# 7

# Documentation

Ask any real guru a question, so the saying goes, and he will reply with a cryptic "*RTFM*".[*] Cynics claim this is even the answer to the question "Where can I find the manual?" All too often, programmers consider documentation a necessary (or even unnecessary) evil, and if it gets done at all, it's usually the last thing that gets done. This is particularly evident when you look at the quality of documentation supplied with some free software packages (though many free packages, such as most of those from the Free Software Foundation, are very well documented). The quality and kind of the documentation in source packages varies wildly. In Chapter 2, *Unpacking the goodies*, page 25, we looked at the documentation that *should* be automatically supplied with the package to describe what it is and how to install it. In this chapter, we'll look at documentation that is intended for use after you have installed the package.

The documentation you get with a package is usually in one of the following formats:

- *man pages*, the traditional on-line documentation for UNIX, which are formatted with *nroff*.
- *info* files, used with the GNU project's *info* on-line documentation reader.
- Unformatted *roff*, *TEX*, or *texinfo* hardcopy documentation.
- Preformatted documentation in PostScript or *.dvi* format, or occasionally in other formats such as HP LaserJet.

We know where we want to get to—the formatted documentation—but we don't always know where to start, so it's easier to look at documentation in reverse order: first, we'll look at the end result, then at the formatters, and finally at the input files.

## Preformatted documentation

Occasionally you get documentation that has been formatted so that you can print it on just about any printer, but this doesn't happen very much: in order to achieve this, the text must be free of any frills and formatted so that any typewriter can print it. Nearly any printer

* "Read The Manual"—the F is usually silent.

nowadays is capable of better results, so preformatted files are usually supplied in a format that can print high quality printout on a laser printer. The following three are about the only ones you will come across:

- *PostScript* is a specialized programming language for printers, and the printed data are in fact embedded in the program. This makes it an extremely flexible format.
- *.dvi* is the format that is output by TEX. In order to print it, you need a TEX driver.
- Unlike PostScript and *.dvi*, the *Hewlett-Packard LaserJet* format is not portable: you need a LaserJet-compatible printer to print it. The LaserJet format is obsolescent: even many LaserJet printers made today also support PostScript, and there are programmatic ways to print PostScript on other laser printers, so there is little motivation for using the much more restrictive LaserJet format.

## PostScript

PostScript is the current format of choice. Because it is a programming language, it is much more flexible than conventional data formats. For example, it is easily scalable. You can take a file intended for a phototypesetter with a resolution of 2540 bpi and print it on a laser printer, and it will come out correctly.* In addition, better quality printers perform the formatting themselves, resulting in a considerable load reduction for the computer. A large number of printers and all modern phototypesetters can process PostScript directly.

If your printer doesn't handle PostScript, you can use programs like *ghostscript*, which interpret PostScript programs and output in a multitude of other formats, including LaserJet, so even if you have a LaserJet, it can be a better idea to use PostScript format. *ghostscript* is distributed by the Free Software Foundation—see Appendix E, *Where to get sources*. *ghostscript* can also display PostScript files on X displays.

Most PostScript files are encoded in plain ASCII without any control characters except newline (though that doesn't make them easy to read). Even when you include special characters in your text, they appear in the PostScript document as plain ASCII sequences. It's usually pretty easy to recognize PostScript, even without the *file* program. Here's the start of a draft version of this chapter:

```
%!PS-Adobe-3.0
%%Creator: groff version 1.09
%%CreationDate: Thu Aug 18 17:34:24 1994
%%DocumentNeededResources: font Times-Bold
```

The data itself is embedded in parentheses between the commands. Looking at a draft of this text, we see things like

```
(It’)79.8 273.6 Q 2.613(su)-.55 G .113
(sually pretty easy to recognize a PostScript program, e)-2.613 F -.15
(ve)-.25 G 2.614(nw).15 G .114(ithout the)-2.614 F F2(\214le)2.614 E F1
(program--here’)79.8 285.6 Q 2.5(st)-.55 G(he start of a draft v)-2.5 E
```

* You may have to wait a while before a few megabytes of font information are transferred and processed, but eventually you get your document.

## Problems with PostScript

PostScript doesn't pose too many problems, but occasionally you might see one of these:

*Missing fonts*

PostScript documents include information about the fonts they require. Many fonts are built in to printers and PostScript display software, but if the fonts are not present, the system chooses a default value which may have little in common with the font which the document requested. The default font is typically Courier, which is `fixed-width`, and the results look terrible. If this happens, you can find the list of required fonts with the following:

```
$ grep '%%.* font ' mumble.ps
%%DocumentNeededResources: font Garamond-BookItalic
%%+ font Times-Roman
%%+ font Garamond-Light
%%+ font Garamond-LightItalic
%%+ font Courier
%%+ font Garamond-Book
%%+ font Courier-Bold
%%IncludeResource: font Garamond-BookItalic
%%IncludeResource: font Times-Roman
%%IncludeResource: font Garamond-Light
%%IncludeResource: font Garamond-LightItalic
%%IncludeResource: font Courier
%%IncludeResource: font Garamond-Book
%%IncludeResource: font Courier-Bold
(%%DocumentNeededResources: font Times-Bold)131.711 327.378 S F1 1.281
```

This extracts the font requests from the PostScript file: in this case, the document requires Times Roman, Courier and Garamond fonts. Just about every printer and software package supplies Times Roman and Courier, but Garamond (the font in which this book is written) is less common. In addition, most fonts are copyrighted, so you probably won't be able to find them on the net. If you have a document like this in PostScript format, your choices are:

- Reformat it with a different font if you have the source.
- Get the Garamond fonts.
- Edit the file and change the name of the font to a font with similar metrics (in other words, with similar size characters). The results won't be as good, but if the font you find is similar enough, they might be acceptable. For example, you might change the text *Garamond* to *Times Roman*.

*Wrong font type*

*Most* PostScript fonts are in plain ASCII. You may also come across *Type 2 PostScript* and *display PostScript*, both of which include binary data. Many printers can't understand the binary format, and they may react to it in an unfriendly way. For example, my National KX-P 4455 printer just hangs if I copy display PostScript to it. See the section *format conversion* below for ways to solve this dilemma.

## .dvi format

One of the goals of $\TeX$ was to be able to create output for just about any printer. As we will see, old versions of *troff*, the main competitor, were able to produce output only for a very limited number of phototypesetters. Even if you have one of them in your office, it's unlikely that you will want to use it for printing out a draft of a 30-page paper.

The $\TeX$ solution, which was later adopted by *troff* in *ditroff* (device independent troff), was to output the formatted data in a *device-independent* format, *.dvi*, and leave it to another program, a so-called *driver*, to format the files in a format appropriate to the output device. Unlike PostScript, *.dvi* contains large numbers of control characters and characters with the sign bit set, and is not even remotely legible. Most versions of *file* know about *.dvi* format.

## Format conversion

Not so long ago your choice of documentation software determined your output format. For example, if you used $\TeX$, you would get *.dvi* output, and you would need a $\TeX$ driver to print it. Nowadays, it's becoming easier to handle file formats. GNU *troff* will output in *.dvi* format if you wish, and programs are available to convert from *.dvi* to PostScript and back again. Here's a list of conversions you might like to perform—see Appendix E, *Where to get sources* for how to get software to perform them.

- A number of programs convert from *.dvi* to PostScript—for example, *dvips*.
- There's no good reason to want to convert from *PostScript to .dvi*, so there are no programs available. *.dvi* is not much use in itself—it needs to be tranformed to a final printer form, and if you have PostScript output, you can do that directly with *ghostscript* (see below) without going via *.dvi*.
- To display *.dvi* files on an X display, use *SeeTeX*.
- To convert from *.dvi* to a printer output format, use one of the *dvi2xxx* programs.
- To convert from PostScript to a printer format, use *ghostscript*.
- To display PostScript on an X display, you can also use *ghostscript*, but *ghostview* gives you a better interface.
- To convert PostScript with binary data into ASCII, use *t1ascii*.

# roff and friends

The original UNIX formatting program was called *roff* (for *run-off)*. It is now completely obsolete, but it has a number of descendents:

- *nroff* is a comparatively simple formatter designed to produce output for plain ASCII displays and printers.
- *troff* is a more sophisticated formatter designed to produce output for phototypesetters. Many versions create output only for the obsolete APS-5 phototypesetter, and you need

postprocessing software to convert this output to something that modern typesetters or laser printers understand. Fortunately, versions of *troff* that produce PostScript output are now available.

- *ditroff* (device independent troff) is a newer version of *troff* that produces output in a device-independent intermediate form that can then be converted into the final form by a conversion program. This moves the problem of correct output format from *troff* to the conversion program. Despite the terminology, this device-independent format is not the same as *.dvi* format.
- *groff* is the GNU project *troff* and *nroff* replacement. In *troff* mode it can produce output in PostScript and *.dvi* format.

All versions of *roff* share the same source file syntax, though *nroff* is more restricted in its functionality than *troff*. If you have a usable version of *troff*, you can use it to produce properly formatted hardcopy versions of the man pages, for example. This is also what *xman* (the X11 manual browser) does.

## formatting with nroff or troff

*troff* input bears a certain resemblance to the traces left behind when a fly falls into an inkwell and then walks across a desk. The first time you run *troff* against a file intended for *troff*, the results may be less than heartening. For example, consider the following passage from the documentation of the Revision Control System RCS. When correctly formatted, the output is:

Besides the operations *ci* and *co*, RCS provides the following commands:

| | |
|---|---|
| *ident* | extract identification markers |
| *rcs* | change RCS file attributes |
| *rcsclean* | remove unchanged working files (optional) |
| *rcsdiff* | compare revisions |
| *rcsfreeze* | record a configuration (optional) |
| *rcsmerge* | merge revisions |
| *rlog* | read log messages and other information in RCS files |

A synopsis of these commands appears in the Appendix.

**2.1 Automatic Identification**

RCS can stamp source and object code with special identification strings, similar to product and serial numbers. To obtain such identification, place the marker

*$Id$*

into the text of a revision, for instance inside a comment. The check-out operation will replace this marker with a string of the form

*$Id: filename revisionnumber date time author state locker $*

To format it, you can try

```
$ troff rcs.ms >rcs.ps
```

This assumes the use of *groff* or another flavour of *troff* that creates PostScript output (thus the name *rcs.ps* for the output fi le). If you do this, you get an output that looks like:

```
Besides the operations ci and co, RCS provides the following commands: tab(%); li l.
ident%extract identifi cation markers rcs%change RCS fi le attributes rcsclean%remove
unchanged working fi les (optional) rcsdiff%compare revisions rcsfreeze%record a confi gura-
tion (optional) rcsmerge%merge revisions rlog%read log messages and other information in
RCS fi les A synopsis of these commands appears in the Appendix. Automatic Identifi cation
RCS can stamp source and object code with special identifi cation strings, similar to product
and serial numbers. To obtain such identifi cation, place the marker Id into the text of a revi-
sion, for instance inside a comment. The check-out operation will replace this marker with a
string of the form Id: fi lename revisionnumber date time author state locker
```

Most of the text seems to be there, but it hasn't been formatted at all (well, it *has* been right justifi ed). What happened?

Almost every *troff* or *roff* input document uses some set of *macros*. You can defi ne your own macros in the source, of course, but over time a number of standard macro packages have evolved. They are stored in a directory called *tmac*. In the days of no confusion, this was */usr/lib/tmac*, but nowadays it might equally well be */usr/share/tmac* (for systems close to the System V.4 ABI—see Chapter 4, *Package configuration*, page 48, for more details) or */usr/local/groff/tmac* for GNU *roff*. The name is known to *troff* either by environment variables or by instinct (the path name is compiled into the program). *troff* loads specifi c macros if you specify the name of the fi le as an argument to the `-m` flag. For example, to specify the man page macros */usr/lib/tmac/an*, you would supply *troff* with the parameter `-man`. *man* makes more sense than *an*, so these macros are called the *man* macros. The names of other macro packages also usually grow an *m* at the beginning. Some systems change the base name of the macros from, say, */usr/lib/tmac/an* to */usr/lib/tmac/tmac.an*.

Most versions of *troff* supply the following macro packages:

- The *man* (*tmac/an*) and *mandoc* (*tmac/andoc*) packages are used to format man pages.
- The *mdoc* (*tmac/doc*) package is used to format hardcopy documents, including some man pages.
- The *mm* (*tmac/m*) macros, the so-called *memorandum* macros, are described in the documentation as macros to "format letters, reports, memoranda, papers, manuals and books". It doesn't describe what you shouldn't use them for.
- The *ms* (*tmac/s*) macros were the original macros supplied with the Seventh Edition. They are now claimed to be obsolescent, but you will see them again and again. This book was formatted with a modifi ed version of the *ms* macros.
- The *me* (*tmac/e*) macros are another, more recent set of macros which originated in Berkeley.

There is no sure-fi re way to tell which macros a fi le needs. Here are a couple of possibilities:

- The fi le name suffi x might give a hint. For example, our fi le is called *rcs.ms*, so there is a very good chance that it wants to be formatted with `-ms`.

- The program *grog*, which is part of *groff*, examines the source and guesses the kind of macro set. It is frequently wrong.
- The only other way is trial and error. There aren't that many different macro sets, so this might be a good solution.

In this case, our file name suggests that it should be formatted with the *ms* macros. Let's try that:

```
$ troff rcs.ms >rcs.ps
```

Now we get:

> Besides the operations *ci* and *co*, RCS provides the following commands:
> tab(%); li l. ident%extract identification markers rcs%change RCS file attributes rcsclean%remove unchanged working files (optional) rcsdiff%compare revisions rcs-freeze%record a configuration (optional) rcsmerge%merge revisions rlog%read log messages and other information in RCS files A synopsis of these commands appears in the Appendix.
>
> **2.1 Automatic Identification**
>
> RCS can stamp source and object code with special identification strings, similar to product and serial numbers. To obtain such identification, place the marker
>
> *$Id$*
>
> into the text of a revision, for instance inside a comment. The check-out operation will replace this marker with a string of the form
>
> *$Id: filename revisionnumber date time author state locker $*

Well, it doesn't look quite as bad, but it's still not where we want to be. What happened to that list of program names?

*troff* does not do all the work by itself. The tabular layout of the program names in this example is done by the preprocessor *tbl*, which handles tables. Before we let *troff* at the document, we need to pass it through *tbl*, which replaces the code

```
.TS
tab(%);
li l.
ident%extract identification markers
rcs%change RCS file attributes
rcsclean%remove unchanged working files (optional)
rcsdiff%compare revisions
rcsfreeze%record a configuration (optional)
rcsmerge%merge revisions
rlog%read log messages and other information in RCS files
.TE
```

with a couple of hundred lines of complicated and illegible *troff* instructions to build the table. To get the desired results, we need to enter:

```
$ tbl rcs.ms | troff -ms >rcs.ps
```

*nroff*, *troff* and *groff* use a number of preprocessors to perform special functions. They are:

- *soelim* replaces *.so* statements (which correspond to C *#include* statements) with the contents of the file to which the line refers. The *roff* programs do this too, of course, but the other preprocessors don't, so if the contents of one of the files is of interest to another preprocessor, you need to run *soelim* first.
- *refer* processes references.
- *pic* draws simple pictures.
- *tbl* formats data in tabular form.
- *eqn* formats equations.

Unless you *know* that the document you're formatting doesn't use any of these preprocessors, or formatting takes a very long time, it's easier to use them all. There are two possible ways to do this:

- You can pipe from one processor to the next. This is the standard way:

  ```
  $ soelim rcs.ms | refer | pic | tbl | eqn | troff -ms
  ```

  The *soelim* preprocessor reads in the document, and replaces any *.so* commands by the contents of the file to which they refer. It then passes the output to *refer*, which processes any textual references and passes it to *pic*, which processes any pictures it may find, and passes the result to *tbl*. *tbl* processes any tables and passes its result to *eqn*, which processes any equations before passing the result to *troff*.
- Some versions of *troff* invoke the preprocessors themselves if passed appropriate flags. For example, with *groff*:

*Table 7−1: Starting preprocessors from groff*

| Flag | Processor |
|---|---|
| `-e` | *eqn* |
| `-t` | *tbl* |
| `-p` | *pic* |
| `-s` | *soelim* |
| `-R` | *refer* |

Starting the preprocessors from *troff* not only has the advantage of involving less typing—it also ensures that the preprocessors are started in the correct sequence. Problems can arise if you run *eqn* before *tbl*, for example, when there are equations within tables. See *Typesetting tables with tbl* by Henry McGilton and Mary McNabb for further details.

## Other roff-related programs

As you can see, the *troff* system uses a large number of programs. Once they were relatively small, and this was the UNIX way. Now they are large, but there are still a lot of them. Apart from the programs we have already seen, you could encounter the GNU variants, which can optionally be installed with a name beginning in *g*—for example, GNU *eqn* may be installed as *geqn* if the system already has a different version of *eqn*. *indxbib* and *lookbib* (sometimes called *lkbib)* process bibliographic references, and are available in the *groff* package if you don't have them. *groff* also includes a number of other programs, such as *grops*, and *grotty*, which you don't normally need to invoke directly.

## Man pages

Almost from the beginning, UNIX had an *on-line manual*, traditionally called *man pages*. You can peruse man pages with the *man* program, or you can print them out as hardcopy documentation.

Traditionally, *man* pages are cryptic and formalized: they were introduced at a time when disk storage was expensive, so they are short, and they were intended as a reference for people who already understand the product. More and more, unfortunately, they are taking on the responsibility of being the sole source of documentation. They don't perform this task very well.

### man history

The UNIX *man* facility has had a long and varying history, and knowing it helps understand some of the strangenesses. The Seventh Edition of the Unix Programmer's Manual was divided into nine sections. Section 9, which contained the quick reference cards, has since atrophied. Traditionally, you refer to man pages by the name of the item to which they refer, followed by the section number in parentheses, so the man page for the C compiler would be called *cc(1)*. BSD systems have substantially retained the Seventh Edition structure, but System V has reorganized them. There are also differences of opinion about where individual man pages belong, so Table 7-2 can only be a guide:

*Table 7−2: UNIX manual sections*

| Seventh Edition Section | Contents | System V Section |
|---|---|---|
| 1 | Commands (programs) | 1 |
| 2 | System Calls (direct kernel interface) | 2 |
| 3 | Subroutines (library functions in user space) | 3 |
| 4 | Special fi les | 7, 4 |
| 5 | File Formats and Conventions | 4, 5 |
| 6 | Games | 6 |
| 7 | Macro Packages and Language Conventions | 7 |
| 8 | Maintenance | 1m |
| 9 | Quick Reference cards | |

What distinguished the UNIX manual from that of other systems was that it was designed to be kept online. Each of these sections, except for the quick reference cards, was stored in *nroff* format in a directory called */usr/man/man<section>*, where *<section>* was the section number. Each entry was (and is) called a *man page*, although nowadays some can run on for 100 pages or more.

The manual was stored in *nroff* format in order to be independent of the display hardware, and because formatting the whole manual took such a long time. For these reasons it was chosen to format pages on an individual basis when they were accessed, which made access to the manual slower and thus less attractive to use.

The speed problem was solved by saving the formatted copy of the man page in a second directory hierarchy, */usr/man/cat<section>*, the first time that the page was formatted. Subsequent accesses would then find the formatted page and display that more quickly.

This basic hierarchy has survived more or less intact to the present day. People have, of course, thought of ways to confuse it:

- As the manual got larger, it seemed reasonable to subdivide it further. Most users weren't interested in system administration functions, so some systems put them into a separate directory, such as */usr/man/cat1m*, or gave them a filename suffix such as *m*, so that the manual page for *shutdown* might end up being called */usr/man/cat1/shutdown.1m* or */usr/man/man1m/shutdown.1m* or something similar.
- Various commercial implementations reorganized the sequence of the sections in the printed manual, and reorganized the directories to coincide. For example, in System V the description of the file */etc/group* is in section 4, but in the Seventh Edition and BSD it is in section 5.
- Even without the uncertainty of which section to search for a command, it was evident that section numbers were not very informative. Some implementations, such as XENIX and some versions of System V, chose to replace the uninformative numbers with uninformative letters. For example, *ls(1)* becomes *ls(C)* in XENIX.
- Some *man* programs have lost the ability to format the man pages, so you need to format them before installation. You'll find this problem on systems where *nroff* is an add-on component.
- There is no longer a single directory where you can expect to put man pages: some System V versions put formatted man pages for users in a directory */usr/catman/u_man*, and man pages for programmers in */usr/catman/p_man*. Since most programmers are users, and the distinction between the use of the man pages is not always as clear as you would like, this means that *man* has to search two separate directory hierarchies for the man pages.
- As we saw in Chapter 4, *Package configuration*, page 48, System V.4 puts its man pages in */usr/share/man*. Many System V.4 systems require formatted man pages, and some, such as UnixWare, don't provide a man program at all.
- Many *man* programs accept compressed input, either formatted or non-formatted. For some reason, the *pack* program still survives here, but other versions of *man* also understand man pages compressed with *compress* or *gzip*. We looked at all of these programs

in Chapter 2, *Unpacking the goodies*, page 20.

- Different *man* programs place different interpretations on the suffix of the man page filename. They seldom document the meanings of the suffix.
- To keep up the tradition of incompatible man pages, BSD has changed the default macro set from *man* to *mdoc*. This means that older man page readers can't make any sense of unformatted BSD man pages.

This combination of affairs makes life difficult. For example, on my system I have a number of different man pages in different directories. The file names for the man pages for *printf*, which is both a command and a library function, are:

```
BSD printf command, formatted:
        /usr/share/man/cat1/printf.0
Solaris printf command, nroff:
        /pub/man/solaris-2.2/man1/printf.1
SVR4.2 printf command, formatted, compressed:
        /pub/man/svr4.2/cat1/printf.1.Z
BSD printf function, formatted:
        /usr/share/man/cat3/printf.0
Solaris 2.2 printf function, nroff, standard:
        /pub/man/solaris-2.2/man3/printf.3s
Solaris 2.2 printf function, nroff, BSD version:
        /pub/man/solaris-2.2/man3/printf.3b
SunOS 4.1.3 printf function, nroff:
        /pub/man/sunos-4.1.3/man3/printf.3v
SVR3 printf function, formatted, packed:
        /pub/man/catman/p_man/man3/printf.3s.z
SVR4.2 printf function, formatted, compressed:
        /pub/man/svr4.2/cat3/printf.3s.Z
SVR4.2 printf function, formatted, compressed, BSD version:
        /pub/man/svr4.2/cat3/printf.3b.Z
XENIX printf function, nroff, packed:
        /pub/man/xenix-2.3.2/man.S/printf.S.z
```

Most packages assume that unformatted man pages will be installed in */usr/man*. They usually accept that the path may be different, and some allow you to change the subdirectory and the file name suffix, but this is as far as they normally go.

This lack of standardization can cause such problems that many people just give up and don't bother to install the man pages. This is a pity—instead, why not install a *man* program that isn't as fussy? A number of alternatives are available, including one for System V.4 from Walnut Creek and a number on various Linux distributions.

## TeX

*TEX* is Donald Knuth's monument to the triumph of logic over convention. To quote Knuth's *The TEX* book,

> *Insiders pronounce the $\chi$ of TEX as a Greek chi, not as an 'x', so that TEX rhymes with the word blecchhh. It's the 'ch' sound in Scottish words like* loch *or German words like* ach*; it's a*

*Spanish 'j' and a Russian 'kh'. When you say it correctly to your computer, the terminal may become slightly moist.*

This is one of the more informative parts of *The TEX book*. It is, unfortunately, not a manual but a textbook, and most of the essential parts are hidden in exercises flagged "very difficult". If you just want to figure out how to format a TEX document, *Making TEX work*, by Norman Walsh, is a much better option.

If *troff* input looks like a fly having left an inkwell, TEX input resembles more the attempts of a drunken spider. Here's part of the file *plain.tex* which defines some of the things that any TEX macro package should be able to do:

```
\def\cases#1{\left\{\,\vcenter{\normalbaselines\m@th
    \ialign{$##\hfil$&\quad##\hfil\crcr#1\crcr}}\right.}
\def\matrix#1{\null\,\vcenter{\normalbaselines\m@th
    \ialign{\hfil$##$\hfil&&\quad\hfil$##$\hfil\crcr
      \mathstrut\crcr\noalign{\kern-\baselineskip}
      #1\crcr\mathstrut\crcr\noalign{\kern-\baselineskip}}}\,}
```

More than anywhere else in porting, it is good for your state of mind to steer clear of TEX internals. The assumptions on which the syntax is based differ markedly from those of other programming languages. For example, identifiers may not contain digits, and spaces are required only when the meaning would otherwise be ambiguous (to TEX, not to you), so the sequence `fontsize300` is in fact the identifier `fontsize` followed by the number `300`. On the other hand, it is almost impossible to find any good solid information in the documentation, so you could spend hours trying to solve a minor problem. I have been using TEX frequently for years, and I still find it the most frustrating program I have ever seen.*

Along with TEX, there are a couple of macro packages that have become so important that they are almost text processors in their own right:

- LATEX is a macro package that is not quite as painful as plain TEX, but also not as powerful. It is normally built as a separate program when installing TEX, using a technique of dumping a running program to an object file that we will examine in Chapter 21, *Object files and friends*, page 376.
- BIBTEX is an auxiliary program which, in conjuntion with LATEX, creates bibliographic references. Read all about it in *Making TEX* work. It usually takes three runs through the source files to create the correct auxiliary files and format the document correctly.
- *texinfo* is a GNU package that supplies both online and hardcopy documentation. It uses TEX to format the hardcopy documentation. We'll look at it along with GNU *info* in the next section.

* When I wrote this sentence, I wondered if I wasn't overstating the case. Mike Loukides, the author of *Programming with GNU Software*, reviewed the final draft and added a single word: *Amen*.

## GNU Info and Texinfo

It's unlikely that you'll break out in storms of enthusiasm about the documentation techniques we've looked at so far. The GNU project didn't, either, when they started, though their concerns were somewhat different:

- Man pages are straightforward, but the *man* program is relatively primitive. In particular, *man* does not provide a way to follow up on references in the man page.
- Man pages are intended to be stored on-line and thus tend to be cryptic. This makes them unsuited as hardcopy documentation. Making them longer and more detailed makes them less suited for online documentation.
- There is almost no link between man pages and hardcopy documentation, unless they happen to be the same thing for a particular package.
- Maintaining man pages *and* hardcopy documentation is double the work and opens you to the danger of omissions in one or the other document.

As in other areas, the GNU project started from scratch and came up with a third solution, *info*. This is a combined system of online and hardcopy documentation. Both forms of documentation are contained in the source file: you use *makeinfo* program to create *info* documents, which you read with the on-line browser *info*, and you use *TEX* and the *texinfo* macro set are used to format the documentation for printing.

*info* is a menu-driven, tree-structured online browser. You can follow in-text references and then return to the original text. *info* is available both as a stand-alone program and as an *emacs* macro.

If you have a package that supplies documentation in *info* format, you should use it. Even if some GNU programs, such as *gcc* and *emacs*, have both *info* and man pages, the *info* is much more detailled.

Running *texinfo* is straightforward: run *TEX*. The document reads in the file *texinfo.tex*, and about the only problem you are likely to encounter is if it doesn't find this file.

## The World-Wide Web

The World-Wide Web (WWW) is not primarily a program documentation system, but it has a number of properties which make it suitable as a manual browser: as a result of the proliferation of the Internet, it is well known and generally available, it supplies a transparent cross-reference system, and the user interface is easier to understand. It's likely that it will gain importance in the years to come. Hopefully it will do this without causing as much confusion as its predecessors.