the tar file. This method is good for other programs.

```
dist: bison.info
        echo bison-‘sed -e ’/version_string/!d’ \
          -e ’s/[^0-9.]*\([0-9.]*\).*/\1/’ -e q version.c‘ > .fname
        -rm -rf ‘cat .fname‘
        mkdir ‘cat .fname‘
        dst=‘cat .fname‘; for f in $(DISTFILES); do \
           ln $(srcdir)/$$f $$dst/$$f || { echo copying $$f; \
             cp -p $(srcdir)/$$f $$dst/$$f ; } \
        done
        tar --gzip -chf ‘cat .fname‘.tar.gz ‘cat .fname‘
        -rm -rf ‘cat .fname‘ .fname
```

Source files that are symbolic links to other file systems cannot be installed in the temporary directory using `ln`, so use `cp` if `ln` fails.

7.2 Distribution Patches

If the program is large, it is useful to make a set of diffs for each release, against the previous important release.

At the front of the set of diffs, put a short explanation of which version this is for and which previous version it is relative to. Also explain what else people need to do to update the sources properly (for example, delete or rename certain files before installing the diffs).

The purpose of having diffs is that they are small. To keep them small, exclude files that the user can easily update. For example, exclude info files, DVI files, tags tables, output files of Bison or Flex. In Emacs diffs, we exclude compiled Lisp files, leaving it up to the installer to recompile the patched sources.

When you make the diffs, each version should be in a directory suitably named—for example, `gcc-2.3.2` and `gcc-2.3.3`. This way, it will be very clear from the diffs themselves which version is which.

If you use GNU `diff` to make the patch, use the options ‘`-rc2P`’. That will put any new files into the output as “entirely different.”

If the distribution has subdirectories in it, then the diffs probably include some files in the subdirectories. To help users install such patches reliably, give them precise directions for how to run patch. For example, say this:

To apply these patches, cd to the main directory of the program

and then use 'patch -p1'. '-p1' avoids guesswork in choosing
which subdirectory to find each file in.

It's wise to test your patch by applying it to a copy of the old version, and checking that the result exactly matches the new version.

7.3 Distribution on `prep`

Only the latest version of any program needs to be on `prep`. Being an archive of old versions is not the function of `prep`.

Diffs are another matter. Since they are much smaller than distribution files, it is good to keep the diffs around for quite a while.

7.4 Test Releases

When you release a greatly changed new major version of a program, you might want to do so as a beta test release.

Once a program gets to be widely used and people expect it to work solidly, it is a good idea to do pretest releases before each "real" release. This means that you make a tar file, but send it only to a group of volunteers that you have recruited. (Use a suitable GNU mailing list/newsgroup to recruit them.)

One thing that you should never do is to release a distribution which is considered a pretest or beta test but which contains the version number for the planned real release. Many people will look only at the version number (in the tar file name, in the directory name that it unpacks into, or wherever they can find it) to determine whether a tar file is the latest version. People might look at the test release in this way and mistake it for the real release. Therefore, always change the number when you make a new tar file.

If you are about to release version 4.6 but you want to do a pretest or beta test first, call it 4.5.90. If you need a second pretest, call it 4.5.91, and so on.

Table of Contents