Chapter 3

Finding Help

3.1 Introduction

There's an old joke that goes something like this: "Programmers shouldn't document their code. It should be as difficult to use as it was to write." Intentional or not, the fact is that open source code often lives up to these expectations.

Although most tools do a good job of providing documentation and making it easy to find, many don't. Sometimes, you just need to know where to look.

3.2 Online Help Tools

An important innovation of UNIX was the use of online[1] help. This was a welcome alternative to keeping a shelf full of printed manuals at your disposal, which was

1. The term *online* here refers to documentation in electronic format as opposed to hard copy, not the modern definition that implies the Internet.

also common at the time. The `man` program was, and still is, the main interface for online help. The GNU project, looking for a more flexible online format, invented the Texinfo format, which generates the documentation used by the `info` program. Some tools may neglect to provide `man` pages or `info` pages, and instead provide some plain text or other format tucked away in the distribution, just waiting for you to find it.

Although `man` and `info` are the formats of choice for command-line tools, many GUIs choose to use their help menus as the only source of online documentation.

3.2.1 The man Page

The idea of viewing documentation in a text window may seem archaic in these days of HTML and PDF, but there are advantages to reading documentation in a terminal window. This is especially true if, like many Linux developers, that is where you do most of your work. Getting the information from a `man` page is much quicker than opening a browser or a PDF viewer. When you want answers fast, the `man` page is what you want.

`man` pages have a concise format that is meant to be read in a linear fashion. Simple tools can get by with a single `man` page as their only source of documentation; more complicated tools and programming libraries rely on a suite of `man` pages for documentation. Perl even created its own manual section containing hundreds of pages covering various aspects of Perl.

In the past, `man` pages were limited to ASCII (or ISO-8859) characters, which limited the selection of languages in which they could be written. This was largely due to the limitations of the text terminals used to display them. In the old days, text terminals were capable of handling only 7- or 8-bit character encodings, and memory constraints limited the number of available fonts. Those dumb terminals are largely a thing of the past. GUI-based terminals such as `xterm` are capable of handling many fonts and character encodings with no issues. The tools that create `man` pages allow more of a selection of character encodings, such as UTF-8, so `man` pages can now be written in any language.

`man` pages are written in an ancient markup language called `troff`. The `troff` text-formatting language, although quite old, is still very powerful. Like HTML, `troff` output is formatted to fit the device on which it is being presented. So in addition to viewing in a text window, the output can be formatted for printout or

transformed into HTML or PDF. To format a `man` page for PostScript printout, for example, you can use the `-t` option as follows:

```
$ man -t man | lp -Pps
```

You can pipe this output to any tool that understands PostScript to manipulate the output any way you like.

There are two flavors of `man` in use in various distributions. Red Hat and many other RPM-based distributions use the traditional `man` program, whereas Debian-based distributions use a package named `man-db`. The difference between the two is primarily in the database that is used to index and catalog the `man` pages. The `man-db` approach has some advantages over the traditional `man` page database, but for the most part, both sets of tools behave the same way.

3.2.2 man Organization

The Linux manual is broken up into sections. This follows the Filesystem Hierarchy Standard,[2] which, among other things, specifies the contents of each `man` section. A summary of these sections is listed in Table 3-1.

TABLE 3-1 Linux Manual Sections

Section	Description
1	User commands available from the shell
2	System calls available to programs via library functions
3	Library functions available to programs
4	Devices available in `/dev` directory
5	Miscellaneous system files (for example, `/etc`)
6	Games, if any
7	Miscellaneous information
8	Commands available to administrators

2. www.pathname.com/fhs

The division of the manual into sections allows man pages to avoid name clashes. A command named sync is documented in section 1, for example, and a function named sync is documented in section 2. Both man pages are named sync. If they weren't documented in different sections, the system would have to resort to some odd naming conventions to distinguish between the two. Occasionally, you have to jump through a few hoops to get the man page you want. If you want to see the man page for the sync command, the following will do on most systems:

```
$ man sync
```

If you want the man page for the sync *function*, you need to know that sync is a system call and that it is documented in section 2. To look at this specific man page, you specify the section number before the page name, as follows:

```
$ man 2 sync
```

Another confounding factor is that some distributions take the liberty of creating their own sections. Perl is one example mentioned earlier. Another is section 3p, which comes with Fedora. This contains POSIX functions (hence, the p). This section also contains a man page for the sync function, which is virtually identical to the man page in section 2. If you are not sure, and you just want to see all the man pages the system knows about, you can type

```
$ man -a sync
```

This brings up the man pages in order, and each time you press q to exit the man page, you are presented with the next matching man page.

The convention for referring to a man page in a specific section is to put the section number in parentheses following the page name. The sync function in section 2, for example, would be referenced as sync(2), whereas the sync function in section 3p would be referenced as sync(3p). This notational convention is used throughout man pages, and I follow this convention in the footnotes of this book.

You might expect that when the section is not specified, each section is searched in sequence. But most distributions choose to search for commands before functions. That means that man searches section 8 (System Administration Commands) before section 3 (Programming Libraries). If you are a system administrator (and who isn't?), that makes sense. But as a programmer, if I am writing a socket program and need the man page for the accept function located in section 3, typing man accept will give me the man page for the accept command from the Common

UNIX Printer System package documented in section 8. If you are like me, you probably don't remember which section is which off the top of your head. That is where the `whatis` command comes in handy:

```
$ whatis accept
accept (8)           - accept/reject jobs sent to a destination
accept (2)           - accept a connection on a socket
```

This shows you the `man` pages in the order in which they are found. In this case, it's apparent that section 8 is searched before section 2. It also shows you that the `man` page you are looking for is in section 2.

The `man` page search order is determined in a system configuration file, which varies based on installation. Fedora and Ubuntu (Debian-based) use `/etc/man.config`, whereas Knoppix (also Debian-based) uses `/etc/manpath.config`. The `mandb` version allows you to override the settings for yourself in `~/.manpath`. The traditional package, however, does not allow you to override the defaults without a command-line option or environment variable.

The command-line manual tools leave a few things to be desired. It would be nice, for example, to browse section 3 of the manual. Unfortunately, the command-line tools do not allow casual browsing of the manual; neither do they allow you simply to list all the entries in a particular section. The old `xman` tool, which was part of the original X11 distribution, would let you browse by section, but most newer distributions no longer include this tool. One tool that will let you browse the `man` pages by section is part of the KDE project. The `khelpcenter` program allows you to browse not only KDE documentation, but the plain old `man` pages as well, as you can see in Figure 3-1.

You might be asking, "What about GNOME?" As of this writing, the `gnome-help` tool does not support browsing of the `man` pages.

3.2.3 Searching the man Pages: apropos

If you can't browse the manuals, the next-best thing you can do is search them. There are two basic tools for this purpose: `apropos` and `whatis`. `apropos` is an unusual name for a UNIX command, not just because it has more than three letters, but also because this is a word most native English speakers haven't seen since their last high-school vocabulary quiz. The word *apropos* means *relevant*, which is the idea behind the command. You give a keyword, and it comes up with relevant results (ideally).

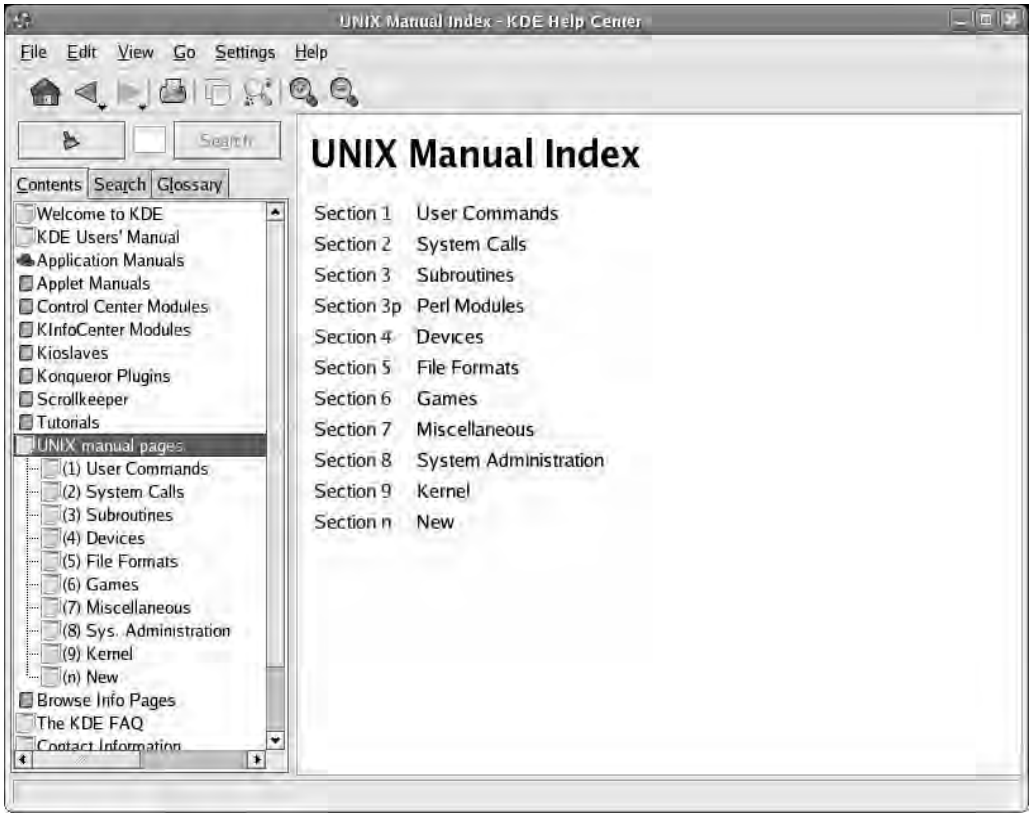

FIGURE 3-1 Using `khelpcenter` to Browse the `man` Sections

Unfortunately, the `apropos` command searches only the `NAME` section of the manual entries, which contains only a brief one-line description of the topic. So if your keyword doesn't show up in the `NAME` section, `apropos` won't help. Only one keyword at a time is searched. So if you specify two keywords, you get all the matches for the first keyword, followed by all the matches for the second keyword. By default, matching occurs anywhere in the text, so the word `man` matches the words *manual, command,* etc., as well as the word `man`. This is not a search engine. The results you get may be anything but apropos.

You also have to be careful that your keyword is not too restrictive. Searching for the word *compression,* for example, ought to show you something about compression utilities, but when I type `apropos compression`, the results are missing two of the most popular ones: `bzip2` and `gzip`. Here are the results from Fedora Core 4, for example:

```
$ apropos compression
Compress::Zlib       (3pm)  - Interface to zlib ...
pbmtopsg3            (1)  - convert PBM ...
SSL_COMP_add_compression_method (3ssl)  - handle ...
zlib                 (3)  - compression/decompression ...
```

The problem is that the term *compression* does not match the words *compress* and *compressor*, which, as it turns out, are the words used to describe `gzip` and `bzip2`, respectively. So our keyword search comes up with nothing. You can see this summary for yourself with the `whatis` command as follows:

```
$ whatis gzip bzip2
gzip (1)             - compress or expand files
bzip2 (1)            - a block-sorting file compressor, v1.0.2
```

The traditional version of `apropos`, found in Red Hat distributions, takes no options. The `mandb` version of `apropos`, found in Debian distributions, takes several, including options to restrict matches to exact matches or whole words. Both versions allow you to use regular expressions.[3] For example:

```
$ apropos 'mag[tn]'
mt (1)               - control magnetic tape drive operation
mt-gnu (1)           - control magnetic tape drive operation
rmt (8)              - remote magtape protocol module
rmt-tar (8)          - remote magtape protocol module
xmag (1x)            - magnify parts of the screen
```

The expression `mag[tn]` matches *magtape* and *magnify*, so your search results include anything that has either word. The `man-db` version of `apropos` found in Debian distributions allows you to restrict output to exact matches with the `-e` option. This allows you, for example, to look for the word *compress* without also matching the word *compressor*. Table 3-2 shows a summary of features by package.

---

3. If you need a primer on regular expressions, see `regex(7)` in the manual.

TABLE 3-2 Search Features by Package

Search Features	Traditional	mandb
Regular expressions	Yes	Yes
Exact matches		Yes
Brute-force matching	Yes	

The traditional version of `man` allows you to do a brute-force search for keywords by looking at every word in every single `man` page. The `-K` option (note the capitalization) of `man` does this and is guaranteed to be slow even on the fastest machines. When a thorough search is in order, this is the best you can do.

3.2.4 Getting the Right man Page: whatis

We saw how different sections can have `man` pages with the same name, which can cause you to get the wrong `man` page when you don't know what section to look in. If you want to look up the usage of the `readdir` function, for example, typing `man readdir` will take you to section 2 of the manual, which says:

```
This is not the function you are interested in.  Look at readdir(3) for the
POSIX conforming C library interface.  This page documents the bare kernel
system call interface, which can change, and which is superseded by getdents(2).
```

Luckily, in this case the `man` page is helpful enough to let you know that you are looking at a system call and not a POSIX function. It even tells you what section to look in. In the more likely event that the `man` page is not so helpful, the `whatis` command can help:

```
$ whatis readdir
readdir              (2)  - read directory entry
readdir              (3)  - read a directory
```

Because section 3 contains the `man` page we are looking for, we can then specify the correct `man` page as follows:

```
$ man 3 readdir
```

The traditional version of `whatis` takes no arguments, so matching is permissive. The `mandb` version allows regular expressions and shell-style wildcard matching. If

you can't remember a command but remember that it ends with `zip`, for example, you can try the following search:

```
$ whatis -w "*zip"
funzip (1)           - filter for extracting from a ZIP ..
gunzip (1)           - compress or expand files
gzip (1)             - compress or expand files
unzip (1)            - list, test and extract compressed ...
zip (1)              - package and compress (archive) files
```

A less elegant way to get the `man` page you are looking for is to read all of them until you find the one you like. This is done with the `-a` option to `man`, which says, "Show me all the `man` pages that match." The sections are presented in the order determined by the local configuration, which can get tedious. On my Fedora Core 4 installation, for example, `man -a read` brings up eight `man` pages. Sifting through eight `man` pages might provide some incentive for you to remember the section number for next time.

3.2.5 Things to Look for in the man Page

A Linux `man` page follows the conventions that are documented in section 7 of the manual.[4] A minimal `man` page has a `NAME` section, which consists of the name of the program or topic being documented, followed by a brief description. This is the only text that is searched by the `apropos` command.

Most `man` pages consist of more than just a `NAME` section, of course. There are several standard sections defined by convention, but programmers are free to define additional sections as appropriate.

The `SEE ALSO` section is the closest thing you will get to cross-referenced documentation in a `man` page. Don't be surprised if this refers you to a `man` page that does not exist on your system (or perhaps anywhere else). Occasionally, some older `man` pages refer to programs that don't exist anymore. It's also possible that the `man` page refers to a program that just isn't installed on your system. You should always look at the `SEE ALSO` section and try to find the cross references. You can learn a lot more than you would by reading one `man` page alone.

The `ENVIRONMENT` section provides valuable information about how environment variables affect the program's behavior. Usually, this section covers locale

4. See `man(7)`.

issues, but occasionally, there are some nice shortcuts that can be encapsulated in an environment variable. Some programs, for example, allow you to put lengthy command-line options in an environment variable to save you the trouble of typing them every time. The ENVIRONMENT section is always worth a peek.

Another section worth looking at is the CONFORMING TO section, which tells you what standards apply to the command or function. This is important when you are writing portable programs. There are many examples of multiple functions that do the same thing in Linux. This conformance should be the tie-breaker when it comes to deciding which one to use. The bcopy function does the same thing that memcpy does, for example, but if you check the CONFORMING TO section for bcopy, you will see that it conforms to 4.3BSD. The memcpy function also conforms to 4.3BSD, but it is also part of the ISO C standard (ISO 9899), so memcpy should be preferred for portability.

Near the bottom of the man page, you may see a BUGS section. Perhaps this is a misnomer, because some people say that the only difference between a bug and a feature is the documentation. Regardless, the BUGS section usually documents design limitations of the program or features that aren't fully functional. Sometimes, it contains a "to do" list to let you know what features are planned. It's an optional section, but if someone took the time to write the section, it behooves you to read it.

3.2.6 Some Recommended man Pages

Every section of the manual has an intro page that can be helpful if you forget which section is which. Though the intro pages themselves don't have much useful information,[5] they can serve as a quick reminder about section names, as follows:

```
$ whatis intro
intro                (1)  - Introduction to user commands
intro                (2)  - Introduction to system calls
intro                (3)  - Introduction to library functions
intro                (4)  - Introduction to special files
intro                (5)  - Introduction to file formats
intro                (6)  - Introduction to games
intro                (7)  - Introduction to conventions and miscellany
                            section
intro                (8)  - Introduction to administration and
                            privileged commands
```

5. intro(2) is one exception here. There is a useful introduction to system calls.

If the idea of casually reading the manual seems weird, that probably means you have never spent much time reading it. There are several pages with useful information and tutorials that you would never find unless you looked for them. Many of these are in section 7, which is probably the least-read section of the manual. Table 3-3 lists a few of the selections from section 7 you may want to read.

TABLE 3-3 Recommended Reading from Section 7 of the Manual

Name	Description
`ascii(7)`	A short, handy page that simply lists ASCII codes in tables—very handy when you need to know silly details like the octal constant for Ctrl+G.
`boot(7)`	A nice overview of the kernel boot sequence. If you are trying to build distribution from scratch or just want to learn how Linux boots, this is a good read.
`bootparam(7)`	A nice summary of the low-level options that the kernel can take on the command line. This list is long, and it is most assuredly not complete, but it is very informative.
`charsets(7)`	A nice overview of character sets used in Linux, with a brief overview of the features of each. If you work on internationalized programs (i18n), this is a must read.
`hier(7)`	A good overview of the Filesystem Hierarchy Standard, which describes the conventions used to lay out the directories in a Linux system. It answers burning questions like "What's the difference between `/bin` and `/usr/bin`?" If you are developing an application for distribution, read this before you decide where to install your files.
`man(7)`	So you want to write a `man` page? This section tells you how, giving you just enough `troff` to get by as well as an explanation of the conventions used in Linux `man` pages.
`operator(7)`	Lists the C operators and their precedence. If you hate using extra parentheses in your code, you should study this `man` page. After reading this page, you should be able to find the bug in the following C statement: `if ( 1 & 2 == 2 ) printf("bit one is set\n");`

continues

TABLE 3-3 *Continued*

Name	Description
`regex(7)`	An introduction to regular expressions, which is a topic every programmer should master.
`suffixes(7)`	In Linux, file suffixes are conventions for the user. The OS relies on more reliable techniques to determine a file type. This `man` page describes many of the known conventional suffixes in use. It is not complete but can be helpful.
`units(7)`	Lists standard unit multipliers defined by SI.[6] Did you know that according to SI rules, a megabyte is actually 1,000,000 bytes and not 1,048,576 bytes (2^{20} bytes)? Disk drive manufacturers know this, and you should too. We software developers tend to use the prefixes for decimal multipliers when we mean to use the binary prefixes. The SI defines unique prefixes for such binary values that no one uses. Perhaps we should. By the way, 1,048,576 bytes is properly called a *mebibyte* (abbreviated MiB). Although this terminology is not likely to catch on any time soon, you must be aware of the potential for confusion when you are talking requirements with a scientist, hardware engineer, or disk drive vendor. There's also some nifty trivia in here, such as the prefix for a million billion billion. Impress your friends.
`uri(7), url(7), urn(7)`	Three keys for the same `man` page, describing the components of a URL and what they mean. You may think you know all the things that can go into a URL, but check this page out and see whether you learn anything.

Remember that all the `man` pages listed in Table 3-3 are in section 7 and that some of these names clash with those in other sections. Be sure you specify the section in order to get the correct `man` page.

6. *SI* is an abbreviation for *Système International d'unités*, the international standard for scientific measurements.

3.2.7 GNU info

When you're looking for documentation for GNU tools, the `info` program is the preferred tool. Very often, GNU `man` pages contain an abridged version of the documentation that refers you to the `info` program for more details.

The fact that GNU chose to use a unique tool to document its tools is something of a nuisance to UNIX folks, who are used to `man` pages, but `info` provides several features that you don't get in a `man` page. For one thing, `info` files are hyperlinked and heavily cross referenced. So when you browse the documentation for `ssh`, for example, it will refer to other relevant tools, such as `ssh-keygen`. You can follow the link in the documentation to go to the documentation for `ssh-keygen` and go back again, just like on a Web page. Another feature of `info` pages is that they are indexed so that you can locate relevant documentation more effectively than on a `man` page.

GNU `info` pages are written in a markup language called Texinfo. Like `troff`, Texinfo predates HTML and XML but is built upon earlier work. Texinfo evolved from the TeX formatting language (still in wide use today) and another project called scribe, which lives on today as scheme scribe. The goal of Texinfo was to mark up content, not format, which makes Texinfo documents easier to write from scratch than those in some other markup languages. Like `nroff`, Texinfo format can be translated into hard copy, HTML, or plain text.

Although Texinfo is a very flexible format, most binary distributions don't include Texinfo source files. Normally, a binary distribution ships with specially processed text files that are suited only for input to the `info` viewer. These files are created from Texinfo source with the `makeinfo` program. Because they do not contain any typesetting information, they don't produce the best hard copy. If you want to translate the documentation into hard copy, or if you are looking to create PDF or HTML output, you should locate the Texinfo source. It usually can be found in the source distribution or on the project's home page.

3.2.8 Viewing info Pages

The GNU `info` system is tightly integrated with Emacs, the flagship GNU text editor. Figure 3-2 shows what `info` looks like in the Emacs editor.

If you are not an Emacs user, there are a couple of alternatives. One of these is the text-based `info` program. People who are used to `man` pages have a hard time

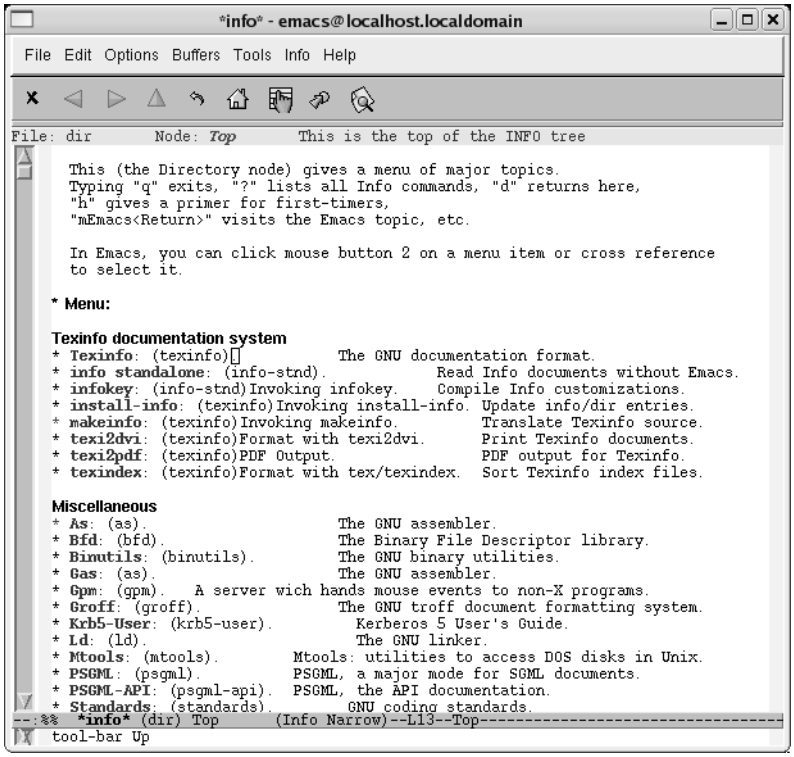

FIGURE 3-2 Using Emacs to View `info` Pages

using the `info` browser, mainly because it does not behave anything like the `less` pager used by `man`. This can be very irritating when you have to transition between `man` pages and `info` pages. Programmers who use the `vi` text editor are comfortable with the `less` program because its key bindings are very similar to those of `vi`. The fact is that `info` borrows heavily from Emacs and behaves much like it. `info` provides `vi` key bindings for Vi users with the `--vi-keys` option, but it doesn't behave much like `vi` or `less`.

So you want to use the `info` program? You can start with the `man` page, which covers only the command-line options. If you want more in-depth information, you need to look at the `info` page—but not so fast. If you type `info info`, you get information about the `info` format, not the `info` program. For that, you have to

type `info info-stnd`, which will give you the correct documentation. The `man` page ought to tell you that, but for some reason, it doesn't.

So you actually like the `info` program? You can use it as your preferred tool to look at both `man` pages and `info` pages. By default, `info` will try to find the information in `info` format and fall back to a `man` page if it does not.

There is another text-based alternative for programmers who don't use Emacs and don't care for `info`: the `pinfo` program.[7] Users who prefer `man` pages will have an easier time learning to use it. Like `info`, this is a text-based program, but it uses more advanced terminal capabilities, including color highlighting for hyperlinks. Perhaps most important, it uses `vi` keys for cursor movement.

In the realm of GUI tools are some more choices for viewing `info` pages. Some Web browsers will accept a URL of the form `info:topic`. In a GNOME system, this simply launches the `gnome-help` browser, which as of this writing does not support `man` or `info` pages. How rude!

A KDE browser sends you to `khelpcenter`, which is an excellent browser for `info` pages. You can see for yourself in Figure 3-3.

An `info` document is usually broken into sections, which makes it easier to read but can make finding exactly what you are looking for tedious. If you know the exact section you are looking for, you can bring it up with a lengthy command line like the following:

```
$ info "(make)Quick Reference"
```

There's nothing quick about having to type all that, but if it's something you refer to often, you can use a shell alias or a script to save typing. Both `pinfo` and `info` allow you to jump to specific sections with this syntax, but as of this writing, `khelpcenter` does not. If what you want is a single continuous stream of output, like a `man` page, you can coerce the `info` program to do so as follows:

```
$ info --subnodes some-topic | less
```

The `--subnodes` option forces `info` to dump the entire contents of the document to `stdout`. Piping the output to `less` gives it the look and feel of a very long `man` page.

---

7. http://pinfo.alioth.debian.org

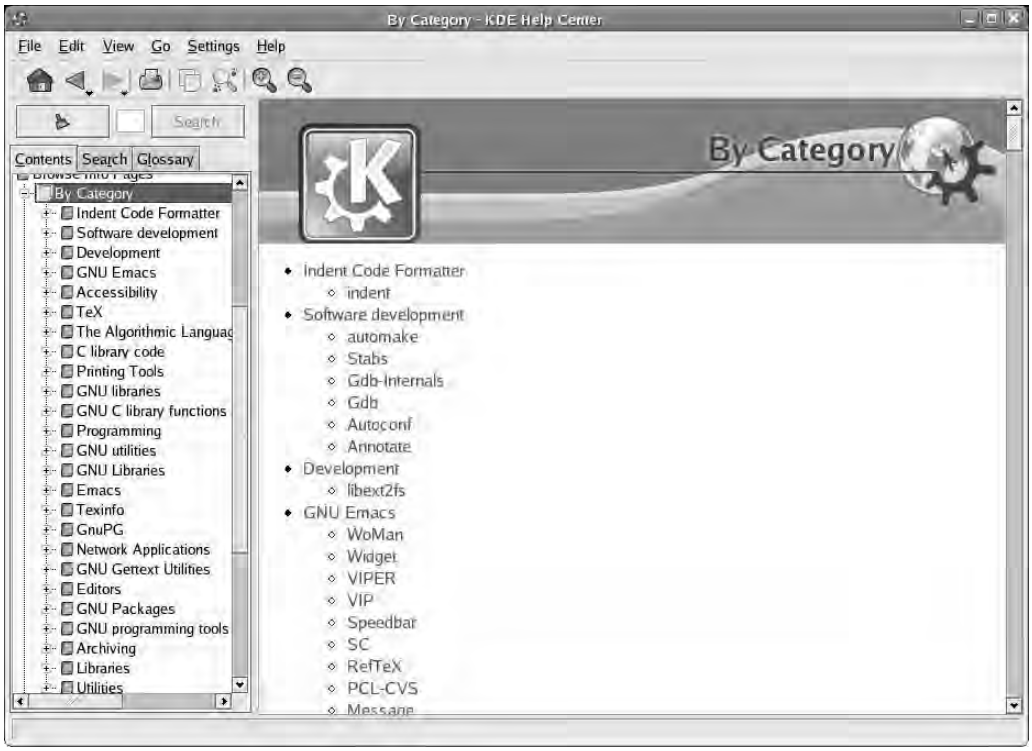

FIGURE 3-3 Using `khelpcenter` to View `info` Pages

3.2.9 Searching info Pages

You can search `info` pages like `man` pages by using the `--apropos` option to `info`. Each `info` file has an index, which can be searched with the `--apropos` option, so your search is more likely to produce output with `info` than with the `apropos` command used for `man` pages. Like `man`, the search is case insensitive and matches anywhere in the text. Unlike with `man`, you cannot use a regular expression. So although searching an index is likely to get you more relevant results, this benefit is largely offset by the lack of any filtering via regular expressions or word-based searches.

`info` also has a `-w` option that behaves like the `whereis` command, but because there are relatively few `info` files, you are unlikely to encounter the frequent name clashes that you often can with `man`.

3.2.10 Recommended info Pages

Although most GNU tools have only terse `man` pages that refer you to the `info` documentation, GNU has made an effort to provide comprehensive `man` pages for some tools, such as `gcc`. Even these, however, are outdone by their Texinfo counterparts. The `man` page for `gcc` 3.4.4, for example, is around 54,000 words, whereas the `info` page comes in at nearly 158,000 words. The `info` file contains much more background and historical information, whereas the `man` page is strictly business.

When you have some time to read and learn, several topics in the `info` pages are worth a read. Table 3-4 lists some selected topics for you to explore in the `info` pages.

Although `info` and `man` cover a large number of tools, there are many more ways to document code.

TABLE 3-4 Recommended Reading

info Page	Comments
`coreutils`	This is some very informative light reading. It contains many of the sundry two- and three-letter commands that UNIX programmers know and love but organizes them by function. There is even a reprint of an article that Arnold Robbins wrote for *Linux Journal* in the node named "Opening the software toolbox."
`cpp`	This probably is one of the most misunderstood tools that every C programmer uses. The `man` page leaves a great deal to be desired. Given that it's a complex topic, the `info` format is preferred.
`gcc`	Provides much more information than the `gcc man` page, including implementation details, and is well organized for online reading.
`ld`	A behind-the-scenes tool that every developer uses, but developers probably don't have a clue how it works. Again, this is a complex topic that is well suited to the `info` format.
`libc`	An in-depth reference to the C standard library, with lots of background information.

3.2.11 Desktop Help Tools

In the Linux world, there are two major players in the desktop environment: GNOME and KDE. Although each desktop environment has its fans and detractors, no desktop environment would be complete without online help. GNOME's help system is provided by the `gnome-help` command (also known as `yelp`) and as of version 2.10.0 is focused exclusively on GNOME tools. What help it provides usually is not relevant to programmers. Although `gnome-help` hasn't much to offer in the way of content, it still has its uses. It makes an excellent lightweight browser for a simple HTML file, such as a `README` file written in HTML. Firing up Mozilla or Firefox to view a simple HTML file is like using an aircraft carrier to go waterskiing. Besides, who wants to do all that typing? Mozilla and Firefox require you to type a fully qualified URL as follows:

```
$ firefox file:///usr/share/doc/someproject/README.html
```

Remember, that's three slashes after the `file:` (and the number of slashes shall be three). `yelp` is smart enough to find files in the current working directory, and it starts up much faster than Firefox or Mozilla. So in the very likely event that you happen to be working in the directory where your documentation resides, you can use a refreshingly simple command such as:

```
$ yelp README.html
```

Because it has no plug-ins to speak of, `yelp` can pull up a simple HTML document almost as fast as `man` can pull up a `man` page.

Just like `yelp`, `khelpcenter` makes an excellent lightweight Web browser for simple HTML files. As a bonus, `khelpcenter` provides a much less myopic view of the tools than `yelp`. Although `khelpcenter` is *the* source of information on KDE tools, it also allows you to view `man` pages and GNU `info` pages. The `info` pages are hyperlinked, just as expected, but less expected is the fact that `man` pages are hyperlinked as well. Scroll down to the `SEE ALSO` section of a `man` page and click any of the cross-referenced `man` pages to view that document. This is a nice feature. `khelpcenter` also provides search capability, as well as a glossary. The search is a little flaky, and the glossary is sparsely populated, but it is sure to get better as the tool matures.

3.3 Other Places to Look

If you are working with a new project, chances are that it has little or no documentation in the form of a `man` or `info` page. Documentation may exist but may

be hard to find. Things like README files have a way of getting lost in the packaging when a project is packaged in a binary distribution. Chances are that the help is somewhere; it's just up to you to find it.

3.3.1 /usr/share/doc

This subdirectory is part of the Filesystem Hierarchy Standard and is where many man pages and info pages are stored. It is also the preferred place to store the electronic equivalent of the writing on the back of a napkin that we call the README file.

A binary distribution typically includes a subdirectory named after each project, where it may place all sorts of sundry information. Typically, you will find release notes, change logs, copyright information, and some sort of README file.

Occasionally, developers distribute only plain text or HTML documentation without any man pages. This is the place you will usually find it. You might even find Texinfo source or some other markup, like DocBook.

Debian provides the doc-linux package, which contains many of the HOWTO documents from The Linux Documentation Project[8] (discussed later in this chapter) stored in /usr/share/doc. These files are not coupled with any installed packages; they're documentation for the sake for documentation.

3.3.2 Cross Referencing and Indexing

Cross referencing turns your documentation from a dictionary into a thesaurus. A dictionary shows you only what words you explicitly look for, whereas a thesaurus leads you to words you may not have thought of. Likewise, cross-referencing Linux documentation is important, because there is usually more than one way to do what you want, and you may not know it unless you check the cross references. Linux documentation does a fairly good job of referring to other tools despite the fact that the documentation is accumulated from many diverse sources.

For man pages, the cross-reference information can be found in the SEE ALSO section. Suppose that you are setting up a DHCP server, and you look at the dhcpd man page. This is the logical place to look, because dhcpd is the name of the DHCP daemon. If you don't look at the cross references, you may miss the dhcpd.conf page, which has more vital information. Many complex tools and servers are

8. www.tldp.org

documented by more than one man page, and these elements are cross referenced in the SEE ALSO section. A typical man page cross reference looks like the following:

```
SEE ALSO
       dhclient(8), dhcrelay(8), dhcpd.conf(5), dhcpd.leases(5)
```

I described the convention for referencing man pages earlier. You can see here that the man pages themselves use this notation.

Another vital tool for cross referencing man pages is the apropos command, which I have already discussed. Beware that apropos works only when the database is indexed. This normally is done for you with a fresh installation, but as you accumulate new software with new man pages and remove others, the index gets out of sync. As a result, relevant documentation may not show up in an apropos search, or you may get hits for software that is no longer installed. To fix this problem, you need to run the appropriate indexer for the man installation. The traditional man tool uses a tool called makewhatis that is usually run as a cron job. This is a fairly time-consuming process, as the tool has to visit every man page in the system. If you have a laptop that you power up and down often, this job may never get a chance to run. When it does, you'll probably want to kill it if you are running on batteries.

The mandb version of man uses a tool named (appropriately enough) mandb. One advantage of the mandb package is that indexing with the mandb program takes much less time than makewhatis. This, too, typically is set up as a cron job. info files do a fairly good job of cross-referencing other info files via hyperlinks. The info tool also has an apropos-like function with the --apropos option. Although it is not as flexible as the apropos command, it is still useful.

GNOME and KDE use HTML and XML extensively to document their tools, and these usually are cross referenced well. Unfortunately, there is no universally accepted convention for storing and indexing HTML files to document text-based commands. These files can refer you to a man page, but they can't link to it. So when you encounter an HTML file for a command-line tool, it is not likely to have hyperlinks to locally installed documentation. Links in these HTML files are likely to point to other HTML files in the same project or to sources on the Web.

3.3.3 Package Queries

In Chapter 1, I discuss in detail how to use packages. Here's how you can put that knowledge to use. For starters, you may want just the basic information about a package. This is useful in situations when you are wondering what a particular

command does. Suppose that you notice the program named `diffstat` in `/usr/bin` and wonder what it does. Perhaps you tried `man diffstat` and got nothing. The next thing you should try is a basic query, such as:

```
$ rpm -qf /usr/bin/diffstat
diffstat-1.38-2
```

This query tells you the name of the package that installed `/usr/bin/diffstat`. In this case, it reports that you have `diffstat-1.38-2` installed. Then you could query the package information to get a basic description of the package that this tool came from, such as:

```
$ rpm -qi diffstat
...
Summary     : A utility which provides statistics based on the output
              of diff.
Description : ...
```

Sometimes, this is enough to tell you what you need to know, but now that you know the name of the package, you can query the package contents to see whether you can glean any other information from it. In the list of installed files, you should look for `README` files or HTML documents that might give you more information. You might be on the lookout for misplaced `man` pages as well. These may be located in some unconventional places that you may not have checked otherwise. In the case of `diffstat-1.38-2`, someone put the `man` page in the wrong place:

```
$ rpm -ql diffstat
/usr/bin/diffstat
/usr/share/man/man1/man1
/usr/share/man/man1/man1/diffstat.1.gz
```

The packager mistakenly created an extra `man1` subdirectory under the `man1` subdirectory, so that the `man` command doesn't find it. Although this problem was fixed in a later version of `diffstat`, it is a useful illustration of how package queries can be helpful. Packagers are humans, too. In case you are wondering, you could still read this `man` page with the following command:

```
$ man /usr/share/man/man1/man1/diffstat.1.gz
```

In most distributions, when a `man` page cannot be found, you get nothing. In a Debian-based distribution you may encounter `undocumented(7)`, which is a generic `man` page describing some of the same topics covered in this section.

3.4 Documentation Formats

Linux documentation comes in many formats. Although plain text is common, several markup languages have been used over the years to produce prettier printouts or more browser-friendly output. One of the first attempts was `troff`, which is the language used to create `man` pages. `troff` generates output that can be typeset or viewed in a terminal window. Other formats such as LaTeX were designed with typeset printout in mind, but they are capable of producing browsable documentation. The preferred format of GNU, Texinfo, is designed to produce hypertext output that can be browsed or printed. Finally, there's the ubiquitous HTML which is designed exclusively for the browser. Which one are you most likely to encounter? All of them.

3.4.1 TeX/LaTeX/DVI

TeX and LaTeX markup go back a long way in UNIX history. LaTeX is an extension of the TeX markup system. Most TeX documents these days are actually LaTeX, but the two names are often used interchangeably. Primarily intended for hard-copy printout, TeX has extensive support for the special typesetting requirements of research papers that contain mathematical symbols and formulas. TeX made this easy for authors at a time when word processors were in their infancy, thereby creating a niche for itself among researchers. Despite advances in conventional word processing applications, TeX is still widely used today.

The native output format of TeX is called DVI (for Device Independent). By itself, it can be viewed using `xdvi`, `kdvi`, or `evince`, but the intended use of DVI was as an intermediate format. As a result, there are many mature tools to convert DVI files to any format under the sun. Many open source applications prefer to take advantage of this rather than reinvent the wheel, so they provide DVI or TeX output that can be manipulated further. Some tools include compressed DVI files with the binary distribution in `/usr/share/doc`. The Debian package of `gdb`, for example, includes a nifty reference card in DVI format.

Both TeX and LaTeX use the `.tex` extension for source files, which can be confusing. Documents written for TeX may not compile with the `latex` command, in which case you should try the `tex` command. In many cases, a LaTeX document consists of more than one source file, so make sure that you have all the source files before you try to create a document from LaTeX source. Sometimes, the document is complicated enough that it requires a makefile.

The `tetex-bin` package contains many programs to convert DVI files to many other formats. To keep things confusing, numerous wrapper programs allow you to shortcut the process. One such tool is `pdflatex`, which converts LaTeX source directly to PDF. It looks direct to you, but the process is essentially the same as if you compiled with LaTeX and converted the DVI file to PDF with `dvipdf`. Aside from requiring less typing, `pdflatex` will clean up some of the many intermediate files that LaTeX creates in the process of compiling.

Most desktop distributions include support for TeX and LaTeX, but some smaller ones will strip these tools out to save space, and others may not include some of the various tools. Keep this in mind when you work with TeX and LaTeX documentation.

3.4.2 Texinfo

Texinfo is the preferred format for documentation of GNU programs. The basic documentation browser for GNU documentation is the `info` program or Emacs, depending on who you ask. The files used by `info` are stored in the `/usr/share/info` directory, but these are not Texinfo files. These files are created from Texinfo source using the `makeinfo` program. Most binary distributions do not include Texinfo source files but instead provide the preformatted `info` files. These are indicated with the `.info` extension, whereas Texinfo source files usually are indicated by the `.texi` or `.texinfo` extension. As you might expect, a source distribution will include Texinfo source as well, but some binary distributions also include Texinfo source. If any Texinfo source files are provided with a binary distribution, they are most likely to be found under `/usr/share/doc`. It's worth looking, because occasionally, you will find more comprehensive documentation than is in the manual or `info` pages.

Texinfo source code is formatted with the `makeinfo` program. This allows you to convert it to `info`, HTML, DocBook, XML, or plain text. With the `--html` flag, for example, `makeinfo` creates a working Web document in a subdirectory under the current working directory. A simple example might look like the following:

```
$ makeinfo --html foo.texi
$ yelp foo/index.html
```

`makeinfo` produces several formats, including DocBook, XML, and HTML, but not DVI. For DVI, you can use the TeX formatter as long as it includes this line at the top:

```
\input texinfo
```

This line is a directive for the TeX formatter (ignored by `makeinfo`) that instructs it to import the `texinfo` package, which is what allows the TeX compiler to format Texinfo source. With this line, you can use the `tex` command to format a simple Texinfo source file to produce DVI. For most Texinfo source documents, you should use `texi2dvi` instead of the `tex` command.

3.4.3 DocBook

DocBook is an SGML[9] format for authoring documentation used by some tools. Strictly speaking, DocBook is not a file format but an SGML Document Type Definition (DTD). SGML is the predecessor and a superset of the ubiquitous XML format. DocBook follows the goals of SGML, which are to provide a storage format for documentation that separates content from style. This allows content to be searched electronically without style and markup information corrupting the search.

The DocBook DTD is valid in both SGML and XML, although DocBook files usually have the `.sgml` extension. DocBook is not a user-friendly format for writing documents from scratch. Authors are encouraged to use other tools to create formatted documentation and convert it to DocBook. `makeinfo` will generate DocBook-formatted SGML, for example. The purpose of this would be to strip the content from the Texinfo markup for the purpose of searching. The DocBook source could still be used to produce the typeset documentation in whatever format you want. KDE, for example, uses DocBook for `khelpcenter` documents.

DocBook source files can be identified by the `.sgml` or `.docbook` extension. In a binary distribution, these are most likely to be found in `/usr/share/doc`, but these may not be suitable as stand-alone documents. A DocBook file may be part of a larger scheme for online help that requires many other supporting files.

If you want to manipulate DocBook source, the `docbook-utils` package contains several programs to transform DocBook source into other formats. This requires the `jadetex` package to do its transformations using the TeX language.

---

9. *SGML* stands for *Standard Generalized Markup Language*.

This gives you access to DVI output, which means you can transform DocBook using any tool that understands DVI.

Before trying to manipulate DocBook source by hand, you should look around and see whether the files were created from other source that's easier to manipulate, such as Texinfo or `troff`.

3.4.4 HTML

Because HTML is so pervasive, many authors are more comfortable with it than with other markup languages, such as Texinfo, DocBook, and `troff`. Web pages used for project documentation are usually created by hand in a text editor like `vi` or Emacs. Often, this is only one or two plain text files with no images, JavaScript, or flashy content. Thanks to lightweight browsers like `yelp` and `khelpcenter`, you don't have to start up your bloated Web browser to read one of these HTML files. Here again, the conventional location for HTML documentation is in `/usr/share/doc`.

Some tools use HTML exclusively, without any `man` pages. The NTP (Network Time Protocol) package, for example, has extensive HTML documentation, a sample of which is shown in Figure 3-4.

FIGURE 3-4 A Sample of HTML Documentation from the NTP Package

The downside to HTML documentation is that there are no formatting conventions to guide authors on style or content. I don't know about you, but pictures from *Alice in Wonderland* don't help me much. Don't look for HTML to replace `man` any time soon. There is also no convention for storing HTML documentation in any centralized fashion, so your ability to search HTML documentation is determined by the whims of the author. You won't find much, if any, cross referencing to other documentation, either.

Despite the downside, a good deal of high-quality documentation is provided in HTML format. The ability to present data with proportional and fixed-width fonts can go a long way toward making a document more readable than plain text. HTML is also better suited to internationalized documentation than plain text. Figure 3-5 shows an HTML file in Japanese from the `udev` package that happened to be installed on my system.

FIGURE 3-5 A Sample of HTML Documentation in Japanese from the `udev` Package

Depending on your system setup, your terminal may have trouble presenting plain text encoded in a non–ISO-8859 encoding. Most browsers, on the other hand, can present other encodings without much fuss.

3.4.5 PostScript

The PostScript language, created by Adobe, is the predecessor to the Portable Document Format (PDF) in wide use today. PostScript was once the dominant printer language in the UNIX world. It was common for software packages (both commercial and open source) to include documentation in PostScript format, much the same way that we use Portable Document Format (PDF) today.

At that time, PostScript printers were the preferred printers for UNIX systems. To typeset without PostScript, you would need a driver that could typeset a document on the local machine and understand how to send the proper commands to the particular printer.[10] Typically, this involved sending a large amount of data through a low-speed interface such as a serial port, parallel port, or shared Ethernet, which made printing typeset documents a very time-consuming task. As an alternative, PostScript provided a relatively concise language that allowed you to offload the typesetting task onto the printer. This made PostScript print jobs much faster but made the printers significantly more expensive.

Things have changed since then, and PostScript has faded from popularity. For one thing, the idea of a smart printer that can offload typesetting tasks is not as appealing as it used to be. The world is now full of inexpensive, dumb printers capable of producing high-quality output. A typical low-cost printer has a high-speed USB interface and is connected to a computer with horsepower and memory to spare. Although there are still top-of-the-line workgroup printers that support PostScript, it is no longer dominant in the UNIX world.

The GNU Ghostscript tools were a big enabler for using PostScript on UNIX. Ghostscript allowed those of us who couldn't afford expensive PostScript printers to print PostScript documents on less expensive printers (albeit very slowly). One thing Ghostscript allows you to do is view a PostScript file onscreen instead of

---

10. You can still see evidence of this legacy in the DVI and `groff` tools, which still provide output in HP PCL and other printer languages.

printing it. This is done via the `gs` command. The default output is to an X window, for example:

```
$ gs somedoc.ps          Displays the contents of somedoc.ps onscreen
```

When Ghostscript was introduced, viewing typeset documentation onscreen still was fairly novel. Only TeX users had that luxury with `xdvi`. With Ghostscript, anyone could view typeset documentation in PostScript format onscreen.

Ghostscript is primarily a printing tool; there are better ways to view a PostScript document onscreen. One such tool is GNOME's `evince`[11] document viewer, which can be used to view PostScript and PDF files. KDE uses `konqueror` as a front end for Ghostscript, which is a little more user-friendly than Ghostscript by itself.

3.4.6 Portable Document Format (PDF)

Although PDF is excellent for producing typeset output for printing, it is equally suitable for browsing online. One reason why it is so popular is that it saves vendors money by requiring users to print their own manuals. Although it's inconvenient to the user, this is preferable to having a single sheet of paper printed in ten languages or no manual at all.

On the more positive side, PDF is a nice way to encapsulate hyperlinked content in a single file. A well-done PDF file has a hyperlinked table of contents and index, which can make browsing the manual online much more effective than using a printed manual.

Thanks to Adobe, PDF is an open format, so open source tools can be created to view and create PDF files without royalties. Despite this, PDF documentation is not found in open source packages as commonly as it is in commercial packages. This is probably due to the tools, which have been lacking for some time, but that is changing.

Ghostscript can display PDF files but lacks the refinements of Adobe Acrobat Reader. There are no hyperlinks or table of contents, for example. The open source `xpdf`[12] program is still maturing but is slow at rendering pages. GNOME's `evince` is relatively new but is much faster at rendering pages than `xpdf`. GNOME has big plans for `evince` as a tool for viewing PostScript, PDF, and DVI.

---

11. www.gnome.org/projects/evince
12. www.foolabs.com/xpdf

Adobe has made Acrobat Reader available for Linux for a long time, although historically, the Linux version has lagged the Windows version considerably in features. Adobe recently ported Acrobat Reader 7.0 to Linux, which brings it up to date with the latest Windows version. Unfortunately, it also has much of the bloat of the Windows version. For this reason, it may be preferable to use one of the open source viewers.

3.4.7 troff

This is the native markup language of `man` pages. It has a long history that predates UNIX.[13] The tool used for this purpose is `groff`, which is the GNU version of `troff`, but you don't need to know that, because the `man` program will do most of what you need. The following two commands, for example, are equivalent:

```
$ man intro
$ gzip -dc /usr/share/man/man1/intro.1.gz | groff -man -Tascii | less
```

The `man` program handles the messy tasks of finding the `man` page and uncompressing it with `gzip`, of course.

Although `groff` can generate DVI files from `troff` source, it also can generate several other formats on its own. Many of these are of the printer-language variety, such as HP PCL and PostScript. No doubt this is part of its UNIX legacy. Because you can also convert your output to DVI, you can do what you like with it from there.

3.5 Internet Sources of Information

When you can't find the answers you need on your hard drive, you may be inclined to use the Internet to search for help. Using a search engine can lead you down many blind alleys and consume a great deal of your time. In this section, I discuss some resources you can use to search for information more effectively.

3.5.1 www.gnu.org

In addition to providing the source code for many of the tools that enable Linux, this site provides the manuals. Here, you will find the manuals that come with your

13. Interested readers are encouraged to read `roff(7)` in the online manual. Be advised that there are some inaccuracies.

packages, as well as some that don't. The GNU C library (`glibc`), for example, is fully documented in a manual published in two volumes and spanning more than 1,300 pages. You could purchase the two volumes for $60 each, or you could download the Texinfo source from GNU. The Texinfo source is quietly tucked away under the `glibc` link amid all the other documentation that GNU provides. You might never know that such extensive documentation was here unless you were to look. The Web page gives you no clue as to how much documentation is in each project or how good the documentation; you just have to explore and find that out for yourself. So visit the site and have a look around.

3.5.2 SourceForge.net

Many open source projects that are not sponsored by GNU are hosted by SourceForge.net, including many of the tools in your distribution that you use every day. This isn't always obvious even from the packaging information. The `strace` package that comes with the Ubuntu distribution, for example, does not give credit to the SourceForge.net Web page.

```
$ dpkg -s strace
Package: strace
Maintainer: Roland McGrath <frob@debian.org>
Description: A system call tracer
 strace is a system call tracer, i.e. a debugging tool ...
```

In fact, the only reference to SourceForge.net you will find is an email address for a mailing list on the last line of the `man` page. That's a shame, because SourceForge.net is a valuable resource for users as well as developers. RPM packages seem to do a better job of attributing credit to the original authors via the URL field of the RPM header:

```
$ rpm -qi strace
Name        : strace
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
URL         : http://sourceforge.net/projects/strace/
Summary     : Tracks and displays system calls ...
```

If you have an RPM distribution, it pays to look at the information section of the RPM to see whether there is a place you can look for information. If you are using a Debian distribution, you may want to go directly to SourceForge.net and look around.

Each project on SourceForge.net has its own home page and the ability to host forums where users can discuss issues and ask questions. The forums are where you can communicate with other users and power users about a particular project you are interested in. These forums typically have very low traffic, so the "signal-to-noise" ratio is high. Just browsing the archives will be educational, and the subscribers aren't likely to flame you too badly if you post a dumb question.

3.5.3 The Linux Documentation Project

If you have never visited www.tldp.org before, you owe it to yourself to visit. TLDP is the source for the `man` pages that most Linux distributions use. Much of the other information here is of a more general nature, so you are unlikely to find information about a specific open source tool, for example. There is some very good reading here nonetheless. The material is arranged into a few major buckets, including `HOWTO`s, Guides, FAQs, and of course `man` pages.

Be sure to check the dates of the documents before you invest too much time reading them. Linux changes so fast that the documentation has a relatively short shelf life. These documents are written and maintained by volunteers, so they are likely to go out of date before someone has time to update them.

The `HOWTO` documents are often very useful, because they cover very specific topics and generally walk you through a process without requiring a great deal of background knowledge. The topic of the `HOWTO` is usually enough of a clue to tell you whether you want to read it. Because these are very specific topics, it's hard to find one that will waste your time.

The FAQ comes from Usenet newsgroups, where the same questions are often posted over and over.[14] The FAQ was an attempt to cut down on the noise of the same questions being posted so often. Unfortunately, FAQ is a misused term these days, because FAQs often contain questions that the author wished had been asked, rather than ones that have actually been asked. Unlike the lame, contrived FAQs you may encounter on a run-of-the mill Web page, most of the TLDP FAQs are created from real questions asked by real people (some even frequently). If you have a specific question, the FAQ is worth a look.

A good place to start is The Linux Documentation Project FAQ, located at www.tldp.org/FAQ/LDP-FAQ/index.html.

---

14. In case you didn't know, *FAQ* stands for *Frequently Asked Questions.*

3.5.4 Usenet

Usenet has a long history that predates the Internet. Usenet "newsgroups" are often more gossip and blather than news. Here, anyone can say anything, and the only repercussions are the so-called "flames" of postings by other angry readers. You will find many strong opinions and a general lack of manners.

Browsing Usenet archives via Google or another search engine is advised before actually posting a question on one of these newsgroups. There's a chance that your question has been asked before. It also helps to get to know some of the regular posters to see whether anyone is actually getting help.

Before you use any information you get from Usenet, make sure to verify it with some other source. Be skeptical of everything you read. There is no guarantee that things you read are accurate or even safe to try.

Be advised that there is no attempt to hide your e-mail address in Usenet postings, so it is out there for the world to see. Usenet groups are gold mines for spammers.

3.5.5 Mailing Lists

Of the many forum formats, this is the one that I have found to be the most valuable. What you get in a mailing list is a group of people who are passionate about the topic and generally eager to help (otherwise, they wouldn't subscribe). As with any forum, it is always advisable to do your homework before posting a question. Although people on mailing lists usually have good manners, it's rude to waste people's time with simple questions you could look up yourself.

A valuable resource for finding mailing lists is the mailing-list archive (MARC) located at http://marc.theaimsgroup.com. This site contains archives of hundreds of mailing lists on Linux as well as many other computer topics.

3.5.6 Other Forums

Many Linux Web sites host forums similar to Usenet that are targeted to Linux topics. These usually require some kind of subscription before you can post, but many allow you to read their archives without signing in. The quality of postings can vary wildly.

3.6 Finding Information about the Linux Kernel

Documenting the Linux kernel is a huge task. Several good books have been written on the topic, but the kernel is a moving target. This makes just about any book

on the kernel obsolete by the time it is published. The most up-to-date documentation you are going to find is located in the kernel source tree. The kernel source, as it is distributed from http://kernel.org, contains quite a bit of documentation. In addition, many kernel modules provide helpful information embedded in the kernel objects themselves.

3.6.1 The Kernel Build

The kernel build process is well documented in many places on the Internet. Although the 2.6 series has simplified the build process significantly, there are plenty of gotchas and little details worth knowing about. The first place to look, of course, is the `README` file located at the root of the kernel source tree. It starts with "What Is Linux," which may make your eyes glaze over and cause you to give up prematurely. But if you stick with it, you will find that somewhere in the middle, it talks about all the targets you can build with the top-level makefile. Some of these are quite useful, and a summary can be printed at any time by typing

```
$ make help

Cleaning targets:
  clean           - remove most generated files but keep the config
  mrproper        - remove all generated files + config + various
                    backup files

Configuration targets:
  config          - Update current config utilising a line-oriented
                    program
  menuconfig      - Update current config utilising a menu based
                    program
  xconfig         - Update current config utilising a QT based front-
                     end
  gconfig         - Update current config utilising a GTK based front-
                    end
  oldconfig       - Update current config utilising a provided .config
                    as base
  randconfig      - New config with random answer to all options
  defconfig       - New config with default answer to all options
  allmodconfig    - New config selecting modules when possible
  allyesconfig    - New config where all options are accepted with yes
  allnoconfig     - New minimal config

Other generic targets:
  all             - Build all targets marked with [*]
* vmlinux         - Build the bare kernel
* modules         - Build all modules
```

```
  modules_install - Install all modules
  dir/            - Build all files in dir and below
  dir/file.[ois]  - Build specified target only
  rpm             - Build a kernel as an RPM package
  tags/TAGS       - Generate tags file for editors
  cscope          - Generate cscope index

Static analysers
  buildcheck      - List dangling references to vmlinux discarded
                    sections
                    and init sections from non-init sections
  checkstack      - Generate a list of stack hogs
  namespacecheck  - Name space analysis on compiled kernel

Kernel packaging:
  rpm-pkg         - Build the kernel as an RPM package
  binrpm-pkg      - Build an rpm package containing the compiled
                    kernel & modules
  deb-pkg         - Build the kernel as an deb package

Documentation targets:
  Linux kernel internal documentation in different formats:
  xmldocs (XML DocBook), psdocs (PostScript), pdfdocs (PDF)
  htmldocs (HTML), mandocs (man pages, use installmandocs to install)

Architecture specific targets (i386):
* bzImage       - Compressed kernel image (arch/i386/boot/bzImage)
  install       - Install kernel using
                    (your) ~/bin/installkernel or
                    (distribution) /sbin/installkernel or
                    install to $(INSTALL_PATH) and run lilo
  bzdisk       - Create a boot floppy in /dev/fd0
  fdimage      - Create a boot floppy image

  make V=0|1 [targets] 0 => quiet build (default), 1 => verbose build
  make O=dir [targets] Locate all output files in "dir", including
                       .config
  make C=1   [targets] Check all c source with $CHECK (sparse)
  make C=2   [targets] Force check of all c source with $CHECK
                       (sparse)

Execute "make" or "make all" to build all targets marked with [*]
For further info see the ./README file
```

Additionally, each selectable feature of the kernel is documented in the `Kconfig` files that you find in many subdirectories of the kernel source tree. These files contain the text of the help messages you see when you create a new kernel configuration via `make config`, `make menuconfig`, etc. The `Kconfig` files are plain text files, so you can look at them with any text editor.

3.6.2 Kernel Modules

Linux provides features for kernel modules to do a modest amount of self documentation. The `modinfo` command allows you to see information in the module that the author may have placed there. Specifically, modules can take parameters when loaded, much like command-line parameters. These options are shown by the `modinfo` command, along with whatever documentation the author provided. For most modules, the information is minimal, but for some, it is quite extensive. If the module takes a kernel parameter, it will be listed by `modinfo`, whether it's documented or not. Even if it's not documented, this at least gives you some direction as to what to look for in the source.

One example of how extensive the `modinfo` documentation can be is the `aic79xx` module used with Adaptec SCSI controllers. The `modinfo` output is shown below:

```
aic79xx:period delimited, options string.
  verbose       Enable verbose/diagnostic logging
  allow_memio   Allow device registers to be memory mapped
  debug         Bitmask of debug values to enable
  no_reset      Suppress initial bus resets
  extended      Enable extended geometry on all controllers
  periodic_otag Send an ordered tagged transaction
                periodically to prevent tag starvation.
                This may be required by some older disk
                or drives/RAID arrays.
  reverse_scan  Sort PCI devices highest Bus/Slot to lowest
  tag_info:<tag_str>    Set per-target tag depth
  global_tag_depth:<int> Global tag depth for all targets on all buses
  rd_strm:<rd_strm_masks> Set per-target read streaming setting.
  dv:<dv_settings>      Set per-controller Domain Validation Setting.
  slewrate:<slewrate_list>Set the signal slew rate (0-15).
  precomp:<pcomp_list>  Set the signal precompensation (0-7).
  amplitude:<int>   Set the signal amplitude (0-7).
  seltime:<int>     Selection Timeout:
                    (0/256ms,1/128ms,2/64ms,3/32ms)

  Sample /etc/modprobe.conf line:
    Enable verbose logging
    Set tag depth on Controller 2/Target 2 to 10 tags
    Shorten the selection timeout to 128ms

  options aic79xx 'aic79xx=verbose.tag_info:{{}.{}.{..10}}.seltime:1'

  Sample /etc/modprobe.conf line:
    Change Read Streaming for Controller's 2 and 3


  options aic79xx 'aic79xx=rd_strm:{..0xFFF0.0xC0F0}'
```

The driver takes only one parameter, named `aic79xx`, but the variable contains text that encodes dozens of details. Indeed, the `modinfo` output for the `aic79xx` module reads almost like a `man` page, but this is an extreme case. Most modules take no parameters and contain little or no information, but you will never know unless you try.

3.6.3 Miscellaneous Documentation

The kernel source distribution contains a treasure trove of reference material in the `Documentation` directory. The `Documentation` directory has more than 700 files, most of them plain text with a few DocBook files thrown in for good measure. The DocBook source is actually part of the kernel build. If you want to view it, there are several targets you can use to create the format you want:

```
$ make pdfdocs          # Generate PDF files
$ make mandocs          # Generate man pages
$ make psdocs           # Generate PostScript output
```

There is a great deal of diverse information here, including information for kernel hackers as well as system administrators.

3.7 Summary

This chapter introduced the main sources of documentation for Linux tools and how to use them. It looked at the various tools used to retrieve documentation and examined some of the sources of documentation. Along with the sources, the chapter looked at the formats that Linux documentation comes in. Finally, the chapter looked specifically at the kernel and the documentation that is available for various kernel modules and for the rest of the kernel.

3.7.1 Tools Used in This Chapter

- `man`—the original UNIX help tool
- `apropos`—searches the `man` headlines for keywords
- `whatis`—searches the `whatis` database of `man` pages (created by `makewhatis`) for keywords
- `info`—the GNU help tool that supports more complex documents

- `yelp`—the GNOME help tool that is also a functional Web browser
- `khelpcenter`—the KDE help tool
- `xdvi`, `kdvi`—tools for viewing documents in DVI format
- `evince`—the GNU all-purpose viewer for DVI, PDF, PostScript, and other formats
- `makeinfo`—used to render Texinfo-formatted documentation into other formats (HTML, DocBook, and others)
- `gs`—the command-line front end for Ghostscript, the GNU PostScript viewer
- `xpdf`—an open source PDF viewer for the X Window System

3.7.2 Online Resources

- www.troff.org—dedicated to the `troff` formatting language
- www.pathname.com/fhs—the Filesystem Hierarchy Standard
- http://marc.theaimsgroup.com—archives of various mailing lists
- www.foolabs.com/xpdf—the `xpdf` project home page
- www.gnome.org/projects/evince—the `evince` project home page
- pinfo.alioth.debian.org—the `pinfo` home page