Texinfo

The GNU Documentation Format
Edition 2.05
April 1991

by Robert J. Chassell and Richard M. Stallman

This is the second edition of the Texinfo documentation,
and is consistent with version 2 of '`texinfo.tex`'.

Published by the Free Software Foundation
675 Massachusetts Avenue,
Cambridge, MA 02139 USA
Printed copies are available for $15 each.

Cover art by Etienne Suvasa.

Texinfo Copying Conditions

The programs currently being distributed that relate to Texinfo include portions of GNU Emacs, plus other separate programs (including `makeinfo`, `info`, `texindex`, and '`texinfo.tex`'). These programs are *free*; this means that everyone is free to use them and free to redistribute them on a free basis. The Texinfo-related programs are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to Texinfo, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the Texinfo related programs, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to Texinfo. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to Texinfo are found in the General Public Licenses that accompany them.

1 Overview of Texinfo

Texinfo is a documentation system that uses a single source file to produce both on-line help (and other information) and a printed manual. This means that instead of writing two different documents, one providing on-line information and the other for a printed manual, you need write only one document. When the system is revised, you need revise only one document. (You can read the on-line help, known as an *Info file*, with the Info documentation-reading program.)[1]

Using Texinfo, you can create a printed document with the normal features of a book, including chapters, sections, cross references, and indices. From the same Texinfo source file, you can create a menu-driven, on-line Info file with nodes, menus, cross references, and indices. You can, if you wish, make the chapters and sections of the printed document correspond to the nodes of the on-line information; and you use the same cross references and indices for both the Info file and the printed document. The *GNU Emacs Manual* is a good example of a Texinfo file.

To make a printed manual, process a Texinfo source file with the TEX typesetting program. This creates a DVI file that you can typeset and print as a book. (Note that the Texinfo language is completely different from TEX's usual language, PlainTEX, which Texinfo replaces.) If you do not have TEX, but do have `troff` or `nroff`, you can use the `texi2roff` program instead. To create an Info file, process a Texinfo source file with the `makeinfo` utility or Emacs's `texinfo-format-buffer` command; this creates an Info file that you can install on-line.

Info works with almost every type of computer terminal; similarly, TEX and `texi2roff` work with many types of printer. This power makes Texinfo a general purpose system, but brings with it a constraint, which is that a Texinfo file may contain only the customary "typewriter" characters (letters, numbers, spaces, and punctuation marks) but no special graphics.

A Texinfo file is a plain ASCII file containing text and `@`-*commands* (words preceded by an '`@`') that tell the typesetting and formatting programs what to do. You may edit a Texinfo file with any text editor; but it is especially convenient to use GNU Emacs since that editor has a special mode, called Texinfo mode, that provides various Texinfo-related features. (See Chapter 2 [Texinfo Mode], page 15.)

---

[1] Note that the first syllable of "texinfo" is pronounced like "speck", not "hex". This odd pronunciation is derived from TEX, in which the '`X`' is actually the Greek letter "chi" rather than the English letter "ex" (the '`T`' and '`E`' are Greek letters also, but they happen to be pronounced the same way in Greek as in English).

Before writing a Texinfo source file, you should become familiar with the Info documentation reading program and learn about nodes, menus, cross references, and the rest. (See Info file '`info`', node '`Top`', for more information.)

Texinfo creates both on-line help and a printed manual; moreover, it is freely redistributable. For these reasons, Texinfo is the format in which documentation for GNU utilities and libraries should be written.

1.1 Info files

An Info file is a file formatted so that the Info documentation reading program can operate on it. Info files are divided into pieces called *nodes*, each of which contains the discussion of one topic. Each node has a name, and contains both text for the user to read and pointers to other nodes, which are identified by their names. The Info program displays one node at a time, and provides commands with which the user can move to the other related nodes.

Each node of an Info file may have any number of child nodes that describe subtopics of the node's topic. The names of these child nodes, if any, are listed in a *menu* within the parent node; this allows you to use certain Info commands to move to one of the child nodes. Generally, a Texinfo file is organized like a book. If a node is at the logical level of a chapter, its child nodes are at the level of sections; likewise, the child nodes of sections are at the level of subsections.

All the children of any one parent are linked together in a bidirectional chain of 'Next' and 'Previous' pointers. This means that all the nodes that are at the level of sections within a chapter are linked together. Normally the order in this chain is the same as the order of the children in the parent's menu. Each child node records the parent node name, as its 'Up' pointer. The last child has no 'Next' pointer, and the first child has the parent both as its 'Previous' and as its 'Up' pointer.[2]

The book-like structuring of an Info file into nodes that correspond to chapters, sections, and the like is a matter of convention, not a requirement. The 'Up', 'Previous', and 'Next' pointers of a node can point to any other nodes, and a menu can contain any other nodes. Thus, the node structure can be any directed graph. But it is usually more comprehensible to follow a structure that corresponds to the structure of chapters and sections in a printed manual.

---

[2] In some documents, the first child has no 'Previous' pointer. Occassionally, the last child has the node name of the next following higher level node as its 'Next' pointer.

In addition to 'Next', 'Previous', and 'Up' pointers and menus, Info provides pointers of another kind, called references, that can be sprinkled throughout the text. This is usually the best way to represent links that do not fit a hierarchical structure.

Usually, you will design a document so that its nodes match the structure of chapters and sections in the printed manual. But there are times when this is not right for the material being discussed. Therefore, Texinfo uses separate commands to specify the node structure of the Info file and the section structure of the printed manual.

Generally, you enter an Info file through a node that by convention is called '`Top`'. This node normally contains just a brief summary of the file's purpose, and a large menu through which the rest of the file is reached. From this node, you can either traverse the file systematically by going from node to node, or you can go to a specific node listed in the main menu, or you can search the index menus and then go directly to the node that has the information you want.

If you want to read through an Info file in sequence, as if it were a printed manual, you can get the whole file with the advanced Info command `g* RET`. (See Info file '`info`', node '`Expert`'.)

The '`dir`' file in the '`.../emacs/info`' directory serves as the departure point for the whole Info system. From it, you can reach the top nodes of each of the documents in a complete Info system.

1.2 Printed Manuals

A Texinfo file can be formatted and typeset as a printed manual. To do this, you need to use TEX, a powerful, sophisticated typesetting program written by Donald Knuth.[3]

A Texinfo-based printed manual will be similar to any other book; it will have a title page, copyright page, table of contents, and preface, as well as chapters, numbered or unnumbered sections and subsections, page headers, cross references, footnotes, and indices.

You can use Texinfo to write a book without ever having the intention of converting it into on-line information. You can use Texinfo for writing a printed novel, and even to write a printed

[3] You can also use the `texi2roff` program if you do not have TEX; since Texinfo is designed for use with TEX, `texi2roff` is not described here. `texi2roff` is part of the standard GNU distribution.

memo, although this latter application is not recommended since electronic mail is so much easier.

TEX is a general purpose typesetting program. Texinfo provides a file called '`texinfo.tex`' that contains information (definitions or *macros*) that TEX uses when it typesets a Texinfo file. (The macros tell TEX how to convert the Texinfo `@`-commands to TEX commands, which TEX can then process to create the typeset document.) '`texinfo.tex`' contains the specifications for printing a document, either with 7 inch by 9.25 inch pages or with 8.5 inch by 11 inch pages. (This is 178 mm by 235 mm or else 216 mm by 280 mm.) By changing the parameters in '`texinfo.tex`', you can change the size of the printed document. In addition, you can change the style in which the printed document is formatted; for example, you can change the sizes and fonts used, the amount of indentation for each paragraph, the degree to which words are hyphenated, and the like. By changing the specifications, you can make a book look dignified, old and serious, or light-hearted, young and cheery.

TEX is freely distributable. It is written in a dialect of Pascal called WEB and can be compiled either in Pascal or (by using a conversion program that comes with the TEX distribution) in C. (See section "TeX Mode" in *The GNU Emacs Manual*, for information about TEX.)

TEX is very powerful and has a great many features. Because a Texinfo file must be able to present information both on a character-only terminal in Info form and in a typeset book, the formatting commands that Texinfo supports are necessarily limited.

1.2.1 Obtaining TEX

TEX is freely redistributable. You can obtain TEX for Unix systems from the University of Washington for a distribution fee.

To order a full distribution, send $140.00 for a 1/2-inch 9-track 1600 bpi (tar or cpio) tape reel, or $165.00 for a 1/4-inch 4-track QIC-24 (tar or cpio) cartridge, to:

Northwest Computing Support Center
DR-10, Thomson Hall 35
University of Washington
Seattle, Washington 98195

Please make checks payable to the University of Washington.

Prepaid orders are preferred but purchase orders are acceptable; however, purchase orders carry

an extra charge of $10.00, to pay for processing.

Overseas sites: please add to the base cost $20.00 for shipment via air parcel post, or $30.00 for shipment via courier.

Please check with the Northwest Computing Support Center at the University of Washington for current prices and formats:

telephone: (206) 543-6259
email: elisabet@max.u.washington.edu

1.3 @-commands

In a Texinfo file, the commands that tell TEX how to typeset the printed manual and tell `makeinfo` and `texinfo-format-buffer` how to create an Info file are preceded by '`@`'; they are called `@`-*commands*. For example, `@node` is the command to indicate a node and `@chapter` is the command to indicate the start of a chapter.

> **Please note:** All the `@`-commands, with the exception of the `@TeX{}` command, must be written entirely in lower case.

The Texinfo `@`-commands are a strictly limited set of constructs. The strict limits make it possible for Texinfo files to be understood both by TEX and by the code that converts them into Info files. You can display Info files on any terminal that displays alphabetic and numeric characters. Similarly, you can print the output generated by TEX on a wide variety of printers.

Depending on what they do or what arguments[4] they take, you need to write `@`-commands on lines of their own or as part of sentences:

---

[4] The word *argument* comes from the way it is used in mathematics and does not refer to a disputation between two people; it refers to the information presented to the command. According to the *Oxford English Dictionary*, the word derives from the Latin for *to make clear, prove*; thus it came to mean 'the evidence offered as proof', which is to say, 'the information offered', which led to its mathematical meaning. In its other thread of derivation, the word came to mean 'to assert in a manner against which others may make counter assertions', which led to the meaning of 'argument' as a disputation.

- Write a command such as `@page` at the beginning of a line as the only text on the line.
- Write a command such as `@chapter` at the beginning of a line followed by the command's arguments, in this case the chapter title, on the rest of the line.
- Write a command such as `@dots{}` wherever you wish (but usually within a sentence).
- Write a command such as `@code{`*sample-code*`}` wherever you wish (but usually within a sentence) with its argument, *sample-code*, in this example, between the braces.

As you gain experience with Texinfo, you will rapidly learn how to write the different commands: the different ways to write commands make it easier to write and read Texinfo files than if all commands followed exactly the same syntax.

1.4 General Syntactic Conventions

All ASCII printing characters except '`@`', '`{`' and '`}`' can appear in a Texinfo file and stand for themselves. '`@`' is the escape character which introduces commands. '`{`' and '`}`' should be used only to surround arguments to certain commands. To put one of these special characters into the document, put an '`@`' character in front of it, like this: '`@@`', '`@{`', and '`@}`'.

It is customary in TEX to use doubled single-quote characters to begin and end quotations: '`‘‘`' and '`’’`'. This convention should be followed in Texinfo files. TEX converts doubled single-quote characters to left- and right-hand doubled quotation marks, "like this," and Info converts doubled single-quote characters to ASCII double-quotes: '`‘‘`' and '`’’`' to '`"`'.

Use three hyphens in a row, '`---`', for a dash—like this. In TEX, a single or even a double hyphen produces a printed dash that is shorter than you want. Info reduces three hyphens to two for display on the screen.

To prevent a paragraph from being indented in the printed manual, put the command `@noindent` on a line by itself before the paragraph.

If you mark off a region of the Texinfo file with the `@iftex` and `@end iftex` commands, that region will appear only in the printed copy; in that region, you can use certain commands borrowed from PlainTEX that you cannot use in Info. Likewise, if you mark off a region with the `@ifinfo` and `@end ifinfo` commands, that region will appear only in the Info file; in that region, you can use Info commands that you cannot use in TEX. (See Chapter 17 [Conditionals], page 141.)

> **Caution:** Do not use tabs in a Texinfo file! TEX has trouble with tabs: it treats them like single spaces, and that is not what they look like. This is a problem with TEX. (To

avoid putting tabs into your file, you can set the `indent-tabs-mode` variable in Emacs to `nil` so that Emacs inserts multiple spaces when you press the `TAB` key. Also, you can run `untabify` to convert tabs in a region to multiple spaces.)

1.5 Comments

You can write comments in a Texinfo file that will not appear in either the Info file or the printed manual by using the `@comment` command (which may be abbreviated to `@c`). Such comments are for the person who reads the Texinfo file. All the text on a line that follows either `@comment` or `@c` is a comment; the rest of the line does not appear in either the Info file or the printed manual. (The `@comment` or `@c` does not have to be at the beginning of the line; only the text on the line that follows after the `@comment` or `@c` command does not appear.)

You can write long stretches of text that will not appear in either the Info file or the printed manual by using the `@ignore` and `@end ignore` commands. Write each of these commands on a line of its own, starting each command at the beginning of the line. Text between these two commands does not appear in the processed output. You can use `@ignore` and `@end ignore` for writing comments or for holding text you may wish to use in another version of your document. Often, `@ignore` and `@end ignore` is used to enclose a part of the copying permissions that applies to the Texinfo source file of a document, but not to the Info or printed version of the document.

1.6 What a Texinfo File Must Have

By convention, the names of Texinfo files end with either of the three extensions '`.texinfo`', '`.texi`', or '`.tex`'. The longer extension is preferred since it describes more clearly to a human reader the nature of the file. The shorter extensions are for operating systems that cannot handle long file names.

In order to be made into a printed manual and an Info file, a Texinfo file **must** begin with lines like this:

```
\input texinfo
@setfilename info-file-name
@settitle name-of-manual
```

The contents of the file follow this beginning, and then you must end a Texinfo file with a line like this:

```
@bye
```

The '`\input texinfo`' line tells TEX to use the '`texinfo.tex`' file, which tells TEX how to translate the Texinfo `@`-commands into TEX typesetting commands. The '`@setfilename`' line provides a name for the Info file and the '`@settitle`' line specifies a title for the page headers (or footers) of the printed manual.

The `@bye` line at the end of the file on a line of its own tells TEX that the file is ended and to stop typesetting.

Usually, you won't use quite such a spare format, but will include mode setting and start-of-header and end-of-header lines at the beginning of a Texinfo file, like this:

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename info-file-name
@settitle name-of-manual
@c %**end of header
```

In the first line, '`-*-texinfo-*-`' causes Emacs to switch into Texinfo mode when you edit the file.

The `@c` lines which surround the `@setfilename` and `@settitle` lines are optional, but you need them in order to run TEX or Info on just part of the file. (See Section 3.2.2 [Start of Header], page 29, for more information.)

Furthermore, you will usually provide a Texinfo file with a title page, master menu, and the like. But the minimum, which can be useful for short documents, is just the three lines at the beginning and the one line at the end.

1.7 Six Parts of a Texinfo File

Generally, a Texinfo file contains contains more than the minimal beginning and end—it usually contains six parts:

Header — The Header names the file, tells TEX which definitions' file to use, and performs other "housekeeping" tasks.

Summary Description and Copyright
: The Summary Description and Copyright segment describes the document and contains

the copyright notice and copying permissions for the Info file. The segment must be enclosed between `@ifinfo` and `@end ifinfo` commands so that it appears only in the Info file.

Title and Copyright
: The Title and Copyright segment contains the title and copyright pages and copying permissions for the printed manual. The segment must be enclosed between `@titlepage` and `@end titlepage` commands and appears only in the printed manual.

Top Node and Master Menu
: The Master Menu contains a complete menu of all the nodes in the whole Info file. It appears only in the Info file, in the Top node.

Body
: The Body of the document may be structured like a traditional book or encyclopedia or it may be free form.

End
: The End contains commands for printing indices and generating the table of contents, and the `@bye` command on a line of its own.

1.8 A Short Sample Texinfo File

Here is a complete but very short Texinfo file. The first three parts of the file, from '`\input texinfo`' through to '`@end titlepage`', look more intimidating than they are. Most of the material is standard boilerplate; when you write a manual, simply insert the names for your own manual in this segment. (See Chapter 3 [Beginning a File], page 27.)

Part 1: Header

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename sample.info
@settitle Sample Document
@c %**end of header
@setchapternewpage odd
```

Part 2: Summary Description and Copyright

The summary description and copyright segment does not
appear in the printed document.

```
@ifinfo
This is a short example of a complete Texinfo file.

Copyright @copyright{} 1990 Free Software Foundation, Inc.
@end ifinfo
```

Part 3: Titlepage and Copyright

The titlepage segment does not appear in the Info file.

```
@titlepage
@sp 10
@comment The title is printed in a large font.
@center @titlefont{Sample Title}

@c The following two commands start the copyright page.
@page
@vskip 0pt plus 1filll
Copyright @copyright{} 1990 Free Software Foundation, Inc.
@end titlepage
```

Part 4: Top Node and Master Menu

The Top node contains the master menu for the Info file.
Since a printed manual uses a table of contents rather than
a menu, the master menu appears only in the Info file.

```
@node    Top,       First Chapter, (dir),    (dir)
@comment node-name, next,          previous, up


@menu
* First Chapter::    The first chapter is the
                     only chapter in this sample.
* Concept Index::    This index has two entries.
@end menu
```

Part 5: The Body of the Document

```
@node     First Chapter, Concept Index, Top,      Top
@comment  node-name,     next,          previous, up
@chapter First Chapter
```

```
@cindex Sample index entry

This is the contents of the first chapter.
@cindex Another sample index entry

Here is a numbered list.

@enumerate
@item
This is the first item.

@item
This is the second item.
@end enumerate

The @code{makeinfo} and @code{texinfo-format-buffer}
commands transform a Texinfo file such as this into
an Info file; and @TeX{} typesets it for a printed
manual.
```

Part 6: The End of the Document

```
@node    Concept Index,    ,  First Chapter, Top
@comment node-name,    next,  previous,      up
@unnumbered Concept Index

@printindex cp

@contents
@bye
```

The Results

Here is what the contents of the first chapter of the sample look like:

This is the contents of the first chapter.

Here is a numbered list.

1. This is the first item.
2. This is the second item.

The `makeinfo` and `texinfo-format-buffer` commands transform a Texinfo file such

as this into an Info file; and TEX typesets it for a printed manual.

2 Using Texinfo Mode

In GNU Emacs, Texinfo mode provides commands and features especially designed for working with Texinfo files:

- Insert commonly used strings of text.
- Automatically create node lines.
- Show the structure of a Texinfo source file.
- Automatically create or update the 'Next', 'Previous', and 'Up' pointers of a node.
- Automatically create or update menus.
- Automatically create a master menu.
- Format a part or all of a file for Info.
- Typeset and print part or all of a file.

The special Texinfo commands are in addition to the usual editing commands, which are generally the same as the commands of Text mode. However, in Texinfo mode the paragraph separation variable and syntax table are redefined so that Texinfo commands that should be on lines of their own are not inadvertently included in paragraphs. Thus, the `M-q` (`fill-paragraph`) command will refill a paragraph but not mix an indexing command on a line adjacent to it into the paragraph.

You may name a Texinfo file however you wish, but the convention is to end a Texinfo file name with one of the three extensions '`.texinfo`', '`.texi`', or '`.tex`'. A longer extension is preferred, since it is explicit, but a shorter extension may be necessary for operating systems that limit the length of file names. GNU Emacs automatically enters Texinfo mode when you visit a file with any of these extensions. Also, Emacs switches to Texinfo mode for a file that has '`-*-texinfo-*-`' in its first line. If ever you are in another mode and wish to switch to Texinfo mode, type `M-x texinfo-mode`.

Like all other Emacs features, you can customize or enhance Texinfo mode as you wish. In particular, the keybindings are very easy to change. The keybindings described here are the default or standard ones.

2.1 Inserting Frequently Used Commands

Texinfo mode provides commands to insert various frequently used `@`-commands into the buffer.

You can use these commands to save keystrokes.

The insert commands are invoked by typing `C-c` twice and then the first letter of the `@`-command.

`C-c C-c c`
`M-x texinfo-insert-@code`
: Insert `@code{}` and put the cursor between the braces.

`C-c C-c d`
`M-x texinfo-insert-@dfn`
: Insert `@dfn{}` and put the cursor between the braces.

`C-c C-c e`
`M-x texinfo-insert-@end`
: Insert `@end`.

`C-c C-c i`
`M-x texinfo-insert-@item`
: Insert `@item` and put the cursor at the beginning of the next line.

`C-c C-c k`
`M-x texinfo-insert-@kbd`
: Insert `@kbd{}` and put the cursor between the braces.

`C-c C-c n`
`M-x texinfo-insert-@node`
: Insert `@node` and a comment line listing the sequence for the 'Next', 'Previous', and 'Up' nodes. Leave point after the `@node`.

`C-c C-c o`
`M-x texinfo-insert-@noindent`
: Insert `@noindent` and put the cursor at the beginning of the next line.

`C-c C-c s`
`M-x texinfo-insert-@samp`
: Insert `@samp{}` and put the cursor between the braces.

`C-c C-c v`
`M-x texinfo-insert-@var`
: Insert `@var{}` and put the cursor between the braces.

`C-c C-c x`
`M-x texinfo-insert-@example`
: Insert `@example` and put the cursor at the beginning of the next line.

`C-c C-c {`
`M-x texinfo-insert-braces`
: Insert `{}` and put the cursor between the braces.

`C-c C-c }`
`M-x up-list`
: Move forward past one set of closing braces.

This set of insert commands was created after analyzing the frequency with which different `@`-commands are used in the *GNU Emacs Manual* and the *GDB Manual*. If you wish to add your own insert commands, you can bind a keyboard macro to a key, use abbreviations, or extend the code in '`texinfo.el`'.

2.2 Showing the Section Structure of a File

You can show the section structure of a Texinfo file by using the `C-c C-s` command (`texinfo-show-structure`). This command shows the section structure of a Texinfo file by listing the lines that begin with the `@`-commands for `@chapter`, `@section`, and the like. The command constructs what amounts to a table of contents. These lines are displayed in another buffer called the '`*Occur*`' buffer. In that buffer, you can position the cursor over one of the lines and use the `C-c C-c` command (`occur-mode-goto-occurrence`), to jump to the corresponding spot in the Texinfo file.

`C-c C-s`
`M-x texinfo-show-structure`
: Show the `@chapter`, `@section`, and such lines of a Texinfo file.

`C-c C-c`
`M-x occur-mode-goto-occurrence`
: Go to the line in the Texinfo file corresponding to the line under the cursor in the '`*Occur*`' buffer.

If you call `texinfo-show-structure` with a prefix argument by typing `C-u C-c C-s`, it will list not only those lines with the `@`-commands for `@chapter`, `@section`, and the like, but also the `@node` lines. (This is how the `texinfo-show-structure` command worked without an argument in the first version of Texinfo. It was changed because `@node` lines clutter up the the '`*Occur*`' buffer and are usually not needed.) You can use `texinfo-show-structure` with a prefix argument to check whether the 'Next', 'Previous', and 'Up' pointers of a node line are correct.

Often, when you are working on a manual, you will be interested only in the structure of the current chapter. In this case, you can mark off the region of the buffer that you are interested in with the `C-x n` (`narrow-to-region`) command and `texinfo-show-structure` will work on only that region. To see the whole buffer again, use `C-x w` (`widen`). (See section "Narrowing" in *The GNU Emacs Manual*, for more information about the narrowing commands.)

2.3 Updating Nodes and Menus

Texinfo mode provides commands for automatically creating or updating menus and node pointers. The commands are called "update" commands because their most frequent use is for updating a Texinfo file after you have worked on it; but you can use them to insert the 'Next', 'Previous', and 'Up' pointers into a node line that has none and to create menus in a file that has none.

You can use the updating commands

- to insert or update the 'Next', 'Previous', and 'Up' pointers of a node,
- to insert or update the menu for a section, and
- to create a master menu for a Texinfo source file.

You can also use the commands to update all the nodes and menus in a region or in a whole Texinfo file.

Texinfo mode has five updating commands that are used most often: two are for updating the node pointers or menu of a single node (or a region), two are for updating every node pointer and menu in a file, and one, the `texinfo-master-menu` command, is for creating a master menu for a complete file, and optionally, for updating every node and menu in the whole Texinfo file.

The `texinfo-master-menu` command is the primary command:

`C-c C-u m`
`M-x texinfo-master-menu`

Create or update a master menu that includes all the other menus (incorporating the descriptions from pre-existing menus, if any).

With an argument (prefix argument, if interactive), first create or update all the nodes and all the regular menus in the buffer before constructing the master menu. (See Section 3.5 [The Top Node and Master Menu], page 39, for more about a master menu.)

For `texinfo-master-menu` to work, the Texinfo file must have a 'Top' node.

After extensively editing a Texinfo file, it is common to type `C-u C-c C-u m` or `C-u M-x texinfo-master-menu` to update all the nodes and menus completely and all at once.

The other major updating commands do smaller jobs and are designed for the person who updates nodes and menus as he or she writes a Texinfo file.

The commands are:

`C-c C-u C-n`
`M-x texinfo-update-node`
: Insert the 'Next', 'Previous', and 'Up' pointers for the node point is within (i.e., for the `@node` line preceding point). If the `@node` line has pre-existing 'Next', 'Previous', or 'Up' pointers in it, the old pointers are removed and new ones inserted. With an argument (prefix argument, if interactive), this command updates all `@node` lines in the region (which is the text between point and mark).

`C-c C-u C-m`
`M-x texinfo-make-menu`
: Create or update the menu in the node that point is within. With an argument (prefix argument, if interactive), the command makes or updates menus for the nodes within or part of the region.

  Whenever `texinfo-make-menu` updates an existing menu, the descriptions from that menu are incorporated into the new menu. This is done by copying descriptions from the existing menu to the entries in the new menu that have the same node names. If the node names are different, the descriptions are not copied to the new menu.

  Menu entries that refer to other Info files are removed since they do not refer to nodes within the current buffer. This is a deficiency.

`C-c C-u C-e`
`M-x texinfo-every-node-update`
: Insert or update the 'Next', 'Previous', and 'Up' pointers for every node in the buffer .

`C-c C-u C-a`
`M-x texinfo-all-menus-update`
: Create or update all the menus in the buffer. With an argument (prefix argument, if interactive), first insert or update all the node pointers before working on the menus.

  If a master menu exists, the `texinfo-all-menus-update` command updates it; but the command does not create a new master menu if none already exists. (Use the `texinfo-master-menu` command for that.)

The `texinfo-column-for-description` variable specifies the column to which menu descriptions are indented. By default, the value is 32 although it is often useful to reduce it to as low as 24. You can set the variable with the `M-x edit-options` command (see section "Editing Variable Values" in *The GNU Emacs Manual*) or with the `M-x set-variable` command (see section "Examining and Setting Variables" in *The GNU Emacs Manual*).

Also, the `texinfo-indent-menu-description` command may be used to indent existing menus to a specified column. See Section 2.3.2 [Other Updating Commands], page 20.

Finally, if you wish, you can use the `texinfo-insert-node-lines` command to insert missing `@node` lines into a file. (See Section 2.3.2 [Other Updating Commands], page 20, for more information.)

2.3.1 Updating Requirements

To use the updating commands, you must organize the Texinfo file hierarchically with chapters, sections, subsections, and the like. Each `@node` line, with the exception of the line for the 'Top' node, must be followed by a line with a structuring command such as `@chapter`, `@section`, or `@unnumberedsubsec`.

Each `@node` line/structuring-command line combination must look either like this:

```
@node     Comments,  Minimum, Conventions, Overview
@comment  node-name, next,    previous,    up
@section Comments
```

or like this (without the `@comment` line):

```
@node Comments, Minimum, Conventions, Overview
@section Comments
```

(In this example, 'Comments' is the name of both the node and the section. The next node is called 'Minimum' and the previous node is called 'Conventions'. The 'Comments' section is within the 'Overview' node, which is specified by the 'Up' pointer.)

If a file has a 'Top' node, it must be called '`top`' or '`Top`' and be the first node in the file.

Incidently, the `makeinfo` command will create an Info file for hierarchically organized Texinfo file that lacks 'Next', 'Previous' and 'Up' pointers. Thus, if you can be sure that your Texinfo file will be formatted with `makeinfo`, you have no need for the 'update node' commands. (See Chapter 19 [Creating an Info File], page 153, for more information about `makeinfo`.) However, both `makeinfo` and the `texinfo-format-`... commands require that you insert menus in the file.

2.3.2 Other Updating Commands

In addition to the five major updating commands, Texinfo mode possesses several less frequently

used updating commands.

`M-x texinfo-insert-node-lines`
: Insert `@node` before the `@chapter`, `@section`, and other sectioning commands wherever it is missing throughout a region in a Texinfo file. With an argument (prefix argument, if interactive), the `texinfo-insert-node-lines` command not only inserts `@node` lines but also inserts the chapter or section titles as the names of the corresponding nodes; and it inserts their titles for node names in pre-existing `@node` lines that lack names. Since node names should be more concise than section or chapter titles, node names so inserted should be edited manually.

`M-x texinfo-multiple-files-update`
: Update nodes and menus in a document built from several separate files. With a prefix argument if called interactively (a non-`nil` 'make-master-menu' argument, if called non-interactively), create and insert a master menu in the outer file. With a numeric prefix argument if called interactively (a non-`nil` 'update-everything' argument if called non-interactively), first update all the menus and all the 'Next', 'Previous', and 'Up' pointers of all the included files before creating and inserting a master menu in the outer file. The `texinfo-multiple-files-update` command is described in the appendix on `@include` files. See Appendix B [Include Files], page 177.

`M-x texinfo-indent-menu-description`
: Indent every description in the menu following point to the specified column. You can use this command to give yourself more space for descriptions. With an argument (prefix argument, if interactive), the `texinfo-indent-menu-description` command indents every description in every menu in the region. However, this command does not indent the second and subsequent lines of a multi-line description.

`M-x texinfo-sequentially-update-node`
: Insert the names of the nodes immediately following and preceding the current node as the 'Next' or 'Previous' pointers regardless of those nodes' hierarchical level. This means that the 'Next' node of a subsection may well be the next chapter. Sequentially ordered nodes are useful for novels and other documents that you read through sequentially. (However, in Info, the `g* RET` command lets you look through the file sequentially, so sequentially ordered nodes are not strictly necessary.) With an argument (prefix argument, if interactive), the `texinfo-sequentially-update-node` command sequentially updates all the nodes in the region.

2.4 Formatting for Info

Texinfo mode provides several commands for formatting part or all of a Texinfo file for Info.

Often, when you are writing a document, you want to format only part of a file—that is, a region.

You can use either the `texinfo-format-region` or the `makeinfo-region` command to format a region.

`C-c C-e C-r`
`M-x texinfo-format-region`
`C-c C-m C-r`
`M-x makeinfo-region`
: Format the current region for Info.

You can use either the `texinfo-format-buffer` or the `makeinfo-buffer` command to format a whole buffer:

`C-c C-e C-b`
`M-x texinfo-format-buffer`
`C-c C-m C-b`
`M-x makeinfo-buffer`
: Format the current buffer for Info.

> After writing a Texinfo file, you can type `C-u C-c C-u m` or `C-u M-x texinfo-master-menu` to update all the nodes and menus and then type `C-c C-m C-b` or `M-x makeinfo-buffer` to create an Info file.

For the Info formatting commands to work, the file *must* include a line that has `@setfilename` in its header.

Not all systems support the `makeinfo`-based formatting commands.

See Chapter 19 [Creating an Info File], page 153, for details about Info formatting.

2.5 Formatting and Printing

Typesetting and printing a Texinfo file is a multi-step process in which you first create a file for printing (called a DVI file), and then you print the file. Optionally, also, you may create indices.

Often, when you are writing a document, you want to typeset and print only part of a file, to

see what it will look like. You can use the `texinfo-tex-region` and related commands for this purpose. Use the `texinfo-tex-buffer` command to format all of a buffer.

`C-c C-t C-r`
`M-x texinfo-tex-region`
: Run TEX on the region.

`C-c C-t C-b`
`M-x texinfo-tex-buffer`
: Run TEX on the buffer.

`C-c C-t C-i`
`M-x texinfo-texindex`
: Sort the indices of a Texinfo file formatted with `texinfo-tex-region` or `texinfo-tex-buffer`. You must run the `tex` command a second time after sorting the raw index files.

`C-c C-t C-p`
`M-x texinfo-tex-print`
: Print the file (or the part of the file) previously formatted with `texinfo-tex-buffer` or `texinfo-tex-region`.

For `texinfo-tex-region` or `texinfo-tex-buffer` to work, the file *must* start with a '`\input texinfo`' line and must include an `@settitle` line between start of header and end of header lines. The file must end with `@bye` on a line by itself.

See Chapter 18 [Printing Hardcopy], page 143, for a description of the other TEX related commands, such as `texinfo-texindex` and `tex-show-print-queue`.

2.6 Texinfo Mode Summary

In Texinfo mode, each set of commands has default keybindings that begin with the same keys. All the commands that are custom-created for Texinfo mode begin with `C-c`. The keys that follow are arranged mnemonically.

Insert Commands

The insert commands are invoked by typing `C-c` twice and then the first letter of the `@`-command to be inserted. (It might make more sense mnemonically to use `C-c C-i`, for 'custom insert', but `C-c C-c` is quick to type.)

`C-c C-c c`	Insert `‘@code’`.
`C-c C-c d`	Insert `‘@dfn’`.
`C-c C-c e`	Insert `‘@end’`.
`C-c C-c i`	Insert `‘@item’`.
`C-c C-c n`	Insert `‘@node’`.
`C-c C-c s`	Insert `‘@samp’`.
`C-c C-c v`	Insert `‘@var’`.
`C-c C-c {`	Insert braces.
`C-c C-c }`	Move out of enclosing braces.

Show Structure

The `texinfo-show-structure` command is often used within a narrowed region.

`C-c C-s`	List all the headings.

The Master Update Command

The `texinfo-master-menu` command creates a master menu; and can be used to update every node and menu in a file as well.

`C-c C-u m`
`M-x texinfo-master-menu`
: Create or update a master menu.
With an argument, first create or
update all nodes and regular menus.

Update Pointers

The update pointer commands are invoked by typing `C-c C-u` and then either `C-n` for `texinfo-update-node` or `C-e` for `texinfo-every-node-update`.

`C-c C-u C-n`	Update a node.
`C-c C-u C-e`	Update every node in the buffer.

Update Menus

The update menu commands are invoked by typing `C-c C-u` and then either `C-m` for `texinfo-make-menu` or `C-a` for `texinfo-all-menus-update`. You may precede a `C-c C-u C-a` so as to update both nodes and menus.

`C-c C-u C-m`	Make or update a menu.
`C-c C-u C-a`	Make or update all the menus in a buffer; with an argument, first update all the nodes.

Format for Info

The Info formatting commands that are written in Emacs Lisp are invoked by typing `C-c C-e` and then either `C-r` for a region or `C-b` for the whole buffer.

The Info formatting commands that are written in C and based on the `makeinfo` program are invoked by typing `C-c C-m` and then either `C-r` for a region or `C-b` for the whole buffer. Not all systems support the `makeinfo` program used by these commands.

Use the `texinfo-format`... commands:

`C-c C-e C-r`	Format the region.
`C-c C-e C-b`	Format the buffer.

Use `makeinfo`:

`C-c C-m C-r`	Format the region.
`C-c C-m C-b`	Format the buffer.
`C-c C-m C-l`	Recenter the `makeinfo` output buffer.
`C-c C-m C-k`	Kill the `makeinfo` formatting job.

Typeset and Print

The TEX typesetting and printing commands are invoked by typing `C-c C-t` and then another control command: `C-r` for `texinfo-tex-region`, `C-b` for `texinfo-tex-buffer`, and so on.

`C-c C-t C-r` Run TEX on the region.
`C-c C-t C-b` Run TEX on the buffer.
`C-c C-t C-i` Run `texindex`.
`C-c C-t C-p` Print the DVI file.
`C-c C-t C-q` Show the print queue.
`C-c C-t C-d` Delete a job from the print queue.
`C-c C-t C-k` Kill the current TEX formatting job.
`C-c C-t C-l` Recenter the output buffer.

Other Updating Commands

The 'other updating commands' do not have standard keybindings because they are rarely used.

`M-x texinfo-insert-node-lines`
Insert missing node lines using section titles as node names.

`M-x texinfo-multiple-files-update`
Update a multi-file document.

`M-x texinfo-indent-menu-description`
Indent descriptions.

`M-x texinfo-sequentially-update-node`
Insert node pointers in strict sequence.

3 Beginning a Texinfo File

Various pieces of information must be provided at the beginning of a Texinfo file, such as the name of the file and the title of the document.

Generally, the beginning of a Texinfo file has several parts:

1. The header, delimited by special comment lines, that includes the commands for naming the Texinfo file and telling TEX what definitions' file to use when processing the Texinfo file.
2. A short statement of what the file is about, with a copyright notice and copying permissions. This is enclosed in `@ifinfo` and `@end ifinfo` commands so that it appears only in the Info file.
3. A title page and copyright page, with a copyright notice and copying permissions. This is enclosed in `@titlepage` and `@end titlepage` commands and appears only in the printed manual.
4. The 'Top' node that contains a master menu for the whole Info file. The contents of this node appear only in the Info file.
5. Optionally, copying conditions for a program and a warranty disclaimer. The copying section will be followed by an introduction or else by the first chapter of the manual.

Since the copyright notice and copying permissions for the Texinfo document (in contrast to the copying permissions for a program) are in parts that appear only in the Info file or only in the printed manual, this information must be given twice.

3.1 Sample Texinfo File Beginning

The following sample shows what is needed.

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename name-of-info-file
@settitle name-of-manual
@setchapternewpage odd
@c %**end of header

@ifinfo
This file documents ...

Copyright @copyright{} year   copyright-owner
```

```
Permission is granted to ...
@end ifinfo


@c  This title page illustrates only one of the
@c  two methods of forming  a title page.

@titlepage
@title name-of-manual-when-printed
@subtitle subtitle-if-any
@subtitle second-subtitle
@author author

@c  The following two commands
@c  start the copyright page.
@page
@vskip 0pt plus 1filll
Copyright @copyright{} year  copyright-owner

Published by ...

Permission is granted to ...
@end titlepage



@node Top, Overview, (dir), (dir)

@ifinfo
This document describes ...
@end ifinfo

@menu
* Copying::          Your rights and freedoms.
* First Chapter::    Getting started ...
* Second Chapter::              ...
  <many more menu items here>
@end menu

@node    First Chapter, Second Chapter, top,      top
@comment node-name,     next,           previous, up
@chapter First Chapter
@cindex Reference to First Chapter
```

3.2 The Texinfo File Header

Texinfo files start with at least three lines that provide Info and TEX with necessary information. These are the `\input texinfo` line, the `@settitle` line, and the `@setfilename` line. If you want

to run TEX on just a part of the Texinfo File, you also have to surround the `@settitle` and `@setfilename` lines with start-of-header and end-of-header lines.

3.2.1 The First Line of a Texinfo File

Every Texinfo file that is to be the top-level input to TEX must begin with a line that looks like this:

```
\input texinfo   @c -*-texinfo-*-
```

The line serves two functions:

1. When the file is processed by TEX, the '`\input texinfo`' command tells TEX to load the macros needed for processing a Texinfo file. These are in a file called '`texinfo.tex`', which is usually located in the '`/usr/lib/tex/macros`' directory.
2. When the file is edited in GNU Emacs, the '`-*-texinfo-*-`' mode specification tells Emacs to use to Texinfo mode.

3.2.2 Start of Header

You should write a start-of-header line on the second line of a Texinfo file. Follow the start-of-header line with `@setfilename` and `@settitle` lines and, optionally, an `@setchapternewpage` command line, and then by an end-of-header line.

With these lines, you can format part of a Texinfo file for Info or typeset part for printing.

A start-of-header line looks like this:

```
@c %**start of header
```

The odd string of characters ('`%**`') is to ensure that no other comment is accidently taken for a start-of-header line.

3.2.3 @setfilename

In order to be made into an Info file, a Texinfo file must contain a line that looks like this:

```
@setfilename info-file-name
```

Write the `@setfilename` command at the beginning of a line followed by the Info file name.

The `@setfilename` line specifies the name of the Info file to be generated. Specify the name with an '`.info`' extension, to produce an Info file name such as '`texinfo.info`'.

The Info formatting commands ignore everything written before the `@setfilename` line, which is why the very first line of the the file (the `\input` line) does not need to be commented out. The `@setfilename` line is ignored when you typeset a printed manual.

3.2.4 @settitle

In order to be made into a printed manual, a Texinfo file must contain a line that looks like this:

```
@settitle title
```

Write the `@settitle` command at the beginning of a line followed by the title name. This tells TEX the title to use in a header or footer.

Conventionally, TEX formats a Texinfo file for double-sided output so as to print the title in the left-hand (even-numbered) page headings and the current chapter titles in the right-hand (odd-numbered) page headings. (TEX learns the title of each chapter from each `@chapter` command.) Page footers are not printed.

Even if you are printing in a single-sided style, TEX looks for an `@settitle` command line, in case you include the manual title in the heading.

The `@settitle` command should precede everything that generates actual output in TEX.

The title in the `@settitle` command is usually the same as the real title on the title page, but not always. It can be a shortened or expanded version of the title.

TEX prints page headings only for that text that comes after the `@end titlepage` command in the Texinfo file, or that comes after an `@headings` command that turns on headings. (See Section 3.4.6 [The `@headings` Command], page 38, for more information.)

You may, if you wish, create your own, customized headings and footings. See Appendix C [Page Headings], page 183, for a detailed discussion of this process.

3.2.5 `@setchapternewpage`

In a book or a manual, usually text is printed on both sides of the paper, chapters start on right-hand pages, and right-hand pages have odd numbers. But in short reports, text often is printed only on one side of the paper. Also in short reports, chapters sometimes do not start on new pages, but are printed on the same page as the end of the preceding chapter, after a small amount of vertical whitespace.

You can use the `@setchapternewpage` command with various arguments to specify how TEX should start chapters and whether it should typeset pages for printing on one or both sides of the paper (single-sided or double-sided printing).

Write the `@setchapternewpage` command at the beginning of a line followed by its argument.

For example, you would write the following to cause each chapter to start on a fresh odd-numbered page:

```
@setchapternewpage odd
```

You can specify one of three alternatives for the `@setchapternewpage` command:

`@setchapternewpage off`
: This tells TEX to typeset for double-sided printing, but not to start new chapters on new pages.

`@setchapternewpage on`
: This tells TEX to typeset for single-sided printing and to start new chapters on new pages. This is the form most often used for short reports.

  This alternative is the default.

`@setchapternewpage odd`
: This tells TEX to typeset for double-sided printing and to start new chapters on new,

odd-numbered pages (right-handed pages). This is the form most often used for books and manuals.

Texinfo does not have an `@setchapternewpage even` command.

(You can countermand or modify an `@setchapternewpage` command with an `@headings` command. See Section 3.4.6 [The `@headings` Command], page 38.)

At the beginning of a manual or book, pages are not numbered—for example, the title and copyright pages of a book are not numbered. By convention, table of contents pages are numbered with roman numerals and not in sequence with the rest of the document.

Since an Info file does not have pages, the `@setchapternewpage` command has no effect on it.

Usually, you do not use an `@setchapternewpage` command when you are printing single-sidedly, and you use the `@setchapternewpage odd` command when you are printing double-sidedly.

3.2.6 Paragraph Indenting

The Info formatting commands may insert spaces at the beginning of the first line of each paragraph, thereby indenting that paragraph. The `@paragraphindent` command specifies the indentation. Write `@paragraphindent` at the beginning of a line followed by either '`asis`' or a number. The template is:

`@paragraphindent` *indent*

The Info formatting commands indent according to the value of *indent*:

- If the value of *indent* is '`asis`', the Info formatting commands do not change the existing indentation.
- If the value of *indent* is 0, the Info formatting commands delete existing indentation.
- If the value of *indent* is greater than 0, the Info formatting commands indent the paragraph by that number of spaces.

The default value of *indent* is '`asis`'.

Write the `@paragraphindent` command before or shortly after the end of header line at the

beginning of a Texinfo file. (If you write the command between the start of header and end of header lines, the region formatting commands indent paragraphs as specified.)

The `makeinfo` utility and the two Emacs formatting commands, `texinfo-format-buffer` and `texinfo-format-region`, work somewhat differently.

- The `makeinfo` utility indents every paragraph that ought to be indented as specified by `@paragraphindent`.
- However, the `texinfo-format-buffer` and `texinfo-format-region` commands do *not* automatically indent paragraphs. These commands only indent paragraphs that are ended by an `@refill` command. The amount of indentation is specified by `@paragraphindent` in exactly the same way as with `makeinfo`. See Appendix E [Refilling Paragraphs], page 199, for more information about `@refill`.

3.2.7 End of Header

Follow the line that contains the `@settitle` or `@setchapternewpage` command with the end-of-header line.

An end-of-header line looks like this:

```
@c %**end of header
```

If you include the `@setchapternewpage` command between the start-of-header and end-of-header lines, TEX will typeset a region as that command specifies.

See Section 3.2.2 [Start of Header], page 29.

3.3 Summary and Copying Permissions for Info

Since the title page and the copyright page appear only in the printed copy of the manual, the same information must be inserted in a section that appears only in the Info file. This section usually contains a brief description of the contents of the Info file, a copyright notice, and copying permissions.

The copyright notice should read:

`Copyright` *year copyright-owner*

and be put on a line by itself.

Standard text for the copyright permissions is contained in the appendix. See Section 3.7.1 [‘`ifinfo`’ Copying Permissions], page 43, for the complete text.

3.4 The Title and Copyright Pages

A manual's name and author are usually printed on a title page. Sometimes copyright information is printed on the title page as well; more often, copyright information is printed on the back of the title page.

The title and copyright pages appear in the printed manual, but not in the Info file. Because of this, it is possible to use several slightly obscure TEX typesetting commands that cannot be used in an Info file. In addition, this part of the beginning of a Texinfo file contains the text of the copying permissions that will appear in the printed manual.

3.4.1 `@titlepage`

Start the material for the title page and following copyright page with `@titlepage` on a line by itself and end it with `@end titlepage` on a line by itself.

The `@end titlepage` command starts a new page and turns on page numbering. (See Appendix C [Page Headings], page 183, for details about the generation of page headings.) All the material that you want to appear on unnumbered pages should be put between the `@titlepage` and `@end titlepage` commands. By using the `@page` command you can force a page break within the region delineated by the `@titlepage` and `@end titlepage` commands and thereby create more than one unnumbered page. This is how the copyright page is produced. (The `@titlepage` command might perhaps have been better named the `@titleandadditionalpages` command, but that would have been rather long!)

Texinfo provides two methods for creating a title page. One method uses the `@titlefont`, `@sp`, and `@center` commands to generate a title page in which the words on the page are centered.

The second method uses the `@title`, `@subtitle`, and `@author` commands to create a title page

with black rules under the title and author lines and the subtitle text set flush to the right hand side of the page. With this method, you do not specify any of the actual formatting of the title page. You must specify the text you want, and Texinfo does the formatting. You may use either method.

3.4.2 `@titlefont`, `@center`, and `@sp`

You can use the `@titlefont`, `@sp`, and `@center` commands to create a title page for a printed document.

Use the `@titlefont` command to select a large font suitable for the title itself. For example:

```
@titlefont{Texinfo Manual}
```

Use the `@center` command at the beginning of a line to center the remaining text on that line. Thus,

```
@center @titlefont{Texinfo Manual}
```

centers the title, which in this example is "Texinfo Manual" printed in the title font.

Use the `@sp` command to insert vertical space. For example:

```
@sp 2
```

This inserts two blank lines on the printed page. (See Section 16.3 [`@sp`], page 139, for more information about the `@sp` command.)

A typical title page looks like the following:

```
@titlepage
@sp 10
@center @titlefont{name-of-manual-when-printed}
@sp 2
@center subtitle-if-any
@sp 2
@center author
```

The spacing of the example fits an 8 1/2 by 11 inch manual.

3.4.3 `@title`, `@subtitle`, and `@author`

You can use the `@title`, `@subtitle`, and `@author` commands to create a title page in which the vertical and horizontal spacing is done for you. This contrasts with the method described in the previous section, in which the `@sp` command is needed to adjust vertical spacing.

Write the `@title`, `@subtitle`, or `@author` commands at the beginning of a line followed by the title, subtitle, or author.

The `@title` command produces a line in which the title is set flush to the left-hand side of the page in a larger than normal font. The title is underlined with a black rule.

The `@subtitle` command sets subtitles in a normal-sized font flush to the right-hand side of the page.

The `@author` command sets the names of the author or authors in a middle-sized font flush to the left-hand side of the page on a line near the bottom of the title page. The names are underlined with a black rule that is thinner than the rule that underlines the title. (The black rule only occurs if the `@author` command line is followed by an `@page` command line.)

There are two ways to use the `@author` command: you can write the name or names on the remaining part of the line that starts with an `@author` command:

```
@author by Jane Smith and John Doe
```

or you can write the names one above each other by using two (or more) `@author` commands:

```
@author Jane Smith
@author John Doe
```

(Only the bottom name is underlined with a black rule.)

Thus, a typical title page looks like the following:

`@title` *name-of-manual-when-printed*

```
@subtitle subtitle-if-any
@subtitle second-subtitle
@author author
```

3.4.4 Copyright Page and Permissions

By international treaty, the copyright notice for a book should be either on the title page or on the back of the title page. The copyright notice should include the year followed by the name of the organization or person who owns the copyright.

When the copyright notice is on the back of the title page, that page is customarily not numbered. Therefore, in Texinfo, the information on the copyright page should be within `@titlepage` and `@end titlepage` commands.

Use the `@page` command to cause a page break. To push the copyright notice and the other text on the copyright page towards the bottom of the page, you can write a somewhat mysterious line after the `@page` command that reads like this:

```
@vskip 0pt plus 1filll
```

This is a TEX command that is not supported by the Info formatting commands. The `@vskip` command inserts white space. The '`0pt plus 1filll`' means to put in zero points of mandatory white space, and as much optional white space as needed to push the following text to the bottom of the page. Note the use of three '`l`'s in the word '`filll`'; this is the correct usage in TEX.

In a printed manual, the `@copyright{}` command generates a '`c`' inside a circle. (In Info, it generates '`(C)`'.) The copyright notice itself has the following legally defined sequence:

```
Copyright © year copyright-owner
```

It is customary to put information on how to get a manual after the copyright notice, followed by the copying permissions for the manual.

Note that permissions must be given here as well as in the summary segment within `@ifinfo` and `@end ifinfo` that immediately follows the header since this text appears only in the printed manual and the '`ifinfo`' text appears only in the Info file.

See Section 3.7 [Sample Permissions], page 42, for recommended permission text.

3.4.5 Heading Generation

An `@end titlepage` command (on a line by itself) not only marks the end of the title and copyright pages but it causes TEX to start generating page headings and page numbers.

To repeat what is said elsewhere (see Section 3.2.5 [`@setchapternewpage`], page 31), Texinfo has two standard heading formats, one for documents printed on one side of each sheet of paper (single-sided printing), and the other for documents printed on both sides of each sheet (double-sided printing). You can specify these formats in different ways:

- The conventional way is to write an `@setchapternewpage` command before the title page commands, and then have the `@end titlepage` command start generating headings in the manner desired. (See Section 3.2.5 [`@setchapternewpage`], page 31.)
- Alternatively, you can use the `@headings` command to prevent headings from being generated or to start them for either single or double-sided printing. See Section 3.4.6 [The `@headings` Command], page 38.
- Or, you may specify your own heading and footing format. See Appendix C [Page Headings], page 183, for detailed information about headings and footings.

Most documents are formatted with the standard single-sided or double-sided format, using `@setchapternewpage odd` for double-sided printing and no `@setchapternewpage` command for single-sided printing.

3.4.6 The `@headings` Command

The `@headings` command is rarely used. It specifies what kind of page headings and footings to print on each page. Usually, this is controlled by the `@setnewchapter` command. You need the `@headings` command only if the `@setnewchapter` command does not do what you want, or if you want to turn off pre-defined headings prior to defining your own.

There are four ways to use the `@headings` command:

`@headings off`
: Turns off printing of headings.

`@headings single`
: Turns on headings appropriate for single-sided printing.

`@headings double`
`@headings on`

Turns on headings appropriate for double-sided printing. The two commands, `@headings on` and `@headings double`, are synonymous.

To prevent TEX from generating any page headings, write `@headings off` on a line of its own immediately after the line containing the `@end titlepage` command. The `@headings off` command overrides the `@end titlepage` command, which would otherwise cause TEX to print headings.

You can also specify your own style of heading and footing. See Appendix C [Page Headings], page 183, for more information.

3.5 The Top Node and Master Menu

The 'Top' node is the node from which you enter the file from outside.

A 'Top' node should contain a brief description of the file and an extensive, master menu for the whole Info file. The contents of this node should appear only in the Info file; none of it should appear in printed output, so enclose it between `@ifinfo` and `@end ifinfo` commands. (TEX does not print either an `@node` line or a menu; they appear only in Info; strictly speaking, you do not have to enclose these parts between `@ifinfo` and `@end ifinfo`, but it is simplest to do so. See Chapter 17 [Conditionally Visible Text], page 141.)

Sometimes, you will want to place an `@top` line containing the title of the document after the `@node` line, followed by a short summary. This helps the reader understand what the Info file is about.

For example, the beginning of the Top node of this manual looks like this:

```
...
@end titlepage

@ifinfo
@node Top, Copying, (dir), (dir)
@top Texinfo

Texinfo is a documentation system...
...
@end ifinfo
```

```
@menu
* Copying::                 Texinfo is freely
                             redistributable.
* Overview::                What is Texinfo?
...
@end menu
```

In a 'Top' node, the 'Previous', and 'Up' nodes usually refer to the top level directory of the whole Info system, which is called '`(dir)`'.

3.5.1 Parts of a Master Menu

A *master menu* is a detailed main menu listing all the nodes in a file.

A master menu is enclosed in `@menu` and `@end menu` commands and does not appear in the printed document.

Generally, a master menu is divided into parts.

- The first part contains the major nodes in the Texinfo file: the nodes for the chapters, chapter-like sections, and the appendices.
- The second part contains nodes for the indices.
- The third and subsequent parts contain a listing of the other, lower level nodes, often ordered by chapter. This way, rather than go through an intermediary menu, an inquirer can go directly to a particular node when searching for specific information. These menu items are not required; add them if you think they are a convenience.

Each section in the menu can be introduced by a descriptive line. So long as the line does not begin with an asterisk, it will not be treated as a menu item. (See Section 6.4 [Making Menus], page 61, for more information.)

For example, the master menu for this manual looks like the following (but has many more entries):

```
@menu
* Copying::             Texinfo is freely
                          redistributable.
* Overview::            What is Texinfo?
* Texinfo Mode::        Special features in GNU Emacs.
```

```
...
...
* Command and Variable Index::
                        An item for each @-command.
* Concept Index::       An item for each concept.

 --- The Detailed Node Listing ---

Overview of Texinfo

* Info Files::          What is an Info file?
* Printed Manuals::     Characteristics of
                          a printed manual.
...
...

Using Texinfo Mode

* Info on a Region::    Formatting part of a file
                          for Info.
...
...
@end menu
```

3.6 Software Copying Conditions

If the Texinfo file has a section containing the "General Public License" and the distribution information and a warranty disclaimer for the software that is documented, this section usually follows the 'Top' node. The General Public License is very important to Project GNU software. It ensures that you and others will continue to have a right to use and share the software.

The copying and distribution information and the disclaimer are followed by an introduction or else by the first chapter of the manual.

Although an introduction is not a required part of a Texinfo file, it is very helpful. Ideally, it should state clearly and concisely what the file is about and who would be interested in reading it. In general, an introduction would follow the licensing and distribution information, although sometimes people put it earlier in the document. Usually, an introduction is put in an `@unnumbered` section. (See Section 5.5 [The `@unnumbered` and `@appendix` Commands], page 52.)

3.7 Sample Permissions

Texinfo files should contain sections that tell the readers that they have the right to copy and distribute the Info file, the printed manual, and any accompanying software. Here are samples containing the standard text of the Free Software Foundation copying permission notice for an Info file and printed manual.

See section "Distribution" in *The GNU Emacs Manual*, for an example of the text that could be used in the software Distribution, General Public License, and NO WARRANTY sections of a document.

In a Texinfo file, the first `@ifinfo` section usually begins with a line that says what the file documents. This is what a person reading the unprocessed Texinfo file or using the advanced Info command `g *` sees first. See Info file '`info`', node '`Expert`', for more information. (A reader using the regular Info commands will usually start reading at the first node and skip this first section, which is not in a node.)

In the `@ifinfo` section, the summary sentence should be followed by a copyright notice and then by the copying permission notice. One of the copying permission paragraphs is enclosed in `@ignore` and `@end ignore` commands. This paragraph states that the Texinfo file can be processed through TEX and printed, provided the printed manual carries the proper copying permission notice. This paragraph is not made part of the Info file since it is not relevant to the Info file; but it is a mandatory part of the Texinfo file since it permits people to process the Texinfo file in TEX.

In the printed manual, the Free Software Foundation copying permission notice follows the copyright notice and publishing information and is located within the region delineated by the `@titlepage` and `@end titlepage` commands. The copying permission notice is exactly the same as the notice in the `@ifinfo` section except that the paragraph enclosed in `@ignore` and `@end ignore` commands is not part of the notice.

To make it simple to copy the permission notice into each section of the Texinfo file, the complete permission notices for each section are reproduced in full below.

Note that you may have to specify the correct name of a section mentioned in the permission notice. For example, in the *GDB Manual*, the name of the section referring to the General Public License is called the "GDB General Public License", but in the sample shown below, that section is referred to generically as the "General Public License".

3.7.1 ‘ifinfo’ Copying Permissions

In the `@ifinfo` section of the Texinfo file, the standard Free Software Foundation permission notices reads as follows:

```
This file documents ...

Copyright 1988 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim
copies of this manual provided the copyright notice and
this permission notice are preserved on all copies.

@ignore
Permission is granted to process this file through TeX
and print the results, provided the printed document
carries a copying permission notice identical to this
one except for the removal of this paragraph (this
paragraph not being relevant to the printed manual).

@end ignore
Permission is granted to copy and distribute modified
versions of this manual under the conditions for
verbatim copying, provided also that the sections
entitled ‘‘Distribution’’ and ‘‘General Public License’’
are included exactly as in the original, and provided
that the entire resulting derived work is distributed
under the terms of a permission notice identical to this
one.

Permission is granted to copy and distribute
translations of this manual into another language, under
the above conditions for modified versions, except that
the sections entitled ‘‘Distribution’’ and ‘‘General
Public License’’ may be included in a translation
approved by the author instead of in the original
English.
```

3.7.2 Titlepage Copying Permissions

In the `@titlepage` section of the Texinfo file, the standard Free Software Foundation copying permission notice follows the copyright notice and publishing information. The standard phrasing is:

```
Permission is granted to make and distribute verbatim
```

4 Ending a Texinfo File

The end of a Texinfo file should include the commands that create indices and generate detailed and summary tables of contents. And it must include the `@bye` command that marks the last line that TEX processes.

For example,

```
@node    Concept Index,     , Variables Index, Top
@comment node-name,     next, previous,      up
@unnumbered Concept Index

@printindex cp

@contents
@bye
```

4.1 Index Menus and Printing an Index

To print an index means to include it as part of a manual or Info file. This does not happen automatically just because you use `@cindex` or other index-entry generating commands in the Texinfo file; those just cause the raw data for the index to be accumulated. To generate an index, you must include the `@printindex` command at the place in the document where you want the index to appear. Also, as part of the process of creating a printed manual, you must run a program called `texindex` (see Chapter 18 [Printing Hardcopy], page 143) to sort the raw data to produce a sorted index file. The sorted index file is what will actually be used to print the index.

Texinfo offers six different types of predefined index: the concept index, the function index, etc. (See Section 15.2 [Predefined Indices], page 131.) Each index type has a two-letter name. You may merge indices, or put them into separate sections (See Section 15.4 [Combining Indices], page 134.).

The `@printindex` command takes a two-letter index name, reads the corresponding sorted index file and formats it appropriately into an index.

The `@printindex` command does not generate a chapter heading for the index. Consequently, you should precede the `@printindex` command with a suitable section or chapter command (usually `@unnumbered`) to supply the chapter heading and put the index into the table of contents. Precede the `@unnumbered` command with an `@node` line. For example,

```
@node Variables Index, Concept Index, Function Index, Top
@comment    node-name,          next,       previous,  up
@unnumbered Variable Index

@printindex vr

@node     Concept Index,     , Variables Index, Top
@comment      node-name, next,        previous,  up
@unnumbered Concept Index

@printindex cp

@summarycontents
@contents
@bye
```

(Readers often prefer that the concept index come last in a book, since that makes it easiest to find.)

4.2 Generating a Table of Contents

The `@chapter`, `@section`, and other structuring commands supply the information to make up a table of contents, but they do not cause an actual table to appear in the manual. To do this, you must use the `@contents` and `@summarycontents` commands.

`@contents`

Generate a table of contents in a printed manual, including all chapters, sections, subsections, etc., as well as appendices and unnumbered chapters. (Headings generated by the `@heading` series of commands do not appear in the table of contents.) The `@contents` command should be written on a line by itself.

`@shortcontents`

`@summarycontents`

(`@summarycontents` is a synonym for `@shortcontents`; the two commands are exactly the same.)

Generate a short or summary table of contents that lists only the chapters (and appendices and unnumbered chapters); sections, subsections and subsubsections are omitted. Write the `@shortcontents` command on a line by itself immediately before the `@contents` command. Only a long manual needs a short table of contents in addition to the full table of contents.

The table of contents commands automatically generate a chapter-like heading at the top of the

first table of contents page. Write tables of contents commands at the very end of a Texinfo file, just before the `@bye` command, following any index sections—anything in the Texinfo file after the table of contents commands will be omitted from the table of contents. When you print a manual with a table of contents, the table of contents will be printed last and numbered with roman numerals. You need to place those pages in their proper place, after the title page, yourself. (This is the only collating you need to do for a printed manual. The table of contents is printed last because it is generated after the rest of the manual is typeset.)

Here is an example of where to write table of contents commands:

```
indices...
@shortcontents
@contents
@bye
```

Since an Info file uses menus instead of tables of contents, the Info formatting commands ignore the `@contents` and `@shortcontents` commands.

4.3 `@bye` File Ending

An `@bye` command terminates TEX or Info formatting. None of the formatting commands see any of the file following `@bye`. The `@bye` command should be on a line by itself.

Optionally, you may follow an `@bye` line with a local variables list. See Section 18.4 [Using Local Variables and the Compile Command], page 147, for more information.

5 Chapter Structuring

The chapter structuring commands divide a document into a hierarchy of chapters, sections, subsections, and subsubsections. These commands generate large headings; they also provide information for the table of contents of a printed manual (see Section 4.2 [Generating a Table of Contents], page 46).

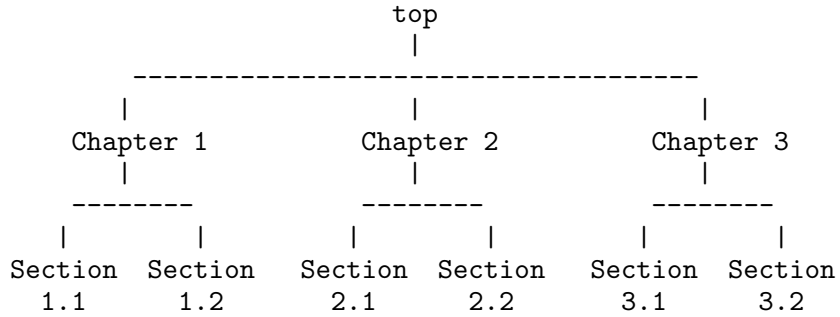
The chapter structuring commands do not create an Info node structure, so normally you should put an `@node` command immediately before each chapter structuring command (see Chapter 6 [Nodes and Menus], page 57). The only time you are likely to use the chapter structuring commands without using the node structuring commands is if you are writing a document that contains no cross references and will never be transformed into Info format.

It is unlikely that you will ever write a Texinfo file that is intended only as an Info file and not as a printable document. If you do, you might still use chapter structuring commands to create a heading at the top of each node—but you don't have to.

5.1 Tree Structure of Sections

A Texinfo file is usually structured like a book with chapters, sections, subsections, and the like. This structure can be visualized as a tree (or rather as an upside-down tree) with the root at the top and the levels corresponding to chapters, sections, subsection, and subsubsections. In Info format, 'Next' and 'Previous' pointers of a node usually lead to other nodes at the same level; an 'Up' pointer usually leads to a node at the level above; and a 'Menu' leads to nodes at a level below. Cross references can point to nodes at any level. See Chapter 7 [Cross References], page 67.

Here is a diagram that shows a Texinfo file with three chapters, each of which has two sections.



In a Texinfo file that has this structure, the beginning of Chapter 2 looks like this:

```
@node    Chapter 2,  Chapter 3, Chapter 1, top
@chapter Chapter 2
```

The chapter structuring commands are described in the sections that follow; the `@node` and `@menu` commands are described in the following chapter (see Chapter 6 [Nodes & Menus], page 57).

5.2 Types of Structuring Command

The chapter structuring commands fall into four groups or series, each of which contains structuring commands corresponding to the hierarchical levels of chapters, sections, subsections, and subsubsections.

The four groups are the `@chapter` series, the `@unnumbered` series, the `@appendix` series, and the `@heading` series.

Each command produces titles that have a different appearance on the printed page or Info file; only some of the commands produce titles that are listed in the table of contents of a printed book or manual.

- The `@chapter` and `@appendix` series of commands produce numbered or lettered entries both in the body of a printed work and in its table of contents.
- The `@unnumbered` series of commands produce unnumbered entries both in the body of a printed work and in its table of contents. The `@top` command, which has a special use, is a member of this series. See Section 5.3 [`@top`], page 51.
- The `@heading` series of commands produce unnumbered headings that do not appear in a table of contents. The heading commands never start a new page.
- The `@majorheading` command produces results similar to using the `@chapheading` command but generates a larger vertical whitespace before the heading.
- When an `@setchapternewpage` command says to do so, the `@chapter`, `@unnumbered`, and `@appendix` commands start new pages in the printed manual; the `@heading` commands do not.

Here are the four groups of chapter structuring commands:

Numbered In contents	Unnumbered In contents	Lettered and numbered In contents	No new pages Unnumbered Not in contents
	`@top`		`@majorheading`
`@chapter`	`@unnumbered`	`@appendix`	`@chapheading`
`@section`	`@unnumberedsec`	`@appendixsec`	`@heading`
`@subsection`	`@unnumberedsubsec`	`@appendixsubsec`	`@subheading`
`@subsubsection`	`@unnumberedsubsubsec`	`@appendixsubsubsec`	`@subsubheading`

5.3 `@top`

A special sectioning command, `@top`, has been created for use with the `@node Top` line. The `@top` command tells `makeinfo` that it marks the 'Top' node in the file. It provides the information that `makeinfo` needs to insert node pointers automatically. Write the `@top` at the beginning of the line immediately following the `@node Top` line. Write the title on the remaining part of the same line as the `@top` command.

In Info, the `@top` command causes the title to appear on a line by itself, with a line of asterisks inserted underneath.

In TEX and `texinfo-format-buffer`, the `@top` command is merely a synonym for `@unnumbered`. Neither of these formatters require an `@top` command, and do nothing special with it. You can use `@chapter` (or some other sectioning command) after the `@node Top` line when you are formatting with `texinfo-format-buffer`; you can do the same when you are using the Texinfo updating commands to create or update pointers and menus.

Whatever sectioning command follows an `@node Top` line, whether it be `@top` or `@chapter`, the `@node Top` line and the immediately following line must be enclosed between `@ifinfo` and `@end ifinfo` command. (See Chapter 17 [Conditionals], page 141.) This prevents the title from appearing in printed output. The `@ifinfo` command should be written before the node line and the `@end ifinfo` command should be written after the `@top` or other sectioning command and after any summary the node contains. (You can write the `@end ifinfo` command after the `@end menu` command if you like.)

5.4 `@chapter`

`@chapter` identifies a chapter in the document. Write the command at the beginning of a line and follow it on the same line by the title of the chapter.

For example, this chapter in this manual is entitled "Chapter Structuring"; the `@chapter` line looks like this:

```
@chapter Chapter Structuring
```

In TEX, the `@chapter` command creates a chapter in the document, specifying the chapter title. The chapter is numbered automatically.

In Info, the `@chapter` command causes the title to appear on a line by itself, with a line of asterisks inserted underneath. Thus, in Info, the above example produces the following output:

```
Chapter Structuring
*******************
```

5.5 `@unnumbered`, `@appendix`

Use the `@unnumbered` command to create a chapter that appears in a printed manual without chapter numbers of any kind. Use the `@appendix` command to create an appendix in a printed manual that is labelled by letter instead of by number.

For Info file output, the `@unnumbered` and `@appendix` commands are equivalent to `@chapter`: the title is printed on a line by itself with a line of asterisks underneath. (See Section 5.4 [`@chapter`], page 51.)

To create an appendix or an unnumbered chapter, write an `@appendix` or `@unnumbered` command at the beginning of a line and follow it on the same line by the title, as you would if you were creating a chapter.

5.6 `@majorheading`, `@chapheading`

The `@majorheading` and `@chapheading` commands put chapter-like headings in the body of a document.

However, neither command causes TEX to produce a numbered heading or an entry in the table of contents; and neither command causes TEX to start a new page in a printed manual.

In TEX, a `@majorheading` command generates a larger vertical whitespace before the heading than a `@chapheading` command but is otherwise the same.

For Info file output, the `@majorheading` and `@chapheading` commands commands are equivalent to `@chapter`: the title is printed on a line by itself with a line of asterisks underneath. (See Section 5.4 [`@chapter`], page 51.)

5.7 `@section`

In a printed manual, an `@section` command identifies a numbered section within a chapter. The section title appears in the table of contents. In Info, an `@section` command provides a title for a segment of text, underlined with '`=`'.

To create a section, write the `@section` command at the beginning of a line and follow it on the same line by the section title.

Thus,

```
@section This is a section
```

produces

```
This is a section
==================
```

in Info.

5.8 `@unnumberedsec`, `@appendixsec`, `@heading`

The `@unnumberedsec`, `@appendixsec`, and `@heading` commands are, respectively, the unnumbered, appendix-like, and heading-like equivalents of the `@section` command. (See Section 5.7 [`@section`], page 53.)

```
@appendixsec
@appendixsection
```

`@appendixsection` is a longer spelling of the `@appendixsec` command; it is a synonym for the `@appendixsec` command.

Conventionally, the `@appendixsec` or `@appendixsection` command is used only within appendices.

`@unnumberedsec`
: The `@unnumberedsec` command may be used within an unnumbered chapter or within a regular chapter or appendix to provide an unnumbered section.

`@heading`
: You may use the `@heading` command anywhere you wish for a section-style heading that will not appear in the table of contents.

5.9 The `@subsection` Command

Subsections are to sections as sections are to chapters. (See Section 5.7 [`@section`], page 53.) In Info, subsection titles are underlined with '`-`'. For example,

```
@subsection This is a subsection
```

produces

```
This is a subsection
--------------------
```

In a printed manual, subsections are listed in the table of contents and are numbered three levels deep.

5.10 The `@subsection`-like Commands

The `@unnumberedsubsec`, `@appendixsubsec`, and `@subheading` commands are, respectively, the unnumbered, appendix-like, and heading-like equivalents of the `@subsection` command. (See Section 5.9 [`@subsection`], page 54.)

In Info, the `@subsection`-like commands generate a title underlined with hyphens. In a printed manual, an `@subheading` command produces a heading like that of a subsection except that it is not numbered and does not appear in the table of contents. Similarly, an `@unnumberedsubsec` command produces an unnumbered heading like that of a subsection and an `@appendixsubsec` command produces a subsection-like heading labelled with a letter and numbers

5.11 The 'subsub' Commands

The fourth and lowest level sectioning commands in Texinfo are the 'subsub' commands. They are:

`@subsubsection`
: Subsubsections are to subsections as subsections are to sections. (See Section 5.9 [`@subsection`], page 54.) In a printed manual, subsubsection titles appear in the table of contents and are numbered four levels deep.

`@unnumberedsubsubsec`
: Unnumbered subsubsection titles appear in the table of contents of a printed manual, but lack numbers. Otherwise, unnumbered subsubsections are the same as subsubsections. In Info, unnumbered subsubsections look exactly like ordinary subsubsections.

`@appendixsubsubsec`
: Conventionally, appendix commands are used only for appendices and are lettered and numbered appropriately in a printed manual. In Info, appendix subsubsections look exactly like ordinary subsubsections.

`@subsubheading`
: The `@subsubheading` command may be used anywhere that you need a small heading that will not appear in the table of contents. In Info, subsubheadings look exactly like ordinary subsubsection headings.

In Info, 'subsub' titles are underlined with periods. For example,

```
@subsubsection This is a subsubsection
```

produces

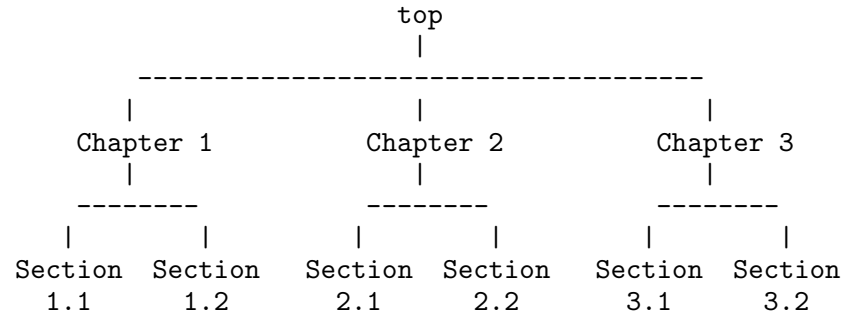
```
This is a subsubsection
.......................
```

6 Nodes and Menus

Most Texinfo files are organized hierarchically like books, with chapters, sections, subsections, and subsubsections. Such a hierarchy is tree-like; the chapters are the major limbs from which the sections branch out. In a conventional diagram, however, such a hierarchy is drawn with the "root" at the top and the "leaves" at the bottom—as an upside-down tree. The root node is called the 'Top' node, and 'Up' pointers carry you closer to the root.

6.1 Node and Menu Illustration

Here is a copy of the diagram shown earlier that illustrates a Texinfo file with three chapters, each of which contains two sections.



In a Texinfo file that has this organization, you would write the beginning of the node for Chapter 2 like this:

```
@node     Chapter 2,  Chapter 3, Chapter 1, top
@comment  node-name,  next,      previous,  up
```

To go to Sections 2.1 and 2.2 using Info, you need a menu inside of Chapter 2 that says:

```
@menu
* Sect. 2.1::    Description of this section.
* Sect. 2.2::
@end menu
```

You would locate this menu inside Chapter 2, after the beginning of the chapter and before Section 2.1.

The node for Sect. 2.1 will look like this:

```
@node     Sect. 2.1, Sect. 2.2, Chapter 2, Chapter 2
@comment  node-name, next,      previous,  up
```

Usually, an `@node` command and a chapter structuring command are used in sequence, along with indexing commands. (The updating commands require this sequence. See Section 2.3.1 [Updating Requirements], page 20.) Also, you may want to follow the `@node` line with a comment line that reminds you which pointer is which. For example, the beginning of the node for the chapter on ending a file looks like this:

```
@node    Ending a File, Structuring, Beginning a File, Top
@comment node-name,     next,        previous,         up
@chapter Ending a Texinfo File
@cindex Ending a Texinfo file
@cindex Texinfo file ending
@cindex File ending
```

The following two sections describe the `@node` and `@menu` commands in detail.

6.2 `@node`

`@node` defines the beginning of a new node in the Info output file. (See Info file '`info`', node '`Top`'.) Write the command at the beginning of a line, followed by four arguments, separated by commas, that make up the rest of the line. These arguments are the name of the node, and the names of the 'Next', 'Previous', and 'Up' pointers, in that order. You may insert spaces before each pointer if you wish. The spaces are ignored.

In TEX, `@node` is nearly ignored. It generates nothing visible. Its only function is to identify the name to use for cross references to the chapter or section which follows the `@node` command and which makes up the body of the node. (Cross references, such as the one following this sentence, are made with `@xref` and its related commands. See Chapter 7 [Cross References], page 67.)

In general, an `@node` line is followed immediately by a chapter-structuring command such as `@chapter`, `@section`, `@subsection`, or `@subsubsection`. (See Section 5.2 [Types of Structuring Command], page 50.)

The name of the node identifies the node. The pointers, which enable you to reach other nodes,

consist of the names of those nodes.

All the node names for a single Info file must be unique. Duplications confuse the Info movement commands. This means, for example, that if you end each chapter with a summary, you must name every summary node differently. You may, however, duplicate section titles (although this practice may confuse a reader).

Try to pick node names that are informative but short. In the Info file, the file name, node name, and pointer names are all inserted on one line, which may run into the right edge of the window. (This does not cause a problem with Info, but is ugly.)

By convention, node names are capitalized just as they would be for section or chapter titles.

Caution: Do not use any of the Texinfo `@`-commands in a node name; these commands confuse Info.

Do not use commas within a node name; a comma terminates the node name.

Do not use apostrophes within a node name; an apostrophe confuses the internals of '`texinfo.tex`'.

Pointer names must be the names of nodes defined elsewhere. It does not matter whether pointers are before or after the node that refers to them.

Normally, a node's 'Up' pointer should contain the name of the node whose menu mentions that node. The node's 'Next' pointer should contain the name of the node that follows that node and its 'Previous' pointer should contain the name of the node that precedes it in that menu. When a node's 'Up' node is the same as its 'Previous' node, both node pointers should name the same node.

6.2.1 Writing a Node Line

The easiest way to write a node line is to write `@node` at the beginning of a line and then the name of the node. You can use update node commands provided by Texinfo mode to insert the names of the pointers; or you can leave the pointers out of the Texinfo file and have `makeinfo` insert pointers into the Info file it creates. (See Chapter 2 [Texinfo Mode], page 15, and Section 6.3 [makeinfo Pointer Creation], page 60.)

Alternatively, you may insert the 'Next', 'Previous', and 'Up' pointers yourself. If you do this, you may find it helpful to use the Texinfo mode keyboard command `C-c C-c n`. This command inserts '`@node`' and a comment line listing the names of the pointers in their proper order. The comment line helps you keep track of which arguments are for which pointers. This template is especially useful if you are not familiar with Texinfo.

If you wish, you can ignore node lines altogether in your first draft and then use the `texinfo-insert-node-lines` command to create node lines for you. However, this practice is not recommended. It is better to name the node itself at the same time you write a section so you can easily make cross references. A large number of cross references are an especially important feature of a good Info file.

After you have inserted a node line, you should immediately write an `@`-command for the chapter or section and insert its name. Next (and this is important!), put in several index entries. Usually, you will find at least two and often as many as four or five ways of referring to the node in the index. Use them all. This will make it much easier for people to find the node.

The top node of the file (which must be named '`top`' or '`Top`') should have as its 'Up' and 'Previous' nodes the name of a node in another file, where there is a menu that leads to this file. Specify the file name in parentheses. If the file is to be installed directly in the Info directory file, use '`(dir)`' as the parent of the 'Top' node; this is short for '`(dir)top`', and specifies the 'Top' node in the '`dir`' file, which contains the main menu for Info. For example, the 'Top' node line of this manual looks like this:

```
@node Top, Overview, (dir), (dir)
```

(You may use the Texinfo updating commands or the `makeinfo` utility to insert these 'Next' and '`(dir)`' pointers automatically.)

See Chapter 20 [Installing an Info File], page 161, for more information about installing an Info file in the '`info`' directory.

6.3 Creating Pointers with `makeinfo`

The `makeinfo` program has a feature for automatically creating node pointers for a hierarchically organized file that lacks them.

When you take advantage of this feature, you do not have to write the 'Next', 'Previous', and

'Up' pointers after the name of a node. However, you must write a sectioning command, such as `@chapter` or `@section`, on the line immediately following each truncated `@node` line. (You cannot write a comment line after a node line; the section line must follow it immediately.)

In addition, you must follow the 'Top' node line with a line beginning with `@top` to mark the 'Top' node in the file. See Section 5.3 [@top], page 51.

Finally, you must write the name of each node (except for the 'Top' node) in a menu that is one or more hierarchical levels above the node's hierarchical level.

This node pointer insertion feature in `makeinfo` is an alternative to the menu and pointer creation and update commands in Texinfo mode. (See Section 2.3 [Updating Nodes and Menus], page 18.) It is especially helpful to people who do not use GNU Emacs to for writing Texinfo documents.

6.4 `@menu`

The `@menu` command is used to create *menus*, which contain pointers to subordinate nodes. In Info, you use menus to go to such nodes. Menus have no effect in printed manuals and do not appear in them.

By convention, a menu is put at the end of a node. This way, it is easy for someone using Info to find the menu, using the `M->` (`end-of-buffer`) command.

A node that has a menu should not contain much text. If you have a lot of text and a menu, move most of the text into a new subnode—all but a few lines. Otherwise, a reader with a terminal that displays only a few lines may miss the menu and its associated text. As a practical matter, you should locate a menu within 20 lines of the beginning of the node.

The short text before a menu may look awkward in a printed manual. To avoid this, you can write a menu near the beginning of its node and follow the menu by an `@node` line and an `@heading` line within `@ifinfo` and `@end ifinfo`. This way, the menu, node line, and title appear only in the Info file, not the printed document.

The preceding two paragraphs follow an Info-only menu, node line, and heading, and look like this:

```
@menu
* Menu Location::               Put a menu in a short node.
* Menu Item::                   How to write a menu item.
* Menu Example::                A menu example.
@end menu

@node Menu Location
@ifinfo
@subheading Menus Need Short Nodes
@end ifinfo
```

See the beginning of the "Cross References" chapter in the Texinfo source for this document for another example this procedure.

6.4.1 Writing a Menu Item

In a menu, every line that begins with a '`* `' is a menu item. (Note the space after the asterisk.) A line that does not start with a '`* `' can appear in the menu but is not a menu item, just a comment.

A menu item has three parts, only the second of which is required:

1. The menu item name.
2. The name of the node.
3. A description of the item.

A menu item looks like this:

```
* Item name: Node name.      Description.
```

Follow the menu item name with a single colon and follow the node name with tab, comma, period, or newline.

In Info, a user can select a node with the `m` (`Info-menu`) command. The menu item name is what the user types after the `m` command.

If the menu item name and the node name are the same, you can write the name immediately after the asterisk and space at the beginning of the line and follow the name with two colons.

For example, write

```
* Name::
```

instead of

```
* Name: Name.
```

You should use the node name for the menu item name whenever possible, since it reduces visual clutter in the menu.

The third part of a menu entry is a short descriptive phrase or sentence. Menu item names and node names are often short; the description explains to the reader what the node is about. The description, which is optional, can spread over two or more lines. A useful description complements the node name rather than repeating it.

6.4.2 A Menu Example

A menu looks like this in Texinfo:

```
@menu
* Menu item name: Node name.    A short description.
* Node name::                   This form is preferred.
@end menu
```

This produces:

```
* menu:

* Menu item name: Node name.    A short description.
* Node name::                   This form is preferred.
```

Here is an example as you might see it in a Texinfo file:

```
@menu
Larger Units of Text

* Files::                 All about handling files.
* Multiples: Buffers.     Multiple buffers; editing
                            several files at once.
@end menu
```

This produces:

```
* menu:
Larger Units of Text

* Files::                  All about handling files.
* Multiples: Buffers.      Multiple buffers; editing
                             several files at once.
```

In this example, the menu has two entries. '`Files`' is both a menu item name and the name of the node referred to by that item. In the other entry, '`Multiples`' is the item name, and it refers to the node named '`Buffers`'.

Since no file name is specified with either '`Files`' or '`Buffers`', they must be the names of nodes in the same Info file. (See Section 6.5 [Referring to Other Info Files], page 64.)

The line '`Larger Units of Text`' is a comment.

6.5 Referring to Other Info Files

You can refer to nodes in other Info files by writing the file name in parentheses just before the node name. In this case, you should use the three-part menu item format, which saves the reader from having to type the file name.

If you do not list the node name, but only name the file, then Info presumes that you are referring to the '`Top`' node.

The format looks like this:

```
@menu
* first-item:(filename)nodename.        description
* second-item:(filename)second-node.    description
@end menu
```

The '`dir`' top level directory for the Info system has menu entries that take you directly to the '`Top`' nodes of each Info document. (See Chapter 20 [Installing an Info File], page 161.)

For example,

```
...
* Info: (info).         Documentation browsing system.
* Emacs: (emacs).       The extensible, self-documenting
                        text editor.
...
```

To refer directly to the '`Outlining`' and '`Rebinding`' nodes in the *Emacs Manual*, you would write a menu similar to the following:

```
@menu
* Outlining: (emacs)Outline Mode. The major mode for
                                  editing outlines.
* Rebinding: (emacs)Rebinding.    How to redefine the
                                  meaning of a key.
@end menu
```

6.6 Describing Command Invocation

Documents that describe programs such as Emacs, GCC, and GAWK that are invoked from a shell each contain a section that describes the command line arguments. Unfortunately, the node names and titles for these sections are all different! This makes it hard to search for the section.

We suggest adopting the convention of naming such sections with a phrase beginning with the word 'Invoking ...', as in 'Invoking Emacs'; this way users can find the section easily.

7 Making Cross References

Cross references are used to refer the reader to other parts of the same or different Texinfo files. In Texinfo, nodes are the points to which cross references can refer.

Often, but not always, a printed document should be designed so that it can be read sequentially. People tire of flipping back and forth to find information that should be presented to them as they need it.

However, in any document, some information will be too detailed for the current context, or incidental to it; use cross references to provide access to such information. Also, an on-line help system or a reference manual is not like a novel; few read such documents in sequence from beginning to end. Instead, people look up what they need. For this reason, such creations should contain many cross references to help readers find other information that they may not have read.

In a printed manual, a cross reference creates a page reference, unless it is to another manual altogether, in which case it names that manual.

In Info, a cross reference creates an entry that you can follow using the Info '`f`' command. (See Info file '`info`', node '`Help-Adv`'.)

The various cross reference commands use nodes to define cross reference locations. This is evident in Info, in which a cross reference takes you to the specified node. TEX also uses nodes to define cross reference locations, but the action is less obvious. When TEX generates a DVI file, it records nodes' page numbers and uses the page numbers in making references. Thus, if you are writing a manual that will only be printed, and will not be used on-line, you must nonetheless write `@node` lines to name the places to which you make cross references.

7.1 Different Cross Reference Commands

There are several different cross reference commands:

`@xref` Used to start a sentence in the printed manual saying 'See ...' or an entry in the Info file saying '`*Note` ...'.

`@ref` Used within or, more often, at the end of a sentence; produces just the reference in the printed manual without a preceding 'See'.

`@pxref` Used within parentheses to make a reference that starts with a lower case 'see' within the printed manual. ('`p`' is for 'parenthesis'.)

`@inforef` Used to make a reference to an Info file for which there is no printed manual.

(The `@cite` command is used to make references to books and manuals for which there is no corresponding Info file and, therefore, no node to which to point. See Section 8.1.8 [`@cite`], page 88.)

7.2 Parts of a Cross Reference

A cross reference command requires only one argument, which is the name of the node to which it refers. But a cross reference command may contain up to four additional arguments. By using these arguments, you can provide a menu item name for Info, a descriptive phrase for the printed output, the name of a different Info file, and the name of a different printed manual.

Here is a simple cross reference example:

```
@xref{Node name}.
```

which produces

```
*Note Node name::.
```

and

See section *nnn* [Node name], page *ppp*.

Here, however, is an example of a full five-part cross reference:

```
@xref{Node name, Item name, Topic, info-file-name,
A Printed Manual}, for details.
```

which produces

```
*Note Item name: (info-file-name)Node name, for details.
```

and

See section Topic of *A Printed Manual*, for details.

The five arguments for a cross reference are:

1. The node name (required). This is the node to which the cross reference takes you. In a printed document, the location of the node provides the page reference (for references within the same document).
2. The item name for the Info reference, if it is to be different from the node name. It is usually omitted.
3. A topic description or section name. Often, this is the title of the section. This is used as the name of the reference in the printed manual. If omitted, the node name is used.
4. The name of the Info file in which the reference is located, if it is different from the current file.
5. The name of another printed manual.

Cross references with one, two, three, four, and five arguments are described separately following the description of `@xref`.

You can write cross reference commands within a paragraph, but note how Info and TEX format the output of each of the various commands: write `@xref` at the beginning of a sentence; write `@pxref` only within parentheses, and so on.

7.3 `@xref`

The `@xref` command generates a cross reference for the beginning of a sentence. The Info formatting commands convert it into an Info cross reference, which the Info '`f`' command can use to bring you directly to another node. The TEX typesetting commands convert it into a page reference, or a reference to another book or manual.

Most often, an Info cross reference looks like this:

`*Note` *node-name*`::.`

or like this

`*Note` *item-name*`:` *node-name*`.`

In TEX, a cross reference looks like this:

```
See section section [node-name], page page
```

or like this

```
See section section [topic], page page
```

The `@xref` command does not generate a period or comma to end the cross reference in either the Info file or the printed output. You must write that period or comma yourself; otherwise, Info will not recognize the end of the reference. (The `@pxref` command works differently. See Section 7.6 [@pxref], page 77.)

> *Please note:* A period or comma **must** follow the closing brace of an `@xref`. It is required to terminate the cross reference. This period or comma will appear in the output, both in the Info file and in the printed manual.

`@xref` must refer to an Info node by name. Use `@node` to define the node (see Section 6.2.1 [Writing a Node], page 59).

`@xref` is followed by several arguments inside braces, separated by commas. Whitespace before and after these commas is ignored.

A cross reference requires only the name of a node; but it may contain up to four additional arguments. Each of these variations produces a cross reference that looks somewhat different.

7.3.1 `@xref` with One Argument

The simplest form of `@xref` takes one argument, the name of another node in the same Info file.

For example,

```
@xref{Tropical Storms}.
```

produces

```
*Note Tropical Storms::.
```

and

> See section *nnn* [Tropical Storms], page *ppp*.

(Note that in the preceding example the closing brace is followed by a period.)

You can write a clause after the cross reference, like this:

```
@xref{Tropical Storms}, for more info.
```

which produces:

```
*Note Tropical Storms::, for more info.
```

> See section *nnn* [Tropical Storms], page *ppp*, for more info.

(Note that in the preceding example the closing brace is followed by a comma, and then by the clause.)

7.3.2 `@xref` with Two Arguments

With two arguments, the second one is used as the name of the Info cross reference, while the first argument is still the node that the cross reference points to:

The template is like this:

`@xref` *node-name*`,` *item-name*`.`

For example:

```
@xref{Electrical Effects, Lightning}.
```

which produces:

```
*Note Lightning: Electrical Effects.
```

and

> See section *nnn* [Electrical Effects], page *ppp*.

(Note that in the preceding example the closing brace is followed by a period; and that the node name is printed, not the item name.)

You can write a clause after the cross reference, like this:

```
@xref{Electrical Effects, Lightning}, for more info.
```

produces

```
*Note Lightning: Electrical Effects, for more info.
```

and

> See section *nnn* [Electrical Effects], page *ppp*, for more info.

(Note that in the preceding example the closing brace is followed by a comma, and then by the clause.)

7.3.3 `@xref` with Three Arguments

A third argument replaces the node name in the TEX output. The third argument should state the topic discussed by the section being referenced, or be the name of the section. Often, you will want to use initial upper case letters so it will be easier to read when the reference is printed. Use a third argument when the node name is unsuitable because of syntax or meaning.

Remember that a comma or period must follow the closing brace of an `@xref` command to terminate the cross reference. In the following examples, a clause follows a terminating comma.

The template is like this:

`@xref` *node-name*`,` *item-name*`,` *topic*`.`

For example,

```
@xref{Electrical Effects, Lightning, Thunder and Lightning}, for details.
```

which produces

```
*Note Lightning: Electrical Effects, for details.
```

and

> See section *nnn* [Thunder and Lightning], page *ppp*, for details.

If a third argument is given and the second one is empty, then the third argument serves both. (Note how two commas, side by side, mark the empty second argument.)

```
@xref{Electrical Effects, , Thunder and Lightning}, for details.
```

produces

```
*Note Thunder and Lightning: Electrical Effects, for details.
```

and

> See section *nnn* [Thunder and Lightning], page *ppp*, for details.

7.3.4 `@xref` with Four and Five Arguments

In a cross reference, a fourth argument specifies the name of another Info file, different from the file in which the reference appears, and a fifth argument specifies its title as a printed manual.

Remember that a comma or period must follow the closing brace of an `@xref` command to terminate the cross reference. In the following examples, a clause follows a terminating comma.

The template is:

`@xref{`*node-name*`,` *item-name*`,` *topic*`,` *info-file-name*`,` *printed-title*`}.`

For example,

```
@xref{Electrical Effects, Lightning, Thunder and Lightning,
weather, An Introduction to Meteorology}, for details.
```

which produces

```
*Note Lightning: (weather)Electrical Effects, for details.
```

The name of the the Info file is enclosed in parentheses and precedes the name of the node.

In a printed manual, the reference looks like this:

> See section Thunder and Lightning of *An Introduction to Meteorology*, for details.

The name of the printed manual is typeset in italics; and the reference lacks a page number since TEX cannot know to which page a refer refers when the reference is to another manual.

Often, you will leave out the second argument when you use the long version of `@xref`. In this case, the third argument, the topic description, will be used as the item name in Info.

The template looks like this:

`@xref{`*node-name*`, , `*topic*`, `*info-file-name*`, `*printed-title*`}, for details.`

which produces

`*Note` *topic*`: (`*info-file-name*`)`*node-name*`, for details.`

and

> See section *topic* of *printed-manual-title*, for details.

For example:

```
@xref{Electrical Effects, , Thunder and Lightning,
weather, An Introduction to Meteorology}, for details.
```

which produces

```
*Note Thunder and Lightning: (weather)Electrical Effects, for details.
```

and

> See section Thunder and Lightning of *An Introduction to Meteorology*, for details.

On rare occasions, you may want to refer to another Info file that is is within a single printed manual—when multiple Texinfo files are incorporated into the same TEX run but make separate Info files. In this case, you need to specify only the fourth argument, and not the fifth.

7.4 Naming a 'Top' Node

In a cross reference, you must always name a node. This means that in order to refer to a whole manual, you must identify the 'Top' node by writing it as the first argument to the `@xref` command. (This is different from the way you write a menu entry. See Section 6.5 [Referring to Other Info Files], page 64.) At the same time, to provide a meaningful section topic or title in the printed cross reference (instead of the word 'Top'), you must write an appropriate entry for the third argument to the `@xref` command.

Thus, to make a cross reference to *The GNU Make Manual*, write:

```
@xref{Top, , Overview, make, The GNU Make Manual}.
```

which produces

```
*Note Overview: (make)Top.
```

and

> See section Overview of *The GNU Make Manual*.

In this example, '`Top`' is the name of the node, and '`Overview`' is the name of the first section of the manual.

7.5 @ref

`@ref` is nearly the same as `@xref` except that it does not generate a 'See' in the printed output, just the reference itself. This makes it useful as the last part of a sentence.

For example:

```
For more information, see @ref{Orogenesis, ,
Mountaing Building}.
```

produces

```
For more information, see *Note Mountain
Building: Orogenesis.
```

and

> For more information, see section *nnn* [Mountain Building]. page *ppp*.

The `@ref` command sometimes leads writers to express themselves in a manner that is suitable for a printed manual but looks awkward in the Info format. Bear in mind that your audience will be using both the printed and the Info format.

For example,

```
Sea surges are described in @ref{Hurricanes}.
```

produces

> Sea surges are described in section *nnn* [Hurricanes].

in a printed document, but

```
Sea surges are described in *Note Hurricanes::.
```

in Info.

Caution: You *must* write a period or comma immediately after an `@ref` command with two or more arguments. Otherwise, Info will not find the end of the cross reference entry and attempts to follow the cross reference will fail. As a general rule, you should write a period or comma after every `@ref` command. This looks best in both the printed and the Info output.

7.6 `@pxref`

The parenthetical reference command, `@pxref`, is nearly the same as `@xref`, but you use it *only* inside parentheses and you do *not* type a comma or period after the command's closing brace. The command differs from `@xref` in two ways:

1. TEX typesets the reference for the printed manual with a lower case 'see' rather than an upper case 'See'.
2. The Info formatting commands automatically end the reference with a closing colon or period.

Because one type of formatting automatically inserts closing punctuation and the other does not, you should use `@pxref` *only* inside parentheses as part of another sentence. Also, you yourself should not insert punctuation after the reference, as you do with `@xref`.

`@pxref` is designed so that the output looks right and works right between parentheses both in printed output and in an Info file. In a printed manual, a closing comma or period should not follow a cross reference within parentheses; such punctuation is wrong. But in an Info file, suitable closing punctuation must follow the cross reference so Info can recognize its end. `@pxref` spares you the need to use complicated methods to put a terminator into one form of the output and not the other.

Don't try to use `@pxref` as a clause in a sentence. It will look bad in either the Info file, the printed output, or both. Use it only as a parenthetical reference.

With one argument, a parenthetical cross reference looks like this:

```
... large storms (@pxref{Hurricanes}) may cause flooding
...
```

which produces

```
... large storms (*Note Hurricanes::) may cause flooding ...
```

and

... large storms (see section *nnn* [Hurricanes], page *ppp*) may cause flooding ...

With two arguments, a parenthetical cross reference has this template:

... `(@pxref{`*node-name*`,` *item-name*`})` ...

which produces

... `(*Note` *item-name*`:` *node-name*`.)` ...

and

... (see section *nnn* [*node-name*], page *ppp*) ...

`@pxref` can be used with up to five arguments just like `@xref` (see Section 7.3 [`@xref`], page 69).

7.7 `@inforef`

`@inforef` is used for cross references to Info files for which there are no printed manuals. Even in a printed manual, `@inforef` generates a reference directing the user to look in an Info file.

The command takes either two or three arguments, in the following order:

1. The node name.
2. The item name (optional).
3. The Info file name.

Separate the arguments with commas, as with `@xref`. Also, you must terminate the reference with a comma or period after the '`}`', as you do with `@xref`.

The template is:

`@inforef{`*node-name*`,` *item-name*`,` *info-file-name*`},`

Thus,

```
@inforef{Expert, Advanced Info commands, info},
for more information.
```

produces

```
*Note Advanced Info commands: (info)Expert,
for more information.
```

and

> See Info file ‘info’, node ‘Expert’, for more information.

Similarly,

```
@inforef{Expert, , info}, for more information.
```

produces

```
*Note (info)Expert::, for more information.
```

and

> See Info file ‘info’, node ‘Expert’, for more information.

The converse of `@inforef` is `@cite`, which is used to refer to printed works for which no Info form exists. See Section 8.1.8 [@cite], page 88.

8 Marking Words and Phrases

In Texinfo, you can mark words and phrases in a variety of ways. These ways specify, for example, whether a word or phrase is a defining occurrence, a metasyntactic variable, or a symbol used in a program. Also, you can emphasize text.

8.1 Indicating Definitions, Commands, etc.

Texinfo has commands for indicating just what kind of object a piece of text refers to. Metasyntactic variables, for example, are marked by `@var` and code by `@code`. Texinfo uses this information to determine how to highlight the text. Since the pieces of text are labelled by commands that tell what kind of object they are, it is easy to change the way Texinfo formats such text. (Texinfo is an *intentional* formatting language rather than a *typesetting* formatting language.)

For example, code is usually illustrated in a typewriter font, but it would be easy to change the way Texinfo highlights code to use another font. This change would not effect how keystroke examples are highlighted. If straight typesetting commands were used in the body of the file and you wanted to make a change, you would have to check every single occurrence to make sure that you were changing code and not something else that should not be changed.

The highlighting commands can be used to generate useful information from the file, such as lists of functions or file names. It is possible, for example, to write a program in Emacs Lisp (or a keyboard macro) to insert an index entry after every paragraph that contains words or phrases marked by a specified command. You could do this to construct an index of functions if you had not already made the entries.

The commands serve a variety of purposes:

`@code{`*sample-code*`}`
: Indicate text that is a literal example of a piece of a program.

`@kbd{`*keyboard-characters*`}`
: Indicate keyboard input.

`@key{`*key-name*`}`
: Use for the conventional name for a key on a keyboard.

`@samp{`*text*`}`
: Indicate text that is a literal example of a sequence of characters.

`@var{`*metasyntactic-variable*`}`
: Indicate a metasyntactic variable.

`@file{`*file-name*`}`
: Indicate the name of a file.

`@dfn{`*term*`}`
: Use for the introductory or defining use of a term.

`@cite{`*reference*`}`
: Indicate the name of a book.

8.1.1 `@code{`*sample-code*`}`

Use the `@code` command to indicate text that is a piece of a program and which consists of entire syntactic tokens. Enclose the text in braces.

Thus, you should use `@code` for an expression in a program, for the name of a variable or function used in a program, or for a keyword. Also, you should use `@code` for the name of a program, such as `diff`, that is a name used in the machine. (You should write the name of a program in the ordinary text font if you regard it as a new English word, such as 'Emacs' or 'Bison'.)

Use `@code` for the `TEXINPUTS` environment variable and other such variables.

Do not use the `@code` command for a string of characters shorter than a syntactic token. In particular, you should not use the `@code` command when writing about the characters used in a token; do not, for example, use `@code` when you are explaining what letters or printable symbols can be used in the names of functions. (Use `@samp`.) Also, you should not use `@code` to mark text that is considered input to programs unless the input is written in a language that is like a programming language. For example, you should not use `@code` for the single character commands of GNU Emacs (use `@kbd` instead) although you may use `@code` for the names of the Emacs Lisp functions that the keyboard commands invoke.

Do use `@code` for command names in command languages that resemble programming languages, such as Texinfo or the shell. Note, however, that you should not use `@code` for options such as '`-c`' when such options stand alone. (Use `@samp`.) Also, an entire shell command often looks better if written using `@samp` rather than `@code`.

It is incorrect to alter the case of a word inside an `@code` command when it appears at the beginning of a sentence. Most computer languages are case sensitive. In C, for example, `Printf` is

a misspelling of `printf`. If you do not want to start a sentence with such a command written all in lower case, you should rearrange the sentence.

In the printed manual, `@code` causes TEX to typeset the argument in a typewriter face. In the Info file, it causes the Info formatting commands to use '...' quotation. For example:

```
Use @code{diff} to compare two files.
```

produces this in the printed manual:

Use `diff` to compare two files.

and this in Info file:

```
Use ‘diff’ to compare two files.
```

8.1.2 `@kbd{`*keyboard-characters*`}`

Use the `@kbd` command for characters of input to be typed by users. For example, to refer to the characters `M-a`, write

```
@kbd{M-a}
```

and to refer to the characters `M-x shell`, write

```
@kbd{M-x shell}
```

The `@kbd` command has the same effect as `@code` in Info, but may produce a different font in a printed manual.

You can embed another `@`-command inside the braces of an `@kbd` command. Here, for example, is the way to describe a command that would be described more verbosely as "press an '`r`' and then press the `RET` key":

```
@kbd{r @key{RET}}
```

This produces: `r RET`

You also use the `@kbd` command if you are spelling out the letters you type; for example:

```
To give the @code{logout} command,
type the characters @kbd{l o g o u t @key{RET}}.
```

This produces

> To give the `logout` command, type the characters `l o g o u t RET`.

(Also, this example shows that you can add spaces for clarity. If you really want to mention a space character as one of the characters of input, write `@key{SPC}` for it.)

8.1.3 `@key{`*key-name*`}`

Use the `@key` command for the conventional name for a key on a keyboard, as in

```
@key{RET}
```

You can use the `@key` command within the argument of an `@kbd` command when the sequence of characters to be typed includes one or more keys that are described by name.

For example, to produce `C-x ESC` you would type:

```
@kbd{C-x @key{ESC}}
```

The recommended names to use for keys are in upper case and are

`SPC`	Space
`RET`	Return
`LFD`	Linefeed
`TAB`	Tab
`BS`	Backspace
`ESC`	Escape
`DEL`	Delete
`SFT`	Shift
`CTL`	Control

`META` Meta

There are subtleties to handling words like ‘meta’ or ‘ctl’ that are names of shift keys. When mentioning a character in which the shift key is used, such as `Meta-a`, use the `@kbd` command alone without the `@key` command, but when you are referring to the shift key in isolation, use the `@key` command. For example, write ‘`@kbd{Meta-a}`’ to produce `Meta-a` and ‘`@key{META}`’ to produce `META`. This is because `Meta-a` refers to keys that you press on a keyboard, but `META` refers to a key without implying that you press it.

8.1.4 `@samp{`*text*`}`

Use the `@samp` command to indicate text that is a literal example of a sequence of characters in a file, string, pattern, etc. Enclose the text in braces. The argument appears within ‘...’ quotation in both the Info file and the printed manual; in addition, it is printed in a fixed-width font.

```
To match @samp{foo} at the end of the line,
use the regexp @samp{foo$}.
```

produces

> To match ‘`foo`’ at the end of the line, use the regexp ‘`foo$`’.

Any time you are referring to single characters, you should use `@samp` unless `@kbd` is more appropriate. `@samp` is used for entire statements in C, for entire shell commands, and for names of command-line options. Use it for buffer names in Emacs and for node names in Info or Texinfo. Basically, `@samp` is a catchall for whatever is not covered by `@code`, `@kbd`, or `@key`.

Only include punctuation marks within braces if they are part of the string you are specifying. Write punctuation marks outside the braces if those punctuation marks are part of the English text that surrounds the string. In the following sentence, for example, the commas and period are outside of the braces:

```
In English, the vowels are @samp{a}, @samp{e},
@samp{i}, @samp{o}, @samp{u}, and sometimes
@samp{y}.
```

This produces:

> In English, the vowels are ‘`a`’, ‘`e`’, ‘`i`’, ‘`o`’, ‘`u`’, and sometimes ‘`y`’.

8.1.5 `@var{`*metasyntactic-variable*`}`

Use the `@var` command to indicate metasyntactic variables. A metasyntactic variable is something that stands for another piece of text. For example, you should use a metasyntactic variable in the documentation of a function to describe the arguments that are passed to that function.

Do not use `@var` for the names of particular variables in programming languages. These are specific names from a program, so `@code` is correct for them. For example, the Lisp variable `texinfo-tex-command` is not a metasyntactic variable; it is properly formatted using `@code`.

The effect of `@var` in the Info file is to upcase the argument; in the printed manual, to italicize it. For example:

```
To delete file @var{filename},
type @code{rm @var{filename}}.
```

produces

> To delete file *filename*, type `rm` *filename*.

(Note that `@var` may appear inside of `@code`, `@samp`, `@file`, etc.)

Write a metasyntactic variable all in lower case without spaces, and use hyphens to make it more readable. Thus, the illustration of how to begin a Texinfo manual looks like this:

```
\input texinfo
@setfilename info-file-name
@settitle name-of-manual
```

In some documentation styles, metasyntactic variables are shown with angle brackets, for example:

```
..., type rm <filename>
```

However, that is not the style we use in Texinfo. (You can, of course, modify the sources to TEX and the Info formatting commands to output the `<...>` format if you wish.)

8.1.6 `@file{`*file-name*`}`

Use the `@file` command to indicate text that is the name of a file, buffer, or directory, or is the name of a node in Info. You can also use the command for filename suffixes. Don't use `@file` for symbols in a programming language; thus, a node name is a name in an Info file but not an identifier in a programming language.

Currently, `@file` is equivalent to `@samp` in its effects on the output. For example,

```
The @file{.el} files are in
the @file{/usr/local/emacs/lisp} directory.
```

produces

> The '`.el`' files are in the '`/usr/local/emacs/lisp`' directory.

8.1.7 `@dfn{`*term*`}`

Use the `@dfn` command to identify the introductory or defining use of a technical term. Use the command only in passages whose purpose is to introduce a term which will be used again or which the reader ought to know. Mere passing mention of a term for the first time doesn't deserve `@dfn`. The command generates italics in the printed manual, and double quotation marks in the Info file. For example:

```
Getting rid of a file is called @dfn{deleting} it.
```

produces

> Getting rid of a file is called *deleting* it.

As a general rule, a sentence containing the defining occurrence of a term should be a definition of the term. The sentence does not have to say explicitly that it is a definition, but it should contain the information of a definition—it should make the meaning clear.

8.1.8 `@cite{`*reference*`}`

Use the `@cite` command for the name of a book that lacks a companion Info file. The command produces italics in the printed manual, and quotation marks in the Info file.

(If a book is written in Texinfo, it is better to use a cross reference command since you can easily follow such a reference in Info. See Section 7.3 [`@xref`], page 69.)

8.2 Emphasizing Text

Usually, Texinfo changes the font to mark words in the text according to what category the words belong to. The `@code` command, for example, does this. Most often, this is the best way to mark words. However, sometimes you will want to emphasize text without indicating a category. Texinfo has two ways to do this: commands that tell Texinfo to emphasize the text but leave the method to the program, and commands that specify the method to use. The first method is generally the best because it makes it possible to change the style of a document without needing to re-edit it line by line.

8.2.1 `@emph{`*text*`}` and `@strong{`*text*`}`

The `@emph` and `@strong` commands are for emphasis; `@strong` is stronger. In printed output, `@emph` produces *italics* and `@strong` produces **bold**.

For example,

```
@quotation
@strong{Caution:} @code{rm * .[^.]*} removes @emph{all}
files in the directory.
@end quotation
```

produces the following in printed output:

> **Caution**: `rm * .[^.]*` removes *all* files in the directory.

and the following in Info:

```
*Caution*: 'rm * .[^.]*' removes *all*
files in the directory.
```

The `@strong` command is seldom used except to mark what is, in effect, a typographical element, such as the word 'Caution' in the preceding example.[1]

In the Info file, both `@emph` and `@strong` put asterisks around the text.

8.2.2 `@sc{`*text*`}`: The Small Caps Font

Use the '`@sc`' command to set text in the printed output in A SMALL CAPS FONT and set text in the Info file in upper case letters.

Write the text between braces in lower case, like this:

```
The @sc{acm} and @sc{ieee} are technical societies.
```

This produces:

The ACM and IEEE are technical societies.

TEX typesets the small caps font in a manner that prevents the letters from 'jumping out at you on the page'. This makes small caps text easier to read than text in all upper case.

TEX typesets any upper case letters in the small caps fonts in FULL-SIZE CAPITALS. Use them sparingly.

The Info formatting commands set all small caps text in upper case.

You may also use the small caps font for a jargon word such as ATO (a NASA word meaning 'abort to orbit').

---

[1] Don't try to use `@emph` or `@strong` with the word '`Note`'; Info will mistake the combination for a cross reference. Use a phrase such as *Please note* or *Caution* instead.

There are subtleties to using the small caps font with a jargon word such as CDR, a word used in Lisp programming. In this case, you should use the small caps font when the word refers to the second and subsequent elements of a list (the CDR of the list), but you should use '`@code`' when the word refers to the Lisp function of the same spelling.

8.2.3 Fonts for Printing, Not Info

Texinfo provides four font commands that specify font changes in the printed manual but have no effect in the Info file. `@i` requests *italic* font (in some versions of TEX, a slanted font is used), `@b` requests **bold** face, `@t` requests the `fixed-width` font used by `@code`, and `@r` requests a roman font, which is the usual font in which text is printed. All four commands apply to an argument that follows, surrounded by braces.

Only the `@r` command has much use: in example programs, you can use the `@r` command to convert code comments from the fixed-width font to a roman font. This looks better in printed output.

For example,

```
@lisp
(+ 2 2)    ; @r{Add two plus two.}
@end lisp
```

produces

```
(+ 2 2)    ; Add two plus two.
```

If possible, you should avoid using the other three font commands. If you need to use one, it probably indicates a gap in the Texinfo language.

9 Special Insertions

Texinfo provides several commands for for inserting single characters that have special meaning in Texinfo, such as braces, and for inserting special graphic symbols that do not correspond to characters, such as dots and bullets.

These are:

- Braces, '@' and periods.
- Dots and bullets.
- The TEX logo and the copyright symbol.
- A minus sign.

9.1 Inserting '@', Braces, and Periods

'@' and curly braces are special characters in Texinfo. To insert these characters into text, you must put an '@' in front of these characters to prevent Texinfo from misinterpreting them.

Periods are also special. Depending on whether the period is inside of or at the end of a sentence, less or more space is inserted after a period in a typeset manual. Since it is not always possible for Texinfo to determine when a period ends a sentence and when it is used in an abbreviation, special commands are needed in some circumstances. (Usually, Texinfo can guess how to handle periods, so you don't have to use the special commands; you just enter a period as you would if you were using a typewriter, which means you put two spaces after the period, question mark, or exclamation mark that ends a sentence.)

Do not put braces after any of these commands; they are not necessary.

9.1.1 Inserting '@'—@@

@@ stands for a single '@' in either printed or Info output.

Do not put braces after an @@ command.

9.1.2 Inserting '{' and '}'—@{ and @}

`@{` stands for a single '{' in either printed or Info output.

`@}` stands for a single '}' in either printed or Info output.

Do not put braces after either an `@{` or an `@}` command.

9.1.3 Spacing After Colons and Periods

Use the `@:` command after a period, question mark, exclamation mark, or colon that should not be followed by extra space. For example, use `@:` after periods that end abbreviations which are not at the ends of sentences. `@:` has no effect on the Info file output.

For example:

```
The U.S.A.@: is a continental nation.
```

produces

The U.S.A. is a continental nation.

Use `@.` instead of a period at the end of a sentence that ends with a single capital letter. Otherwise, TEX will think the letter is an abbreviation and will not insert the correct end-of-sentence spacing. Here is an example:

```
Give it to X. and to Y@.  Give it to Z@.
Give it to X. and to Y.  Give it to Z.
```

If you look carefully at this printed output, you will see a little more whitespace after the Y in the first line than the Y in the second line.

Give it to X. and to Y. Give it to Z.
Give it to X. and to Y. Give it to Z.

In the Info file output, `@.` is equivalent to a simple '.'.

The meanings of `@:` and `@.` in Texinfo are designed to work well with the Emacs sentence motion commands. This made it necessary for them to be incompatible with some other formatting systems that use `@`-commands.

Do not put braces after either an `@:` or an `@.` command.

9.2 Inserting Ellipsis, Dots, and Bullets

An *ellipsis* (a line of dots) is typeset unlike a string of periods, so a special command is used for ellipsis in Texinfo. The `@bullet` command is special, too. Each of these commands is followed by a pair of braces, '`{}`', without any whitespace between the name of the command and the braces. (For an explanation of why the braces are needed, see Appendix F [`@`-Command Syntax], page 201).

9.2.1 `@dots{}`

Use the `@dots{}` command to generate an ellipsis, which is three dots in a row, appropriately spaced, like this: '...'. Do not simply write three periods in the input file; that would work for the Info file output, but would produce the wrong amount of space between the periods in the printed manual.

Here is an ellipsis: ...

Here are three periods in a row: ...

In printed output, the three periods in a row are closer together than the dots in the ellipsis.

9.2.2 `@bullet{}`

Use the `@bullet{}` command to generate a large round dot, or the closest possible thing to one. In Info, an asterisk is used.

Here is a bullet: •

When you use `@bullet` in `@itemize`, you do not need to type the braces, because `@itemize` supplies them.

9.3 Inserting TEX and the Copyright Symbol

The logo TEX is typeset in a special fashion and it needs an `@`-command, as does the command for inserting the copyright symbol. Each of these commands is followed by a pair of braces, '`{}`', without any whitespace between the name of the command and the braces.

9.3.1 `@TeX{}`

Use the `@TeX{}` command to generate 'TEX'. In a printed manual, this is a special logo that is different from three ordinary letters. In Info, it just looks like '`TeX`'. The `@TeX{}` command is unique among Texinfo commands in that the `T` and the `X` are in upper case.

9.3.2 `@copyright{}`

Use the `@copyright{}` command to generate '©'. In a printed manual, this is a '`c`' inside a circle, and in Info, this is '`(C)`'.

9.4 `@minus{}`: Inserting a Minus Sign

Use the `@minus{}` command to generate a minus sign. In a fixed-width font, this is a single hyphen, but in a proportional font, the symbol is the customary length for a minus sign—a little longer than a hyphen.

You can compare the two forms:

'−' is a minus sign generated with '`@minus{}`',

'`-`' is a hyphen generated with the character '`-`'.

In the fixed-width font used by Info, `@minus{}` is the same as a hyphen.

You should not use `@minus{}` inside of `@code` or `@example` because the width distinction is not made in the fixed-width font they use.

When you use `@minus` to specify the mark beginning each entry in an itemized list, you do not need to type the braces.

10 Quotations and Examples

Quotations and examples are blocks of text consisting of one or more whole paragraphs that are set off from the bulk of the text and treated differently. They are usually indented.

In Texinfo, you always begin a quotation or example by writing an `@`-command at the beginning of a line by itself, and end it by writing an `@end` command that is also at the beginning of a line by itself. For instance, you begin an example by writing `@example` by itself at the beginning of a line and end the example by writing `@end example` on a line by at itself, at the beginning of that line.

10.1 The Various Block Enclosing Commands

There are a variety of commands for quotations and examples:

`@quotation`
: Used to indicate text that is quoted. The text is filled, indented, and printed in a roman font by default.

`@example`
: Used to illustrate code, commands, and the like. The text is printed in a fixed-width font without filling.

`@lisp`
: Used to illustrate Lisp code. The text is printed in a fixed-width font without filling.

`@smallexample`
: Used to illustrate code, commands, and the like. The text is printed in a fixed-width font without filling; for use with the `@smallbook` command.

`@smalllisp`
: Used to illustrate Lisp code. The text is printed in a fixed-width font without filling; for use with the `@smallbook` command.

`@display`
: Used for illustrative text. The text is indented but not filled, and no font is specified (so, by default, the font is roman).

`@format`
: Used for illustrative text. The text is not indented and not filled and no font is specified (so, by default, the font is roman).

The `@exdent` command is used within the above constructs to undo the indentation of a line. The `@flushleft` and `@flushright` commands are used to line up the left or right margins of unfilled text.

The `@noindent` command may be used after one of the above constructs to prevent the following text from being indented as a new paragraph.

10.2 @quotation

The text of a quotation is processed normally except that

- The margins are narrower, so the whole of the quotation is indented.
- The first lines of paragraphs are indented no more than the other lines.
- In the printed output, interline spacing and interparagraph spacing are reduced.

> This is an example of text written between an `@quotation` command and an `@end quotation` command. A `@quotation` command is most often used to indicate text that is excerpted from another (real or hypothetical) printed work.

Write an `@quotation` command as text of a line by itself. This line will disappear from the output. Mark the end of the quotation with a line beginning with and containing only `@end quotation`. The `@end quotation` line will likewise disappear from the output. Thus, the input

```
@quotation
This is
a foo.
@end quotation
```

produces

> This is a foo.

10.3 @example

The `@example` command is used to indicate an example that is not part of the running text, such as computer input or output.

```
This is an example of text written between an
@example command and an @end example
command.  The text is indented but not filled.

In the printed manual, the text is typeset in a
fixed-width font, and extra spaces and blank lines are
significant.  In the Info file, an analogous result is
obtained by indenting each line with five extra spaces.
```

Write an `@example` command at the beginning of and as the only text on a line by itself. This line will disappear from the output. Mark the end of the example with a line beginning with and containing only `@end example`. The `@end example` will likewise disappear from the output. For example:

```
@example
mv foo bar
@end example
```

produces

```
mv foo bar
```

Since the lines containing `@example` and `@end example` will disappear, you should put a blank line before the `@example` and another blank line after the `@end example`. (Remember that blank lines between the beginning `@example` and the ending `@end example` will appear in the output.)

> **Caution:** Do not use tabs in lines of an example (or anywhere else in Texinfo, for that matter)! TEX treats tabs like single spaces, and that is not what they look like. This is a problem with TEX. (If necessary, in Emacs, you can use `M-x untabify` to convert tabs in a region to multiple spaces.)

When you use `@example` to describe a C function's calling conventions, use the ANSI C syntax, like this:

```
void dld_init (char *@var{path});
```

And in the subsequent discussion, refer to the argument values by writing the same argument names, again highlighted with `@var`.

Avoid the obsolete style that looks like this:

```
#include <dld.h>

dld_init (path)
char *path;
```

Also, it is best to avoid writing `#include` above the declaration just to indicate that the function is declared in a header file. The practice may give the misimpression that the `#include` belongs near the declaration of the function. Either state explicitly which header file holds the declaration

or, better yet, name the header file used for a group of functions at the beginning of the section that describes the functions.

Examples are often, logically speaking, "in the middle" of a paragraph, and the text continues after an example should not be indented. The `@noindent` command prevents a piece of text from being indented as if it were a new paragraph.

(The `@code` command is used for examples of code that is embedded within sentences, not set off from preceding and following text. See Section 8.1.1 [@code], page 82.)

10.4 @noindent

If you have text following an `@example` or other similar inclusion that reads as a continuation of the text before the `@example`, it is good to prevent this text from being indented as a new paragraph. To accomplish this, write `@noindent` at the beginning of a line by itself preceding the continuation text. For example,

```
@example
This is an example
@end example

@noindent
This line will not be indented.  As you can see, the
beginning of the line is fully flush left with the line
that follows after it.  (This whole example is between
@display and @end display.)
```

produces

```
This is an example
```

This line will not be indented. As you can see, the
beginning of the line is fully flush left with the line
that follows after it. (This whole example is between
`@display` and `@end display`.)

To adjust the number of blank lines properly in the Info file output, remember that the line containing `@noindent` does not generate a blank line, and neither does the `@end example` line.

In the Texinfo source file for this documentation, each of the lines that says 'produces' is preceded by a line containing `@noindent`.

Do not put braces after an `@noindent` command; they are not necessary, since `@noindent` is a command used outside of paragraphs (see Appendix F [Command Syntax], page 201).

10.5 `@lisp`

The `@lisp` command is used for Lisp code. It is synonymous with the `@example` command.

```
This is an example of text written between an
@lisp command and an @end lisp command.
```

Use `@lisp` instead of `@example` so as to preserve information regarding the nature of the example. This is useful, for example, if you write a function that evaluates only and all the Lisp code in a Texinfo file. Then you can use the Texinfo file as a Lisp library.[1]

Mark the end of `@lisp` with `@end lisp` on a line by itself.

10.6 `@smallexample` and `@smalllisp`

In addition to the regular `@example` and `@lisp` commands, Texinfo has two other "example-style" commands. These are the `@smallexample` and `@smalllisp` commands. Both these commands are designed for use with the `@smallbook` command that causes TEX to produce a printed manual in a 7 by 9.25 inch format rather than the regular 8.5 by 11 inch format.

In TEX, the `@smallexample` and `@smalllisp` commands are defined the same as the `@example` and `@lisp` commands for 8.5 by 11 inch output and produce similar looking output. But when TEX is preparing a manual in a 7 by 9.25 inch format, the text bounded by `@smallexample` or `@smalllisp` commands is typeset in a smaller font than that used for the ordinary examples.

---

[1] It would be straightforward to extend Texinfo to work in a similar fashion for C, Fortran, or other languages.

In Info, the `@smallexample` and `@smalllisp` commands are equivalent to the `@example` and `@lisp` commands.

Mark the end of `@smallexample` or `@smalllisp` with `@end smallexample` or `@end smalllisp`, respectively.

```
This is an example of text written between @smallexample
and  @end smallexample.
```

The `@smallexample` and `@smalllisp` commands make it easier to prepare smaller format manuals without forcing you to edit examples by hand to fit them onto narrower pages.

See Section 18.8 [Printing "Small" Books], page 150, for more information about the `@smallbook` command.

10.7 `@display`

The `@display` command begins a kine of example. It is like the `@example` command except that, in a printed manual, `@display` does not select the fixed-width font. In fact, it does not specify the font at all, so that the text appears in the same font it would have appeared in without the `@display` command.

This is an example of text written between an `@display` command
and an `@end display` command. The `@display` command
indents the text, but does not fill it.

10.8 `@format`

The `@format` command is similar to `@example` except that, in the printed manual, `@format` does not select the fixed-width font and does not narrow the margins.

This is an example of text written between an `@format` command
and an `@end format` command.
The `@format` command does not fill the text.

10.9 `@exdent`: Undoing a Line's Indentation

The `@exdent` command removes any indentation a line might have. The command is written at the beginning of a line and applies only to the text that follows the command that is on the same line. Don't use braces around the text. In the printed manual, the text on the `@exdent` line is printed in the roman font.

`@exdent` is usually used within examples. Thus,

```
@example
This line follows an @example command.
@exdent  This line is exdented.
This line follows the exdented line.
The @end example comes on the next line.
@end example
```

produces

```
     This line  follows an @example command.
This line is exdented.
     This line follows the exdented line.
     The @end example comes on the next line.
```

In practice, the `@exdent` command is rarely used. Usually, you un-indent text by ending the example and returning the page to its normal width.

10.10 `@flushleft` and `@flushright`

The `@flushleft` and `@flushright` commands line up the left or right ends of lines on the left and right margins of a page, but do not fill the text. The commands are written on lines of their own, without braces. The `@flushleft` and `@flushright` commands are ended by `@end flushleft` and `@end flushright` commands on lines of their own.

For example,

```
@flushleft
This text is
written flushleft.
@end flushleft
```

produces

This text is
written flushleft.

Flushright produces the type of indentation often used in the return address of letters.

```
@flushright
Here is an example of text written
flushright.  The @code{@flushright} command
right justifies every line but leaves the
left end ragged.
@end flushright
```

produces

Here is an example of text written
flushright. The `@flushright` command
right justifies every line but leaves the
left end ragged.

11 Special Glyphs for Examples

In Texinfo, code is often illustrated in examples that are delimited by `@example` and `@end example`, or by `@lisp` and `@end lisp`. In such examples, you can indicate the results of evaluation or an expansion using '⇒' or '↦'. Likewise, there are special symbols to indicate printed output, an error message, equivalence of expressions, and the location of point.

The special glyph commands do not have to be used within an example, but most often they are. Every special glyph command is followed by a pair of left- and right-hand braces.

⇒	`@result{}` points to the result of an expression.
↦	`@expansion{}` shows the results of a macro expansion.
⊣	`@print{}` indicates printed output.
error	`@error{}` indicates that the following text is an error message.
≡	`@equiv{}` indicates the exact equivalence of two forms.
⋆	`@point{}` shows the location of point.

11.1 ⇒: Indicating Evaluation

Use the `@result{}` command to indicate the result of evaluating an expression.

The `@result{}` command is displayed as '`=>`' in Info and as '⇒' in the printed output.

Thus, the following,

```
(cdr '(1 2 3))
     ⇒ (2 3)
```

may be read as "`(cdr '(1 2 3))` evaluates to (2 3)".

11.2 ↦: Indicating an Expansion

When an expression is a macro call, it expands into a new expression. You can indicate the result of the expansion with the `@expansion{}` command.

The `@expansion{}` command is displayed as '`==>`' in Info and as '↦' in the printed output.

For example, the following

```
@lisp
(third '(a b c))
     @expansion{} (car (cdr (cdr '(a b c))))
     @result{} c
@end lisp
```

produces

```
(third '(a b c))
     ↦ (car (cdr (cdr '(a b c))))
     ⇒ c
```

which may be read as:

> `(third '(a b c))` expands to `(car (cdr (cdr '(a b c))))`; the result of evaluating the expression is `c`.

(Often, as in this case, an example looks better if the `@expansion{}` and `@result{}` commands are indented five spaces.)

11.3 ⊣: Indicating Printed Output

Sometimes an expression will print output during its execution. You can indicate the printed output with the `@print{}` command.

The `@print{}` command is displayed as '`-|`' in Info and as '⊣' in the printed output.

In the following example, the printed text is indicated with '⊣', and the the value of the expression follows on the last line.

```
(progn (print 'foo) (print 'bar))
     ⊣ foo
     ⊣ bar
     ⇒ bar
```

In a Texinfo source file, this example is written as follows:

```
@lisp
(progn (print 'foo) (print 'bar))
     @print{} foo
     @print{} bar
     @result{} bar
@end lisp
```

11.4 error : Indicating an Error Message

A piece of code may cause an error when you evaluate it. You can designate the error message with the `@error{}` command.

The `@error{}` command is displayed as '`error-->`' in Info and as ' error ' in the printed output.

Thus,

```
@lisp
(+ 23 'x)
@error{} Wrong type argument: integer-or-marker-p, x
@end lisp
```

produces

```
(+ 23 'x)
error  Wrong type argument: integer-or-marker-p, x
```

This indicates that the following error message is printed when you evaluate the expression:

```
Wrong type argument: integer-or-marker-p, x
```

Note that ' error ' itself is not part of the error message.

11.5 ≡: Indicating Equivalence

Sometimes two expressions produce identical results. You can indicate the exact equivalence of

two forms with the `@equiv{}` command.

The `@equiv{}` command is displayed as '`==`' in Info and as '≡' in the printed output.

Thus,

```
@lisp
(make-sparse-keymap) @equiv{} (list 'keymap)
@end lisp
```

produces

```
(make-sparse-keymap) ≡ (list 'keymap)
```

This indicates that evaluating `(make-sparse-keymap)` produces identical results to evaluating `(list 'keymap)`.

11.6 Indicating Point in a Buffer

Sometimes you need to show an example of text in an Emacs buffer. In such examples, the convention is to include the entire contents of the buffer in question between two lines of dashes containing the buffer name.

You can use the '`@point{}`' command to show the location of point in the text in the buffer. (The symbol for point, of course, is not part of the text in the buffer; it indicates the place *between* two characters where point is located.)

The `@point{}` command is displayed as '`-!-`' in Info and as '⋆' in the printed output.

The following example shows the contents of buffer '`foo`' before and after evaluating a Lisp command to insert the word `changed`.

```
---------- Buffer: foo ----------
This is the ⋆contents of foo.
---------- Buffer: foo ----------
```

```
(insert "changed ")
     ⇒ nil
---------- Buffer: foo ----------
This is the changed ⋆contents of foo.
---------- Buffer: foo ----------
```

In a Texinfo source file, the example is written like this:

```
@example
---------- Buffer: foo ----------
This is the @point{}contents of foo.
---------- Buffer: foo ----------

(insert "changed ")
     @result{} nil
---------- Buffer: foo ----------
This is the changed @point{}contents of foo.
---------- Buffer: foo ----------
@end example
```

12 Making Lists and Tables

Texinfo has several ways of making lists and two-column tables. Lists can be bulleted or numbered, while two-column tables can highlight the items in the first column.

Texinfo automatically indents the text in lists or tables and numbers an enumerated list. This last feature is useful if you modify the list, since you do not have to renumber it yourself.

Numbered lists and tables begin with the appropriate `@`-command at the beginning of a line, and end with the corresponding `@end` command on a line by itself. The table and itemized-list commands also require that you write formatting information on the same line as the beginning `@`-command.

Begin an enumerated list, for example, with an `@enumerate` command and end the list with an `@end enumerate` command. Begin an itemized list with an `@itemize` command, followed on the same line by a formatting command such as `@bullet`, and end the list with an `@end itemize` command.

Precede each element of a list with an `@item` command.

Here is an itemized list of the different kinds of table and lists:

- Itemized lists with and without bullets.
- Numbered lists.
- Two-column tables with highlighting.

Here is an enumerated list with the same items:

1. Itemized lists with and without bullets.
2. Numbered lists.
3. Two-column tables with highlighting.

And here is a two-column table with the same items and their `@`-commands:

`@itemize` Itemized lists with and without bullets.

`@enumerate`
: Numbered lists.

`@table`
`@ftable` Two-column tables with highlighting.

12.1 Making an Itemized List

The `@itemize` command is used to produce sequences of indented paragraphs, with marks inside the left margin at the beginning of paragraphs for which a mark is desired.

Begin an itemized list by writing `@itemize` at the beginning of a line. Follow the command, on the same line, with a character or a Texinfo command that generates a mark. Usually, you will write `@bullet` after `@itemize`, but you can use `@minus`, or any character or any special symbol that results in a single character in the Info file. (When you write `@bullet` or `@minus` after an `@itemize` command, you may omit the '`{}`'.)

Write the text of the indented paragraphs themselves after the `@itemize`, up to another line that says `@end itemize`.

Before each paragraph for which a mark in the margin is desired, place a line that says just `@item`. Don't put any other text on this line.

Usually, you should put a blank line before an `@item`. This puts a blank line in the Info file. (TEX inserts the proper interline whitespace in either case.) Except when the entries are very brief, these blank lines make the list look better.

Here is an example of the use of `@itemize`, followed by the output it produces. Note that `@bullet` produces a '`*`' in Info and a round dot in TEX.

```
@itemize @bullet
@item
Some text for foo.

@item
Some text
for bar.
@end itemize
```

produces

- Some text for foo.
- Some text for bar.

Itemized lists may be embedded within other itemized lists. Here is a list marked with dashes embedded in a list marked with bullets:

```
@itemize @bullet
@item
First item.

@itemize @minus
@item
Inner item.

@item
Second inner item.
@end itemize

@item
Second outer item.
@end itemize
```

produces

- First item.
  - Inner item.
  - Second inner item.
- Second outer item.

12.2 Making a Numbered List

`@enumerate` is like `@itemize` except that the marks in the left margin contain successive integers starting with 1. (See Section 12.1 [`@itemize`], page 110.) Do not put any argument on the same line as `@enumerate`.

Normally, you should put a blank line between the entries in the list. This generally makes it easier to read the Info file.

```
@enumerate
@item
Some text for foo.

@item
Some text
for bar.
@end enumerate
```

produces

1. Some text for foo.
2. Some text for bar.

12.3 Making a Two-column Table

`@table` is similar to `@itemize`, but the command allows you to specify a name or heading line for each item. (See Section 12.1 [`@itemize`], page 110.) The `@table` command is used to produce two-column tables, and is especially useful for glossaries and explanatory exhibits.

Write the `@table` command at the beginning of a line and follow it on the same line by an argument that is a Texinfo command such as `@code`, `@samp`, `@var`, or `@kbd`. Also, you may use the `@asis` command. Although these commands are usually followed by arguments in braces, in this case you use the command name without an argument because `@item` will supply the argument. This command will be applied to the text that goes into the first column of each item and determines how it will be highlighted. For example, `@samp` will cause the text in the first column to be highlighted with an `@samp` command.

`@asis` is a command that does nothing; if you write this command after `@table`, TEX and the Info formatting commands output the first column entries without added highlighting, ('as is').

(The `@table` command may work with other commands besides those listed here. You can experiment. However, you can only use commands that normally take arguments in braces.)

Begin each table entry with an `@item` command at the beginning of a line. Write the first column text on the same line as the `@item` command. Write the second column text on the line following the `@item` line and on subsequent lines. (You don't have to type anything for an empty second column entry.) You may write as many lines of supporting text as you wish, even several paragraphs. But only text on the same line as the `@item` will be placed in the first column.

The following table, for example, highlights the text in the first column with an `@samp` command:

```
@table @samp
@item foo
This is the text for
@samp{foo}.

@item bar
Text for @samp{bar}.
@end table
```

produces

'foo' This is the text for 'foo'.
'bar' Text for 'bar'.

If you want to list two or more named items with a single block of text, use the `@itemx` command. (See Section 12.3.2 [@itemx], page 113.)

12.3.1 @ftable

The `@ftable` command is the same as the `@table` command except that it automatically enters each of the items in the first column of the table into the index of functions, which makes it easier to create indices. Only the items on the same line as the `@item` commands are indexed, and they are indexed in exactly the form that they appear on that line. See Chapter 15 [Creating Indices], page 131, for more information about indices.

Begin a two columns table using `@ftable` by writing `@ftable` at the beginning of a line, followed on the same line by an argument that is a Texinfo command such as `@code`, exactly as you would for an `@table` command; and end the table with an `@end ftable` command on a line by itself.

12.3.2 @itemx

Use the `@itemx` command inside a table when you have two or more first column entries for the same item, each of which should appear on a line of its own. Use `@itemx` for all but the first entry. The `@itemx` command works exactly like `@item` except that it does not generate extra vertical space above the first column text. For example,

```
@table @code
@item upcase
```

```
@itemx downcase
These two functions accept a character or a string as
argument, and return the corresponding upper case (lower
case) character or string.@refill
@end table
```

produces

upcase
downcase These two functions accept a character or a string as argument, and return the corresponding upper case (lower case) character or string.

(Note also that this example illustrates multi-line supporting text in a two-column table.)

13 Definition Commands: @deffn, etc.

The @deffn command and the other *definition commands* enable you to describe functions, variables, macros, commands, user options, special forms and other such artifacts in a uniform format.

In the Info file, a definition causes the category entity—'Function', 'Variable', or whatever—to appear at the beginning of the first line of the definition, followed by the entity's name and arguments. In the printed manual, the command causes TEX to print the entity's name and its arguments on the left margin and print the category next to the right margin. In both output formats, the body of the definition is indented. Also, the name of the entity is entered into the appropriate index: @deffn enters the name into the index of functions, @defvr enters it into the index of variables, and so on.

13.1 The Template for a Definition

The @deffn command is used for definitions of entities that resemble functions. To write a definition using the @deffn command, write the @deffn command at the beginning of a line and follow it on the same line by the category of the entity, the name of the entity itself, and its arguments (if any). Then write the body of the definition on succeeding lines. (You may embed examples in the body.) Finally, end the definition with an @end deffn command written on a line of its own. (The other definition commands follow the same format.)

The template for a definition looks like this:

```
@deffn category name arguments...
body-of-definition
@end deffn
```

For example,

```
@deffn Command forward-word count
This command moves point forward @var{count} words
(or backward if @var{count} is negative). ...
@end deffn
```

produces

forward-word *count* Command
This function moves point forward *count* words (or backward if *count* is negative). ...

Capitalize the category name like a title. If the name of the category contains spaces, as in the phrase 'Interactive Command', write braces around it. For example,

```
@deffn {Interactive Command} isearch-forward
...
@end deffn
```

Otherwise, the second word will be mistaken for the name of the entity.

Some of the definition commands are more general than others. The `@deffn` command, for example, is the general definition command for functions and the like—for entities that may take arguments. When you use this command, you specify the category to which the entity belongs. The `@deffn` command possesses three predefined, specialized variations, `@defun`, `@defmac`, and `@defspec`, that specify the category for you: "Function", "Macro", and "Special Form" respectively. The `@defvr` command also is accompanied by several predefined, specialized variations for describing particular kinds of variables.

The template for a specialized definition, such as `@defun`, is similar to the template for a generalized definition, except that you don't have to specify the category:

```
@defun name arguments...
body-of-definition
@end defun
```

Thus,

```
@defun buffer-end flag
This function returns @code{(point-min)} if @var{flag}
is less than 1, @code{(point-max)} otherwise.
...
@end defun
```

produces

buffer-end *flag* Function
This function returns `(point-min)` if *flag* is less than 1, `(point-max)` otherwise. ...

See Section 13.5 [A Sample Function Definition], page 126, for a more detailed example of a function definition, including the use of `@example` inside of the definition.

The other specialized commands work like `@defun`.

13.2 Optional and Repeated Parameters

Some entities take optional or repeated parameters, which may be specified by a distinctive special glyph that uses square brackets and ellipses. For example, a special form often breaks its argument list into separate arguments in more complicated ways than a straightforward function.

An argument enclosed within square brackets is optional. Thus, the phrase '[*optional-arg*]' means that *optional-arg* is optional. An argument followed by an ellipsis is optional and may be repeated more than once. Thus, '*repeated-args*...' stands for zero or more arguments. Parentheses are used when several arguments are grouped into additional levels of list structure in Lisp.

Here is the `@defspec` line of an example of an imaginary special form:

foobar (*var* [*from to* [*inc*]]) *body*... Special Form

In this example, the arguments *from* and *to* are optional, but must both be present or both absent. If they are present, *inc* may optionally be specified as well. These arguments are grouped with the argument *var* into a list, to distinguish them from *body*, which includes all remaining elements of the form.

In a Texinfo source file, this `@defspec` line is written like this (except it would not be split over two lines, as it is in this example).

```
@defspec foobar (@var{var} [@var{from} @var{to}
     [@var{inc}]]) @var{body}@dots{}
```

The function is listed in the Command and Variable Index under '`foobar`'.

13.3 The Definition Commands

Texinfo provides more than a dozen definition commands, all of which are described in this section.

The definition commands automatically enter the name of the entity in the appropriate index: for example, `@deffn`, `@defun`, and `@defmac` enter function names in the index of functions; `@defvr` and `@defvar` enter variable names in the index of variables.

Although the examples that follow mostly illustrate Lisp, the commands can be used for other programming languages.

13.3.1 Functions and Similar Entities

This section describes the commands for describing functions and similar entities.

`@deffn` *category name arguments...*

The `@deffn` command is the general definition command for functions, interactive commands, and similar entities that may take arguments. You must choose a term to describe the category of entity being defined; for example, "Function" could be used if the entity is a function. The `@deffn` command is written at the beginning of a line and is followed on the same line by the category of entity being described, the name of this particular entity, and its arguments, if any. Terminate the definition with `@end deffn` on a line of its own.

For example,

```
@deffn Command forward-char nchars
Move point forward @var{nchars} characters.
@end deffn
```

shows a rather terse definition for a "command" named `forward-char` with one argument, *nchars*.

`@deffn` prints argument names such as *nchars* in italics or upper case, as if `@var` had been used, because we think of these names as metasyntactic variables—they stand for the actual argument values. Within the text of the description, write an argument name explicitly with `@var` to refer to the value of the argument. In the example above, we used '`@var{nchars}`' in this way.

The template for `@deffn` is:

```
@deffn category name arguments...
body-of-definition
@end deffn
```

`@defun` *name arguments...*

The `@defun` command is the definition command for functions. `@defun` is equivalent to '`@deffn Function` ...'.

For example,

```
@defun set symbol new-value
Change the value of the symbol symbol to new-value.
@end defun
```

shows a rather terse definition for a function `set` whose arguments are *symbol* and *new-value*. The argument names on the `@defun` line automatically appear in italics or upper case as if they were enclosed in `@var`. Terminate the definition with `@end defun` on a line of its own.

The template is:

```
@defun function-name arguments...
body-of-definition
@end defun
```

`@defun` creates an entry in the index of functions.

`@defmac` *name arguments...*

The `@defmac` command is the definition command for macros. `@defmac` is equivalent to '`@deffn Macro` ...' and works like `@defun`.

`@defspec` *name arguments...*

The `@defspec` command is the definition command for special forms. (In Lisp, a special form is an entity much like a function.) `@defspec` is equivalent to '`@deffn {Special Form}` ...' and works like `@defun`.

13.3.2 Variables and Similar Entities

Here are the commands for defining variables and similar entities:

`@defvr` *category name*

The `@defvr` command is a general definition command for something like a variable—an entity that records a value. You must choose a term to describe the category of entity being defined; for example, "Variable" could be used if the entity is a variable. Write the `@defvr` command at the beginning of a line and followed it on the same line by the category of the entity and the name of the entity.

Capitalize the category name like a title. If the name of the category contains spaces, as in the name 'User Option', write braces around it. Otherwise, the second word will be mistaken for the name of the entity. For example,

```
@defvr {User Option} fill-column
This buffer-local variable specifies
the maximum width of filled lines.
...
@end defvr
```

Terminate the definition with `@end defvr` on a line of its own.

The template is:

```
@defvr category name
body-of-definition
@end defvr
```

`@defvr` creates an entry in the index of variables for *name*.

`@defvar` *name*

The `@defvar` command is the definition command for variables. `@defvar` is equivalent to '`@defvr Variable` ...'.

For example,

```
@defvar kill-ring
...
@end defvar
```

The template is:

```
@defvar name
body-of-definition
@end defvar
```

`@defvar` creates an entry in the index of variables for *name*.

`@defopt` *name*

The `@defopt` command is the definition command for user options. `@defopt` is equivalent to '`@defvr {User Option}` ...' and works like `@defvar`.

13.3.3 Functions in Typed Languages

The `@deftypefn` command and its variations are for describing functions in C or any other language in which you must declare types of variables and functions.

`@deftypefn` *category data-type name arguments...*

The `@deftypefn` command is the general definition command for functions and similar entities that may take arguments and that are typed. The `@deftypefn` command is written at the beginning of a line and is followed on the same line by the category of entity being described, the type of the returned value, the name of this particular entity, and its arguments, if any.

For example,

```
@deftypefn {Library Function} int foobar (int @var{foo},
     float @var{bar})
...
@end deftypefn
```

(where the text before the "...", shown above as two lines, would actually be a single line in a real Texinfo file) produces the following in Info:

```
-- Library Function: int foobar (int FOO, float BAR)
...
```

In a printed manual, it produces:

`int` **foobar** (`int` *foo*, `float` *bar*) Library Function
...

This means that `foobar` is a "library function" that returns an `int`, and its arguments are *foo* (an `int`) and *bar* (a `float`).

The argument names that you write in `@deftypefn` are not subject to an implicit `@var`—since the actual names of the arguments in `@deftypefn` are typically scattered among data type names and keywords, Texinfo can't find them without help. Instead, you must write `@var` explicitly around the argument names. In the example above, the argument names are '`foo`' and '`bar`'.

The template for `@deftypefn` is:

```
@deftypefn category data-type name arguments ...
body-of-description
@end deftypefn
```

Note that if the *category* or *data type* is more than one word then it must be enclosed in braces to make it a single argument.

If you are describing a procedure in a language that has packages, such as Ada, you might consider using `@deftypefn` in a manner somewhat contrary to the convention described in the preceding paragraphs.

For example:

```
@deftypefn stacks private push (@var{s}:in out stack;
     @var{n}:in integer)
...
@end deftypefn
```

(The `@deftypefn` arguments are shown split into two lines, but would be a single line in a real Texinfo file.)

In this instance, the procedure is classified as belonging to the package `stacks` rather than classified as a 'procedure' and its data type is described as `private`. (The name of the procedure is `push`, and its arguments are *s* and *n*.)

`@deftypefn` creates an entry in the index of functions for *name*.

`@deftypefun` *data-type name arguments...*

The `@deftypefun` command is the specialized definition command for functions in typed languages. The command is equivalent to '`@deftypefn Function` ...'.

```
@deftypefun int foobar (int @var{foo}, float @var{bar})
...
@end deftypefun
```

produces the following in Info:

```
-- Function: int foobar (int FOO, float BAR)
...
```

and the following in a printed manual:

`int` **foobar** (`int` *foo*, `float` *bar*) Function
...

The template is:

`@deftypefun` *type name arguments...*
body-of-description
`@end deftypefun`

`@deftypefun` creates an entry in the index of functions for *name*.

13.3.4 Variables in Typed Languages

Variables in typed languages are handled in a manner similar to functions in typed languages. See Section 13.3.3 [Typed Functions], page 120. The general definition command `@deftypevr` corresponds to `@deftypefn` and the specialized definition command `@deftypevar` corresponds to `@deftypefun`.

`@deftypevr` *category data-type name*

The `@deftypevr` command is the general definition command for something like a variable in a typed language—an entity that records a value. You must choose a term to describe the category of the entity being defined; for example, "Variable" could be used if the entity is a variable.

The `@deftypevr` command is written at the beginning of a line and is followed on the same line by the category of the entity being described, the data type, and the name of this particular entity.

For example:

```
@deftypevr {Global Flag} int enable
...
@end deftypevr
```

produces the following in Info:

```
-- Global Flag: int enable
...
```

and the following in a printed manual:

`int` **enable** Global Flag
...

The template is:

```
@deftypevr category data-type name
body-of-description
@end deftypevr
```

`@deftypevr` creates an entry in the index of variables for *name*.

`@deftypefvar` *data-type name*

The `@deftypevar` command is the specialized definition command for variables in typed languages. `@deftypevar` is equivalent to '`@deftypevr Variable` ...'.

For example,

```
@deftypevar int foobar
...
@end deftypevar
```

produces the following in Info:

```
-- Variable: int foobar
...
```

and the following in a printed manual:

`int` **foobar** Variable
...

The template is:

```
@deftypevar data-type name
body-of-description
@end deftypevar
```

`@deftypevar` creates an entry in the index of variables for *name*.

13.3.5 Object-Oriented Programming

Here are the commands for formatting descriptions about abstract objects, such as are used in object-oriented programming. A class is a defined type of abstact object. An instance of a class is a particular object that has the type of the class. An instance variable is a variable that belongs to the class but for which each instance has its own value.

In a definition, if the name of a class is truly a name defined in the programming system for a class, then you should write an `@code` around it. Otherwise, it is printed in the usual text font.

`@defcv` *category class name*

The `@defcv` command is the general definition command for variables associated with classes in object-oriented programming. The `@defcv` command is followed by three arguments: the category of thing being defined, the class to which it belongs, and its name.

Thus,

```
@defcv {Class Option} Window  border-pattern
...
@end defcv
```

illustrates how you would write the first line of a definition of the `border-pattern` class option of the class `Window`.

The template is

```
@defcv category class name
...
@end defcv
```

`@defcv` creates an entry in the index of variables.

`@defivar` *class name*

The `@defivar` command is the definition command for instance variables in object-oriented programming. `@defivar` is equivalent to '`@defcv {Instance Variable} ...`'

The template is:

```
@defivar class instance-variable-name
body-of-definition
@end defivar
```

`@defivar` creates an entry in the index of variables.

`@defop` *category class name arguments...*

The `@defop` command is the general definition command for entities that may resemble methods in object-oriented programming. These entities take arguments, as functions do, but are associated with particular classes of objects.

For example, some systems have constructs called *wrappers* that are associated with classes as methods are, but that act more like macros than like functions. You could use `@defop Wrapper` to describe one of these.

Sometimes it is useful to distinguish methods and *operations*. You can think of an operation as the specification for a method. Thus, a window system might specify that all window classes have a method named `expose`; we would say that this window system defines an `expose` operation on windows in general. Typically, the operation has a name and also specifies the pattern of arguments; all methods that implement the operation must accept the same arguments, since applications that use the operation do so without knowing which method will implement it.

Often it makes more sense to document operations than methods. For example, window application developers need to know about the `expose` operation, but need not be concerned with whether a given class of windows has its own method to implement this operation. To describe this operation, you would write:

```
@defop Operation windows expose
```

The `@defop` command is written at the beginning of a line and is followed on the same line by the overall name of the category of operation, the name of the class of the operation, the name of the operation, and its arguments, if any.

The template is:

```
@defop category class name arguments...
body-of-definition
@end defop
```

`@defop` creates an entry, such as '`expose` on `windows`', in the index of functions.

`@defmethod` *class name arguments...*

The `@defmethod` command is the definition command for methods in object-oriented programming. A method is a kind of function that implements an operation for a particular class of objects and its subclasses. In the Lisp Machine, methods actually were functions, but they were usually defined with `defmethod`.

`@defmethod` is equivalent to '`@defop Method` ...'. The command is written at the beginning of a line and is followed by the name of the class of the method, the name of the method, and its arguments, if any.

For example,

```
@defmethod bar-class bar-method argument
...
@end defmethod
```

illustrates the definition for a method called `bar-method` of the class `bar-class`. The method takes an argument.

The template is:

```
@defmethod class method-name arguments...
body-of-definition
@end defmethod
```

`@defmethod` creates an entry, such as '`bar-method` on `bar-class`', in the index of functions.

13.3.6 Data Types

Here is the command for data types:

`@deftp` *category name attributes...*

The `@deftp` command is the generic definition command for data types. The command is written at the beginning of a line and is followed on the same line by the category, by the name of the type (which is a word like `int` or `float`, and then by names of attributes of objects of that type. Thus, you could use this command for describing `int` or `float`, in which case you could use `data type` as the category. (A data type is a category of certain objects for purposes of deciding which operations can be performed on them.)

In Lisp, for example, *pair* names a particular data type, and an object of that type has two slots called the CAR and the CDR. Here is how you would write the first line of a definition of `pair`.

```
@deftp {Data type} pair car cdr
...
@end deftp
```

The template is:

```
@deftp category name-of-type attributes...
body-of-definition
@end deftp
```

`@deftp` creates an entry in the index of data types.

13.4 Conventions for Writing Definitions

When you write a definition using `@deffn`, `@defun`, or one of the other definition commands, please take care to use arguments that indicate the meaning, as with the *count* argument to the `forward-word` function. Also, if the name of an argument contains the name of a type, such as *integer*, take care that the argument actually is of that type.

13.5 A Sample Function Definition

A function definition uses the `@defun` and `@end defun` commands. The name of the function follows immediately after the `@defun` command and it is followed, on the same line, by the parameter list.

Here is a definition from *The GNU Emacs Lisp Reference Manual*. (See section "Calling Functions" in *The GNU Emacs Lisp Reference Manual*.)

apply *function* &rest *arguments* Function

`apply` calls *function* with *arguments*, just like `funcall` but with one difference: the last of *arguments* is a list of arguments to give to *function*, rather than a single argument. We also say that this list is *appended* to the other arguments.

`apply` returns the result of calling *function*. As with `funcall`, *function* must either be a Lisp function or a primitive function; special forms and macros do not make sense in `apply`.

```
(setq f 'list)
     ⇒ list
(apply f 'x 'y 'z)
error  Wrong type argument: listp, z
(apply '+ 1 2 '(3 4))
     ⇒ 10
(apply '+ '(1 2 3 4))
     ⇒ 10

(apply 'append '((a b c) nil (x y z) nil))
     ⇒ (a b c x y z)
```

An interesting example of using `apply` is found in the description of `mapcar`.

In the Texinfo source file, this example looks like this:

```
@defun apply function &rest arguments
@code{apply} calls @var{function} with @var{arguments}, just like
@code{funcall} but with one difference: the last of @var{arguments} is a
list of arguments to give to @var{function}, rather than a single
argument.  We also say that this list is @dfn{appended} to the other
arguments.

@code{apply} returns the result of calling @var{function}.  As with
@code{funcall}, @var{function} must either be a Lisp function or a
primitive function; special forms and macros do not make sense in
@code{apply}.

@example
(setq f 'list)
     @result{} list
(apply f 'x 'y 'z)
@error{} Wrong type argument: listp, z
(apply '+ 1 2 '(3 4))
     @result{} 10
(apply '+ '(1 2 3 4))
     @result{} 10
```

```
(apply 'append '((a b c) nil (x y z) nil))
     @result{} (a b c x y z)
@end example

An interesting example of using @code{apply} is found in the description
of @code{mapcar}.@refill
@end defun
```

In this manual, this function is listed in the Command and Variable Index under `apply`.

Ordinary variables and user options are described using a format like that for functions except that variables do not take arguments.

14 Footnotes

A *footnote* is for a reference that documents or elucidates the primary text.[1]

In Texinfo, footnotes are created with the `@footnote` command. This command is followed immediately by a left brace, then by the text of the footnote, and then by a terminating right brace. The template is: `@footnote{`*text*`}`.

Footnotes may be of any length, but are usually short.

For example, this clause is followed by a sample footnote[2]; in the Texinfo source, it looks like this:

```
...a sample footnote @footnote{Here is the sample
footnote.}; in the Texinfo source...
```

In a printed manual or book, the reference mark for a footnote is a small, superscripted number; the text of the footnote is written at the bottom of the page, below a horizontal line.

In Info, the reference mark for a footnote is a pair of parentheses with the footnote number between them, like this: '`(1)`'.

Info has two footnote styles, which determine where the text of the footnote is located:

- In the *end* of node style, all the footnotes for a single node are placed at the end of that node. The footnotes are separated from the rest of the node by a line of dashes with the word '`Footnotes`' within it. Each footnote begins with an '`(`*n*`)`' reference mark.

  Here is an example of a single footnote in the end of node style:

---

[1] A footnote should complement or expand upon the primary text, but a reader should not need to read a footnote to understand the primary text. For a thorough discussion of footnotes, see *The Chicago Manual of Style*, which is published by the University of Chicago Press.

[2] Here is the sample footnote.

```
--------- Footnotes ---------

(1)  Here is a sample footnote.
```

- In the *separate* style, all the footnotes for a single node are placed in an automatically constructed node of their own. In this style, a "footnote reference" follows each '(*n*)' reference mark in the body of the node. The footnote reference is actually a cross reference and you use it to reach the footnote node.

  The name of the footnotes' node is constructed by appending '`-Footnotes`' to the name of the node that contains the footnotes. (Consequently, the footnotes' node for the '`Footnotes`' node is '`Footnotes-Footnotes`'!) The footnotes' node has an 'Up' node pointer that leads back to its parent node.

Here is how the first footnote in this manual looks after being formatted for Info in the separate node style:

```
File: texinfo.info  Node: Overview-Footnotes, Up: Overview

(1) Note that the first syllable of "texinfo" is pronounced like
"speck", not "hex". ...
```

A Texinfo file may be formatted into an Info file with either footnote style.

Use the `@footnotestyle` command to specify an Info file's footnote style. Write this command at the beginning of a line followed by an argument, either '`end`' for the end node style or '`separate`' for the separate node style. For example:

```
@footnotestyle end
```

or

```
@footnotestyle separate
```

Write an `@footnotestyle` command before or shortly after the end of header line at the beginning of a Texinfo file. (If you include the `@footnotestyle` command between the start of header and end of header lines, the region formatting commands will format footnotes as specified.)

If you do not specify a footnote style, the formatting commands will chose a default style.

This chapter contains two footnotes.

15 Creating Indices

Using Texinfo, you can generate indices without having to sort and collate entries manually. In an index, the entries are listed in alphabetical order, together with information on how to find the discussion of each entry. In a printed manual, this information consists of page numbers. In an Info file, this information is a menu item leading to the first node referenced.

Texinfo provides several predefined kinds of index: an index for functions, an index for variables, an index for concepts, and so on. You can combine indices or use them for other than their canonical purpose. If you wish, you can define your own indices.

15.1 Making Index Entries

When you are making index entries, it is good practice to think of the different ways people may look for something. Different people *do not* think of the same words when they look something up. A helpful index will have items indexed under all the different words that people may use. For example, someone might think it obvious that the two-letter names for indices should be listed under "Indices, two-letter names", since the word "Index" is the general concept. But another reader may remember the specific concept of two-letter names and search for the entry listed as "Two letter names for indices". A good index will have both entries and will help both kinds of user.

Like typesetting, the construction of an index is a highly skilled, professional art, the subtleties of which are not appreciated until you have to do it yourself.

See Section 4.1 [Printing an Index and Generating Index Menus], page 45, for information about printing an index at the end of a book or creating an index menu in an Info file.

15.2 Predefined Indices

Texinfo provides six predefined indices:

- A *concept index* listing concepts that are discussed.
- A *function index* listing functions (such as, entry points of libraries).
- A *variables index* listing variables (such as, global variables of libraries).

- A *keystroke index* listing keyboard commands.
- A *program index* listing names of programs.
- A *data type index* listing data types (such as, structures defined in header files).

Not every manual needs all of these. This manual has two indices: a concept index and an `@`-command index (that is actually the function index but is called a command index in the chapter heading). Two or more indices can be combined into one using the `@synindex` or `@syncodeindex` commands. See Section 15.4 [Combining Indices], page 134.

15.3 Defining the Entries of an Index

The data to make an index comes from many individual indexing commands scattered throughout the Texinfo source file. Each command says to add one entry to a particular index; after processing, it will give the current page number or node name as the reference.

An index entry consists of an indexing command at the beginning of a line followed, on the rest of the line, by the entry.

For example, this section begins with the following five entries for the concept index:

```
@cindex Defining indexing entries
@cindex Index entries
@cindex Entries for an index
@cindex Specifying index entries
@cindex Creating index entries
```

Each predefined index has its own indexing command—`@cindex` for the concept index, `@findex` for the function index, and so on.

The usual convention is to capitalize the first word of each index entry, unless that word is the name of a function, variable, or other such entitity that should not be capitalized. Thus, if you are documenting Emacs Lisp, your concept index entries are usually capitalized, but not your function index entries. However, if your concept index entries are consistently short (one or two words each) it may look better for each regular entry to start with a lower case letter. Which ever convention you adapt, please be consistent!

By default, entries for a concept index are printed in a small roman font and entries for the other indices are printed in a small `@code` font. You may change the way part of an entry is printed with

the usual Texinfo commands, such as `@file` for file names and `@emph` for emphasis (see Chapter 8 [Marking Text], page 81).

The six indexing commands for predefined indices are:

`@cindex` *concept*
: Make an entry in the concept index for *concept*.

`@findex` *function*
: Make an entry in the function index for *function*.

`@vindex` *variable*
: Make an entry in the variable index for *variable*.

`@kindex` *key*
: Make an entry in the key index for *key*.

`@pindex` *program*
: Make an entry in the program index for *program*.

`@tindex` *data type*
: Make an entry in the data type index for *data type*.

> **Caution:** Do not use a colon in an index entry. In Info, a colon separates the menu item name from the node name. An extra colon confuses Info. See Section 6.4.1 [Writing a Menu Item], page 62, for more information about the structure of a menu entry.

If the same name is indexed on several pages, all the pages are listed in the printed manual's index. However, **only** the **first** node referenced will appear in the index of an Info file. This means that it is best to write indices in which each entry will refer to only one place in the Texinfo file. Fortunately, this constraint is a feature rather than a loss since it means that the index will be easy to use. Otherwise, it can be easy to create an index which has many pages listed for an entry and you don't know which one you need. If you have two entries for one topic, change the topics slightly, or qualify them to indicate the difference.

You are not actually required to use the predefined indices for their canonical purposes. For example, suppose you wish to index some C preprocessor macros. You could put them in the function index along with actual functions, just by writing `@findex` commands for them; then, when you print the "function index" as an unnumbered chapter, you could give it the title 'Function and Macro Index' and all will be consistent for the reader. Or you could put the macros in with the data types by writing `@tindex` commands for them, and give that index a suitable title so the reader will understand. (See Section 4.1 [Printing Indices & Menus], page 45.)

15.4 Combining Indices

Sometimes you will want to combine two disparate indices such as functions and concepts, perhaps because you have few enough of one of them that a separate index for them would look silly.

You could put functions into the concept index by writing `@cindex` commands for them instead of `@findex` commands, and produce a consistent manual by printing the concept index with the title 'Function and Concept Index' and not printing the 'Function Index' at all; but this is not a robust procedure. It works only if your document is never included in part of or together with another document that is designed to have a separate function index; if your document were to be included with such a document, the functions from your document and those from the other would not end up together. Also, to make your function names appear in the right font in the concept index, you would have to enclose every one of them between `@code` and `@end code`.

What you should do instead when you want functions and concepts in one index is to index the functions with `@findex` as they should be, but use the `@syncodeindex` command to redirect these `@findex` commands to the concept index.

The `@syncodeindex` command takes two arguments: the name of an index to redirect, and the name of an index to redirect it to:

```
@syncodeindex from to
```

For this purpose, the indices are given two-letter names:

'`cp`' the concept index
'`vr`' the variable index
'`fn`' the function index
'`ky`' the key index
'`pg`' the program index
'`tp`' the data type index

Write an `@syncodeindex` command before or shortly after the end of header line at the beginning of a Texinfo file. For example, to merge a function index with a concept index, write the following:

```
@syncodeindex fn cp
```

This will cause all entries designated for the function index to go to the concept index instead.

The `@syncodeindex` command puts all the entries from the redirected index into the `@code` font, overriding whatever default font is used by the index to which the entries are redirected. This way, if you redirect function names from a function index into a concept index, all the function names are printed the `@code` font as you would expect.

The `@synindex` command is nearly the same as the `@syncodeindex` command, except that it does not put the redirected index into the `@code` font, but puts it in the roman font.

See Section 4.1 [Printing an Index and Generating Index Menus], page 45, for information about printing an index at the end of a book or creating an index menu in an Info file.

15.5 Defining New Indices

In addition to the predefined indices, you may use the `@defindex` and `@defcodeindex` commands to define new indices. These commands create new indexing `@`-commands with which you mark index entries. The `@defindex` command is used like this:

```
@defindex name
```

The name of an index should be a two letter word, such as '`au`'. For example,

```
@defindex au
```

This defines a new index, called the '`au`' index. At the same time, it creates a new indexing command, `@auindex`, that you can use to make index entries. Use the new indexing command just as you would use a predefined indexing command.

For example, here is a section heading followed by a concept index entry and two '`au`' index entries.

```
@section Cognitive Semantics
@cindex kinesthetic image schemas
@auindex Johnson, Mark
@auindex Lakoff, George
```

(Evidently, '`au`' serves here as a abbreviation for "author".) Texinfo constructs the new indexing

command by concatenating the name of the index with '`index`'; thus, defining an '`au`' index leads to the automatic creation of an `@auindex` command.

Use the `@printindex` command to print the index, as you do with the predefined indices. For example,

```
@node Author Index, Subject Index, , Top
@unnumbered Author Index

@printindex au
```

The `@defcodeindex` is like the `@defindex` command, except that in the printed output, it prints entries in an `@code` font, like the `@findex` command, rather than in a roman font, like the `@cindex` command.

You should define new indices within or right after the end-of-header line of a Texinfo file, before any `@synindex` or `@syncodeindex` commands (see Section 3.2 [Header], page 28).

16 Making and Preventing Breaks

Usually, a Texinfo file is processed both by TEX and by one of the Info formatting commands. Sometimes line, paragraph, or page breaks occur in the 'wrong' place in one or other form of output. You must ensure that text looks right both in the printed manual and in the Info file.

For example, in a printed manual, page breaks may occur awkwardly in the middle of an example; to prevent this, you can hold text together using a grouping command that keeps the text from being split across two pages. Conversely, you may want to force a page break where none would occur normally. Fortunately, problems like these do not often arise. When they do, use the following commands.

The break commands create line and paragraph breaks:

`@*` Force a line break.

`@sp` *n* Skip *n* blank lines.

The line-break-prevention command holds text together all on one line.

`@w{`*text*`}` Prevent *text* from being split across two lines.

The pagination commands apply only to printed output, since Info files do not have pages.

`@page` Start a new page in the printed manual.

`@group` Hold text together that must appear on one printed page.

`@need` *mils*
Start a new printed page if not enough space on this one.

16.1 `@*`: Generate Line Breaks

The `@*` command forces a line break in both the printed manual and in Info.

For example,

```
This line @* is broken @*in two places.
```

produces

```
This line
 is broken
in two places.
```

(Note that the space after the first `@*` command is faithfully carried down to the next line.)

The `@*` command is often used in a file's copyright page:

```
This is version 2.0 of the Texinfo documentation,@*
and is for ...
```

In this case, the `@*` command keeps TEX from stretching the line across the whole page in an ugly manner.

Do not write braces after an `@*` command; they are not needed.

Do not write an `@refill` command at the end of a paragraph containing an `@*` command; it will cause the paragraph to be refilled after the line break occurs, negating the effect of the line break.

16.2 `@w{`*text*`}`: Prevent Line Breaks

`@w{`*text*`}` outputs *text* and prohibits line breaks within *text*.

You can use the `@w` command to prevent TEX from automatically hyphenating a long name or phrase that accidentally falls near the end of a line.

```
You can copy GNU software from @w{@file{prep.ai.mit.edu}}.
```

produces

You can copy GNU software from '`prep.ai.mit.edu`'.

In the Texinfo file, you must write the `@w` command and its argument (all the affected text) all on one line.

Do not write an `@refill` command at the end of a paragraph containing an `@w` command; it will cause the paragraph to be refilled and may thereby negate the effect of the `@w` command.

16.3 `@sp` *n*: Insert Blank Lines

A line beginning with and containing only `@sp` *n* generates *n* blank lines of space in both the printed manual and the Info file. `@sp` also forces a paragraph break. For example,

```
@sp 2
```

generates two blank lines.

The `@sp` command is most often used in the title page.

16.4 `@page`: Start a New Page

A line containing only `@page` starts a new page in a printed manual. The command has no effect on Info files since they are not paginated. An `@page` command is often used in the title section of a Texinfo file to start the copyright page.

16.5 `@group`: Prevent Page Breaks

The `@group` command (on a line by itself) is used inside of an `@example` or similar construct to begin an unsplittable vertical group, which will appear entirely on one page in the printed output. The group is terminated by a line containing only `@end group`. These two lines produce no output of their own, and in the Info file output they have no effect at all.

Although `@group` would make sense conceptually in a wide variety of contexts, its current implementation works reliably only within `@example` and variants, and within `@quotation`, `@display`, `@format`, `@flushleft` and `@flushright`. (What all these commands have in common is that they turn off vertical spacing between "paragraphs".) In other contexts, `@group` can cause anomalous vertical spacing. See Chapter 10 [Quotations and Examples], page 95.

The `@group` command is most often used to hold an example together on one page. In this Texinfo manual, about 100 examples contain text that is enclosed between `@group` and `@end group`.

If you forget to end a group, you may get strange and unfathomable error messages when you run TEX. This is because TEX keeps trying to put the rest of the Texinfo file onto the one page and does not start to generate error messages until it has processed considerable text. It is a good rule of thumb to look for a missing `@end group` if you get incomprehensible error messages in TEX.

16.6 `@need` *mils*: Prevent Page Breaks

A line containing only `@need` *n* starts a new page in a printed manual if fewer than *n* mils (thousandths of an inch) remain on the current page. Do not use braces around the argument *n*. The `@need` command has no effect on Info files since they are not paginated.

This paragraph is preceded by an `@need` command that tells TEX to start a new page if fewer than 300 mils (nearly one-third inch) remain on the page. It looks like this:

```
@need 300
This paragraph is preceded by ...
```

The `@need` command is useful for preventing orphans (single lines at the bottoms of printed pages).

17 Conditionally Visible Text

Sometimes it is good to use different text for a printed manual and its corresponding Info file. In this case, you can use the conditional commands to specify which text is for the printed manual and which is for the Info file.

`@ifinfo` begins text that should be ignored by TEX when it typesets the printed manual. The text appears only in the Info file. The `@ifinfo` command should appear on a line by itself. End the Info-only text with a line containing `@end ifinfo` by itself. At the beginning of a Texinfo file, the Info permissions are contained within a region marked by `@ifinfo` and `@end ifinfo`. (See Section 3.3 [Info Summary and Permissions], page 33.)

The `@iftex` and `@end iftex` commands are used similarly but to delimit text that will appear in the printed manual but not in the Info file.

For example,

```
@iftex
This text will appear only in the printed manual.
@end iftex

@ifinfo
However, this text will appear only in Info.
@end ifinfo
```

The preceding example produces the following. Note how you only see one of the two lines, depending on whether you are reading the Info version or the printed version of this manual.

This text will appear only in the printed manual.

The `@titlepage` command is a special variant of `@iftex` that is used for making the title and copyright pages of the printed manual. (See Section 3.4.1 [`@titlepage`], page 34.)

17.1 Using Ordinary TEX Commands

Inside a region delineated by `@iftex` and `@end iftex`, you can embed some PlainTEX commands. Info will ignore these commands since they are only in that part of the file that is seen by TEX. The TEX commands are the same as any TEX commands except that an '`@`' replaces the '`\`' used

by TEX. For example, in the `@titlepage` section of a Texinfo file, the TEX command `@vskip` is used to format the copyright page. (The `@titlepage` command causes Info to ignore the region automatically, as it does with the `@iftex` command.)

However, many features of PlainTEX will not work, as they are overriden by features of Texinfo.

You can enter PlainTEX completely, and use '\' in the TEX commands, by delineating a region with the `@tex` and `@end tex` commands. (The `@tex` command also causes Info to ignore the region, like the `@iftex` command.)

For example, here is some mathematics:

```
@tex
$\bigl(x\in A(n)\bigm|x\in B(n)\bigr)$
@end tex
```

The output of this example will appear only in the printed manual. If you are reading this in Info, you will not see anything after this paragraph. In the printed manual, the above mathematics looks like this:

$$\bigl(x \in A(n) \bigm| x \in B(n)\bigr)$$

18 Printing Hardcopy

There are three major shell commands for printing hardcopy of a Texinfo file. One is for formatting the file, the second is for sorting the index, and the third is for printing the formatted document. When you use the shell commands, you can either work directly in the operating system shell or work within a shell inside of GNU Emacs.

Instead of shell commands, you can use commands provided by Texinfo mode. In addition to three commands to to format a file, sort the indices, and print the result, Texinfo mode offers key bindings for commands to recenter the output buffer, show the print queue, and delete a job from the print queue.

The typesetting program TEX is used for formatting a Texinfo file.[1] TEX is a very powerful typesetting program and, if used right, does an exceptionally good job.

The `makeinfo`, `texinfo-format-region`, and `texinfo-format-buffer` commands read the very same `@`-commands in the Texinfo file as does TEX, but process them differently to make an Info file. (See Chapter 19 [Creating an Info File], page 153.)

18.1 How to Print Using Shell Commands

Format the Texinfo file with the shell command `tex` followed by the name of the Texinfo file. This produces a formatted DVI file as well as several auxiliary files containing indices, cross references, etc. The DVI file (for *DeVice Independent* file) can be printed on a wide variety of printers.

The `tex` formatting command itself does not sort the indices; it writes an output file of unsorted index data. This is a misfeature of TEX. Hence, to generate a printed index, you first need a sorted index to work from. The `texindex` command sorts indices.[2]

---

[1] If you do not have TEX, you can use the `texi2roff` program for formatting.

[2] The source file '`texindex.c`' comes as part of the standard GNU distribution and is usually installed when Emacs is installed.

The `tex` formatting command outputs unsorted index files under names that obey a standard convention. These names are the name of your main input file to the `tex` formatting command, with everything after the first period thrown away, and the two letter names of indices added at the end. For example, the raw index output files for the input file '`foo.texinfo`' would be '`foo.cp`', '`foo.vr`', '`foo.fn`', '`foo.tp`', '`foo.pg`' and '`foo.ky`'. Those are exactly the arguments to give to `texindex`. Or else, you can use '`??`' as "wild-cards" and give the command in this form:

```
texindex foo.??
```

This command will run `texindex` on all the unsorted index files, including any that you have defined yourself using `@defindex` or `@defcodeindex`. (You may execute '`texindex foo.??`' even if there are similarly named files with two letter extensions that are not index files, such as '`foo.el`'. The `texindex` command reports but otherwise ignores such files.)

For each file specified, `texindex` generates a sorted index file whose name is made by appending '`s`' to the input file name. The `@printindex` command knows to look for a file of that name. `texindex` does not alter the raw index output file.

After you have sorted the indices, you need to rerun the `tex` formatting command on the Texinfo file. This regenerates a formatted DVI file with up-to-date index entries.[3]

To summarize, this is a three step process:

1. Run the `tex` formatting command on the Texinfo file. This generates the formatted DVI file as well as the raw index files with two letter extensions.
2. Run the shell command `texindex` on the raw index files to sort them. This creates the corresponding sorted index files.
3. Rerun the `tex` formatting command on the Texinfo file. This regenerates a formatted DVI file with the index entries in the correct order. This second run also makes all the cross references correct as well. (The tables of contents are always correct.)

---

[3] If you use more than one index and have cross references to an index other than the first, you must run `tex` *three times* to get correct output: once to generate raw index data; again (after `texindex`) to output the text of the indices and determine their true page numbers; and a third time to output correct page numbers in cross references to them. However, cross references to indices are rare.

You need not run `texindex` each time after you run the `tex` formatting. If you don't, on the next run, the `tex` formatting command will use whatever sorted index files happen to exist from the previous use of `texindex`. This is usually OK while you are debugging.

Rather than type the `tex` and `texindex` commands yourself, you can use `texi2dvi`. This shell script is designed to simplify the `tex`—`texindex`—`tex` sequence by figuring out whether index files and DVI files are up-to-date. It runs `texindex` and `tex` only when necessary.

The syntax for `texi2dvi` is like this (where '`%`' is the shell prompt):

```
% texi2dvi filenames...
```

Finally, you can print a DVI file with the DVI print command. The precise command to use depends on the system; '`lpr -d`' is common. The DVI print command may require a file name without any extension or with a '`.dvi`' extension.

The following commands, for example, sort the indices, format, and print the *Bison Manual* (where '`%`' is the shell prompt):

```
% tex bison.texinfo
% texindex bison.??
% tex bison.texinfo
% lpr -d bison.dvi
```

(Remember that the shell commands may be different at your site; but these are commonly used versions.)

18.2 Printing from an Emacs Shell

You can give formatting and printing commands from a shell within GNU Emacs. To create a shell within Emacs, type `M-x shell`. In this shell, you can format and print the document. See Section 18.1 [How to Print Using Shell Commands], page 143, for details.

You can switch to and from the shell buffer while `tex` is running and do other editing. If you are formatting a long document on a slow machine, this can be very convenient.

You can also use `texi2dvi` from an Emacs shell. (See Section 18.1 [How to Print Using Shell Commands], page 143.)

18.3 Formatting and Printing in Texinfo Mode

Texinfo mode provides several predefined key commands for TEX formatting and printing. These include commands for sorting indices, looking at the printer queue, killing the formatting job, and recentering display of the buffer in which the operations occur.

`C-c C-t C-r`
`M-x texinfo-tex-region`
: Run TEX on the current region.

`C-c C-t C-b`
`M-x texinfo-tex-buffer`
: Run TEX on the current buffer.

`C-c C-t C-i`
`M-x texinfo-texindex`
: Sort the indices of a Texinfo file formatted with `texinfo-tex-region` or `texinfo-tex-buffer`.

`C-c C-t C-p`
`M-x texinfo-tex-print`
: Print a DVI file that was made with `texinfo-tex-region` or `texinfo-tex-buffer`.

`C-c C-t C-q`
`M-x texinfo-show-tex-print-queue`
: Show the print queue.

`C-c C-t C-d`
`M-x texinfo-delete-from-tex-print-queue`
: Delete a job from the print queue; you will be prompted for the job number shown by a preceding `C-c C-t C-q` command (`texinfo-show-tex-print-queue`).

`C-c C-t C-k`
`M-x texinfo-kill-tex-job`
: Kill the shell created by `texinfo-tex-region` or `texinfo-tex-buffer`.

`C-c C-t C-l`
`M-x texinfo-recenter-tex-output-buffer`
: Redisplay the shell buffer in which the TEX printing and formatting commands are run to show its most recent output.

Thus, the usual sequence of commands for formatting a buffer is as follows (with comments to the right):

`C-c C-t C-b`	Run TEX on the buffer.
`C-c C-t C-i`	Sort the indices.
`C-c C-t C-b`	Rerun TEX to regenerate indices.
`C-c C-t C-p`	Print the DVI file.
`C-c C-t C-q`	Display the printer queue.

The Texinfo mode TEX formatting commands start a subshell in Emacs called the '`*texinfo-tex-shell*`'. The `texinfo-tex-command`, `texinfo-texindex-command`, and `tex-dvi-print-command` commands are all run in this shell.

You can watch the commands operate in the '`*texinfo-tex-shell*`' buffer, and you can switch to and from and use the '`*texinfo-tex-shell*`' buffer as you would any other shell buffer.

The formatting and print commands depend on the values of several variables. The default values are:

Variable	Default value
`texinfo-tex-command`	`"tex"`
`texinfo-texindex-command`	`"texindex"`
`texinfo-tex-shell-cd-command`	`"cd"`
`texinfo-tex-dvi-print-command`	`"lpr -d"`
`texinfo-show-tex-queue-command`	`"lpq"`
`texinfo-delete-from-print-queue-command`	`"lprm"`
`texinfo-start-of-header`	`"%**start"`
`texinfo-end-of-header`	`"%**end"`
`texinfo-tex-trailer`	`"@bye"`

The default values of `texinfo-tex-command` and `texinfo-texindex-command` are set in the '`texnfo-tex.el`' file.

You can change the values of these variables with the `M-x edit-options` command (see section "Editing Variable Values" in *The GNU Emacs Manual*), with the `M-x set-variable` command (see section "Examining and Setting Variables" in *The GNU Emacs Manual*), or with your '`.emacs`' initialization file (see section "Init File" in *The GNU Emacs Manual*).

18.4 Using the Local Variables List

Yet another way to apply the TEX formatting command to a Texinfo file is to put that command in a *local variables list* at the end of the Texinfo file. You can then specify the TEX formatting command as a `compile-command` and have Emacs run the TEX formatting command by typing `M-x compile`. This creates a special shell called the '`*compilation buffer*`' in which Emacs runs the

compile command. For example, at the end of the 'gdb.texinfo' file, after the @bye, you would put the following:

```
@c Local Variables:
@c compile-command: "tex gdb.texinfo"
@c End:
```

This technique is most often used by programmers who also compile programs this way. (See section "Compilation" in *The GNU Emacs Manual*.)

18.5 TEX Formatting Requirements Summary

Every Texinfo file that is to be input to TEX must begin with a \input command and contain a @settitle command:

```
\input texinfo
@settitle name-of-manual
```

The first command instructs TEX to load the macros it needs to process a Texinfo file and the second command specifies the title of printed manual.

Every Texinfo file must end with a line that terminates TEX processing and forces out unfinished pages:

```
@bye
```

Strictly speaking these three lines are all a Texinfo file needs for TEX, besides the body. (The @setfilename line is the only line that a Texinfo file needs for Info formatting.)

Usually, the file's first line contains an '@c -*-texinfo-*-' comment that causes Emacs to switch to Texinfo mode when you edit the file. In addition, the beginning usually includes an @setfilename for Info formatting, an @setchapternewpage command, a title page, a copyright page, and permissions. Besides an @bye, the end of a file usually includes indices and a table of contents.

See Section 3.2.5 [@setchapternewpage], page 31,
Appendix C [Page Headings], page 183,
Section 3.4 [Titlepage & Copyright Page], page 34,

Section 4.1 [Index Menus and Printing an Index], page 45, and
Section 4.2 [Contents], page 46.

18.6 Preparing for Use of TEX

TEX needs to know where to find the '`texinfo.tex`' file that that you have told it to input with the '`\input texinfo`' command at the beginning of the first line. The '`texinfo.tex`' file tells TEX how to handle `@`-commands.

Usually, the '`texinfo.tex`' file is put in the default directory that contains TEX macros, namely the '`/usr/lib/tex/macros`' directory, when GNU Emacs is installed. In this case, TEX will find the file and you don't have to do anything special. Alternatively, you can put '`texinfo.tex`' in the directory in which the Texinfo source file is located, and TEX will find it there.

However, if you may want to specify the location of the `\input` file yourself. One way to do this is to write the complete path for the file after the `\input` command. Another way is to set the `TEXINPUTS` environment variable in your '`.cshrc`' or '`.profile`' file. The `TEXINPUTS` environment variable will tell TEX where to find the '`texinfo.tex`' file and any other file that you might want TEX to use.

Whether you use a '`.cshrc`' or '`.profile`' file depends on whether you use `csh`, `sh`, or `bash` for your shell command interpreter. When you use `csh`, it looks to the '`.cshrc`' file for initialization information, and when you use `sh` or `bash`, it looks to the '`.profile`' file.

In a '`.cshrc`' file, you could use the following `csh` command sequence:

```
setenv TEXINPUTS .:/usr/me/mylib:/usr/lib/tex/macros
```

In a '`.profile`' file, you could use the following `sh` command sequence:

```
TEXINPUTS=.:/usr/me/mylib:/usr/lib/tex/macros
export TEXINPUTS
```

This would cause TEX to look for '`\input`' file first in the current directory, indicated by the '`.`', then in a hypothetical user's '`me/mylib`' directory, and finally in the system library.

18.7 Overfull "Hboxes"

TEX is sometimes unable to typeset a line without extending it into the right margin. This can occur when TEX comes upon what it interprets as a long word that it cannot hyphenate, such as an electronic mail network address or a very long title. When this happens, TEX prints an error message like this:

```
Overfull \hbox (20.76302pt too wide)
```

(In TEX, lines are in "horizontal boxes", hence the term, "hbox".)

TEX also provides the line number in the Texinfo source file and the text of the offending line, which is marked at all the places that TEX knows how to hyphenate words.

If the Texinfo file has an overfull hbox, you can rewrite the sentence so the overfull hbox does not occur, or you can decide to leave it. A small excursion into the right margin often does not matter and may not even be noticeable.

However, if you do leave an overfull hbox, unless told otherwise, TEX will print a large, ugly, black rectangle beside the line. This is so you will notice the location of the problem if you are correcting a draft. To prevent such a monstrosity from marring your final printout, put the following in the beginning of the Texinfo file on a of its own, before the `@titlepage` command:

```
@finalout
```

18.8 Printing "Small" Books

By default, TEX typesets pages for printing in an 8.5 by 11 inch format. However, you can direct TEX to typeset a document in a 7 by 9.25 inch format that is suitable for bound books by inserting the following command on a line by itself at the beginning of the Texinfo file, before the `@setchapternewpage` command:

```
@smallbook
```

(Since regular sized books are often about 7 by 9.25 inches, this command might better have been called the `@regularbooksize` command, but it came to be called the `@smallbook` command by comparison to the 8.5 by 11 inch format.)

The Free Software Foundation distributes printed copies of the *GNU Emacs Manual* in the "small" book size. See Section 10.6 [`@smallexample` and `@smalllisp`], page 99, for information about commands that make it easier to produce examples for a smaller manual.

19 Creating an Info File

`makeinfo` is a utility you can use from a shell to format a Texinfo file; `texinfo-format-region` and `texinfo-format-buffer` are Emacs commands that you can also use for formatting.

A Texinfo file must possess an `@setfilename` line near its beginning, otherwise the formatting commands will fail.

For information on installing the Info file in the Info system, see Chapter 20 [Installing an Info File], page 161.

The `makeinfo` utility creates an Info file from a Texinfo source file more quickly than either of the Emacs formatting commands and provides better error messages. It is recommended. The `texinfo-format-region` and the `texinfo-format-buffer` commands are obsolete and are useful only if you cannot run `makeinfo`, which is written in C.

`makeinfo` is a C program that is independent of Emacs. You do not have to run Emacs to use `makeinfo`, which means you can use `makeinfo` on machines that are too small to run Emacs.

In addition, `makeinfo` automatically fills paragraphs (as if every paragraph were followed by an `@refill` command). `texinfo-format-region` and `texinfo-format-buffer` do not automatically fill paragraphs. (See Appendix E [Refilling Paragraphs], page 199.)

You can run `makeinfo` in any one of three ways: from an operating system shell, from a shell inside of Emacs, or, in Texinfo mode in Emacs, by typing a key command.

19.1 Running `makeinfo` from a Shell

To create an Info file from a Texinfo file, type `makeinfo` followed by the name of the Texinfo file. Thus, to create the Info file for Bison, type the following at the shell prompt (where '`%`' is the prompt):

```
% makeinfo bison.texinfo
```

(You can run a shell inside of Emacs by typing `M-x shell`.)

19.2 Options for `makeinfo`

The `makeinfo` command takes several options. Most often, options are used to set the value of the fill column and specify the footnote style. Each command line option is a word preceded by a '`+`'. You can use abbreviations for the option names as long as they are unique.

For example, you could use the following command to create an Info file for '`bison.texinfo`' in which each line is filled to only 68 columns (where '`%`' is the prompt):

```
% makeinfo +fill-column=68 bison.texinfo
```

You can write two or more options in sequence, like this:

```
makeinfo +no-split +fill-column=70 ...
```

This would keep the Info file together as one, possibly very long, file set the fill column to 70.

If you wish to discover which version of `makeinfo` you are using, type:

```
% makeinfo +version
```

The options are:

`+error-limit=`*limit*
: Set the maximum number of errors that `makeinfo` will report before exiting (on the assumption that continuing would be useless). The default number of errors that can be reported before `makeinfo` gives up is 100.

`+fill-column=`*width*
: Specify the maximum number of columns in a line; this is the right-hand edge of a line. Paragraphs that are filled will be filled to this width. (Filling is the process of breaking up and connecting lines so that lines are the same length as or shorter than the number specified as the fill column. Lines are broken between words.)

`+footnote-style=`*style*
: Set the footnote style to *style*, either '`end`' for the end node style or '`separate`' for the separate node style. The value set by this option overrides the value set in a Texinfo file by an `@footnotestyle` command. When the footnote style is '`separate`', `makeinfo` makes a new node containing the footnotes found in the current node. When the footnote style is '`end`', `makeinfo` places the footnote references at the end of the current node.

`+no-pointer-validation`
: Suppress the pointer-validation phase of `makeinfo`. Normally, after a Texinfo file is processed, some consistency checks are made to ensure that cross references can be resolved, etc. See Section 19.3 [Pointer Validation], page 155.

`+no-split`
: Suppress the splitting stage of `makeinfo`. Normally, large output files (where the size is greater than 70k bytes) are split into smaller subfiles, each one approximately 50k bytes. If you specify '`+no-split`', `makeinfo` will not split up the output file.

`+no-warn`
: Suppress the output of warning messages. This does *not* suppress the output of error messages, only warnings. You might want this if the file you are creating has examples of Texinfo cross references within it, and the nodes that are referenced don't actually exist.

`+paragraph-indent=`*indent*
: Set the paragraph indentation style to *indent*. The value set by this option overrides the value set in a Texinfo file by an `@paragraph-indent` command. The value of *indent* is interpreted as follows:

  - If the value of *indent* is '`asis`', do not change the existing indentation at the starts of paragraphs.
  - If the value of *indent* zero, delete any existing indentation.
  - If the value of *indent* is greater than zero, indent each paragraph by that number of spaces.

`+reference-limit=`*limit*
: Set the value of the number of references to a node that `makeinfo` will make without reporting a warning. If a node has more than this number of references in it, `makeinfo` will make the references but also report a warning.

`+verbose`
: Cause `makeinfo` to display messages saying what it is doing. Normally, `makeinfo` only outputs messages if there are errors or warnings.

`+version`
: Report the version number of this copy of `makeinfo`.

19.3 Pointer Validation

If you do not suppress pointer-validation (by using the '`+no-pointer-validation`' option), `makeinfo` will check the validity of the final Info file. Mostly, this means ensuring that nodes you have referenced really exist. Here is a complete list of what is checked:

1. If a 'Next', 'Previous', or 'Up' node reference is a reference to a node in the current file and is not an external reference such as to '`(dir)`', then the referenced node must exist.
2. In every node, if the 'Previous' node is different from the 'Up' node, then the 'Previous' node must also be pointed to by a 'Next' node.

3. Every node except the 'Top' node must have an 'Up' pointer.
4. The node referenced by an 'Up' pointer must contain a reference to the current node in some manner other than through a 'Next' reference. This includes menu items and cross references.
5. If the 'Next' reference of a node is not the same as the 'Next' reference of the 'Up' reference, then the node referenced by the 'Next' pointer must have a 'Previous' pointer that points back to the current node. This rule allows the last node in a section to point to the first node of the next chapter.

19.4 Running `makeinfo` inside Emacs

You can run `makeinfo` in GNU Emacs by using either the `makeinfo-region` or the `makeinfo-buffer` commands. In Texinfo mode, the commands are bound to `C-c C-m C-r` and `C-c C-m C-b` by default.

`C-c C-m C-r`
`M-x makeinfo-region`
: Format the current region for Info.

`C-c C-m C-b`
`M-x makeinfo-buffer`
: Format the current buffer for Info.

When you invoke either `makeinfo-region` or `makeinfo-buffer`, Emacs prompts for a file name, offering the name of the visited file as the default. You can edit the default file name in the minibuffer if you wish, before typing `RET` to start the `makeinfo` process.

The Emacs `makeinfo-region` and `makeinfo-buffer` commands run the `makeinfo` program in a temporary shell buffer. If `makeinfo` finds any errors, Emacs displays the error messages in the temporary buffer.

You can parse the error messages by typing `C-x ‘` (`next-error`). This causes Emacs to go to and position the cursor on the line in the Texinfo source that `makeinfo` thinks caused the error. See section "Running `make` or Compilers Generally" in *The GNU Emacs Manual*, for more information about using the `next-error` command.

In addition, you can kill the shell in which the `makeinfo` command is running or make the shell buffer display its most recent output.

`C-c C-m C-k`
`M-x makeinfo-kill-job`
: Kill the current running `makeinfo` job created by `makeinfo-region` or `makeinfo-buffer`.

`C-c C-m C-l`
`M-x makeinfo-recenter-output-buffer`
: Redisplay the `makeinfo` shell buffer to display its most recent output.

(Note that the parallel commands for killing and recentering a TEX job are `C-c C-t C-k` and `C-c C-t C-l`. See Section 18.3 [Texinfo Mode Printing], page 146.)

You can specify options for `makeinfo` by setting the `makeinfo-options` variable with either the `M-x edit-options` or the `M-x set-variable` command, or by setting the variable in your with your '`.emacs`' initialization file.

See section "Editing Variable Values" in *The GNU Emacs Manual*,
section "Examining and Setting Variables" in *The GNU Emacs Manual*,
section "Init File" in *The GNU Emacs Manual*, and Section 19.2 [Options for `makeinfo`], page 154.

19.5 The `texinfo-format`... Commands

In GNU Emacs in Texinfo mode, you can format part or all of a Texinfo file with the `texinfo-format-region` command. This formats the current region and displays the formatted text in a temporary buffer called '`*Info Region*`'.

Similarly, you can format a buffer with the `texinfo-format-buffer` command. This command creates a new buffer and generates the Info file in it. Typing `C-x C-s` will save the Info file under the name specified by the `@setfilename` line which must be near the beginning of the Texinfo file.

`C-c C-e C-r`
`texinfo-format-region`
: Format the current region for Info.

`C-c C-e C-b`
`texinfo-format-buffer`
: Format the current buffer for Info.

The `texinfo-format-region` and `texinfo-format-buffer` commands provide you with some

error checking; and other functions provide you with further help in finding formatting errors. These procedures are described in an appendix. See Appendix D [Catching Mistakes], page 189. However, the `makeinfo` program is faster and provides better error checking.

19.6 Tag Files and Split Files

If a Texinfo file has more than 30,000 bytes, `texinfo-format-buffer` automatically creates a *tag table* for its Info file; `makeinfo` always creates a tag table. With a tag table, Info can jump to new nodes more quickly than it can otherwise.

In addition, if the Texinfo file contains more than about 70,000 bytes, `texinfo-format-buffer` and `makeinfo` split the large Info file into shorter *indirect* subfiles of about 50,000 bytes each. Big files are split into smaller files so that Emacs does not have to make a large buffer to hold the whole of a large Info file; instead, Emacs allocates just enough memory for the small, split off file that is needed at the time. This way, Emacs avoids wasting memory when you run Info. (Before splitting was implemented, Info files were always kept short and *include* files were designed as a way to create a single, large printed manual out of the smaller Info files. See Appendix B [Include Files], page 177, for more information. Include files are still used for very large documents, such as *The Emacs Lisp Reference Manual*, in which each chapter is a separate file.)

When a file is split, Info itself makes use of a shortened version of the original file that contains just the tag table and references to the files that were split off. The split off files are called *indirect* files.

The split off files have names that are created by appending '`-1`', '`-2`', '`-3`' and so on to the file names specified by the `@setfilename` command. The shortened version of the original file continues to have the name specified by `@setfilename`.

At one stage in writing this document, for example, the Info file was saved as '`test-texinfo`' and that file looked like this:

```
Info file: test-texinfo,    -*-Text-*-
produced by texinfo-format-buffer
from file: new-texinfo-manual.texinfo

^_
Indirect:
test-texinfo-1: 102
test-texinfo-2: 50422
test-texinfo-3: 101300
^_^L
Tag table:
(Indirect)
Node: overview^?104
Node: info file^?1271
Node: printed manual^?4853
Node: conventions^?6855
...
```

(But ‘`test-texinfo`’ had far more nodes than are shown here.) Each of the split off, indirect files, ‘`test-texinfo-1`’, ‘`test-texinfo-2`’, and ‘`test-texinfo-3`’, is listed in this file after the line that says ‘`Indirect:`’. The tag table is listed after the line that says ‘`Tag table:`’.

If you are using `texinfo-format-buffer` to create Info files, you may want to run the `Info-validate` command. (The `makeinfo` command does such a good job on its own, you do not need `Info-validate`.) However, you cannot run the `M-x Info-validate` node-checking command on indirect files. For information on how to prevent files from being split and how to validate the structure of the nodes, see Section D.5.1 [Using Info-validate], page 195.

20 Installing an Info File

Info files are usually kept in the '.../emacs/info' directory. (You can find the location of this directory within Emacs by typing C-h i to enter Info and then typing C-x C-f to see the full pathname to the 'info' directory.)

For Info to work, the 'info' directory must contain a file that serves as a top level directory for the Info system. By convention, this file is called 'dir'. The 'dir' file is itself an Info file. It contains the top level menu for all the Info files in the system. The menu looks like this:

```
* Menu:

* Info:    (info).     Documentation browsing system.
* Emacs:   (emacs).    The extensible, self-documenting
                       text editor.
* Texinfo: (texinfo).  With one source file, make
                       either a printed manual using
                       TeX or an Info file.
...
```

Each of these menu entries points to the 'Top' node of the Info file that is named in parentheses. (The menu entry does not have to specify the 'Top' node, since Info goes to the 'Top' node if no node name is mentioned. See Section 6.5 [Nodes in Other Info Files], page 64.)

Thus, the 'Info' entry points to the 'Top' node of the 'info' file and the 'Emacs' entry points to the 'Top' node of the 'emacs' file.

In each of the Info files, the 'Up' pointer of the 'Top' node refers back to the dir file. For example, the node line for the 'Top' node of the Emacs manual looks like this:

```
File: emacs  Node: Top, Up: (DIR), Next: Distrib
```

(Note that in this case, the file name is written in upper case letters—it can be written in either upper or lower case. Info has a feature that it will change the case of the file name to lower case if it cannot find the name as written.)

20.1 Listing a New Info File

To add a new Info file to your system, add the name to the menu in the '`dir`' file by editing the '`dir`' file by hand. Also, put the new Info file in the '`.../emacs/info`' directory. For example, if you were adding documentation for GDB, you would make the following new entry:

```
* GDB: (gdb).           The source-level C debugger.
```

The first item is the menu item name; it is followed by a colon. The second item is the name of the Info file, in parentheses; it is followed by a period. The third part of the entry is the description of the item.

Conventionally, the name of an Info file has a '`.info`' extension. Thus, you might list the name of the file like this:

```
* GDB: (gdb.info).           The source-level C debugger.
```

However, Info will look for a file with a '`.info`' extension if it does not find the file under the name given in the menu. This means that you can write to '`gdb.info`' in a menu as '`gdb`', as shown in the first example. This looks better.

20.2 Info Files in Other Directories

If an Info file is not in the '`info`' directory, there are two ways to specify its location:

- Write the menu's second part as a pathname, or;
- Specify an environment variable in your '`.profile`' or '`.cshrc`' initialization file. (Only you and others with the same environment variable will be able to find Info files whose location is specified this way.)

For example, to reach a test file in the '`~bob/manuals`' directory, you could add an entry like this to the menu in the '`dir`' file:

```
* Test: (~bob/manuals/info-test).  Bob's own test file.
```

In this case, the absolute file name of the '`info-test`' file is written as the second item of the menu entry.

Alternatively, you may set the `INFOPATH` environment variable in your '`.cshrc`' or '`.profile`' file. The `INFOPATH` environment variable will tell Info where to look.

If you use `sh` or `bash` for your shell command interpreter, you must set the `INFOPATH` environment variable in the '`.profile`' initialization file; but if you use `csh`, you must set the variable in the '`.cshrc`' initialization file. The two files require slightly different command formats.

- In a '`.cshrc`' file, you could set the `INFOPATH` variable as follows:

```
setenv INFOPATH .:~bob/manuals:/usr/local/emacs/info
```

- In a '`.profile`' file, you would achieve the same effect by writing:

```
INFOPATH=.:~bob/manuals:/usr/local/emacs/info
export INFOPATH
```

Either form would cause Info to look first in the current directory, indicated by the '`.`', then in the '`~bob/manuals`' directory, and finally in the '`/usr/local/emacs/info`' directory (which is the usual location for the standard Info directory).

Appendix A @-Command List

Here is an alphabetical list of the @-commands in Texinfo.

`@*` Force a line break. Do not end a paragraph that uses `@*` with an `@refill` command. See Section 16.1 [Line Breaks], page 137.

`@.` Stands for a period that really does end a sentence. See Section 9.1.3 [Controlling Spacing], page 92.

`@:` Indicate to TEX that an immediately preceding period, question mark, exclamation mark, or colon does not end a sentence. Prevent TEX from inserting extra whitespace as it does at the end of a sentence. The command has no effect on the Info file output. See Section 9.1.3 [Controlling Spacing], page 92.

`@@` Stands for '`@`'. See Section 9.1 [Inserting '`@`'], page 91.

`@{` Stands for a left-hand brace, '`{`'. See Section 9.1 [Inserting `@` braces and periods], page 91.

`@}` Stands for a right-hand brace, '`}`'. See Section 9.1 [Inserting `@` braces and periods], page 91.

`@appendix` *title*
Begin an appendix. The title appears in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See Section 5.5 [The `@unnumbered` and `@appendix` Commands], page 52.

`@appendixsec` *title*
Begin an appendix section within an appendix. The section title appears in the table of contents of a printed manual. In Info, the title is underlined with equal signs. See Section 5.8 [Section Commands], page 53.

`@appendixsubsec` *title*
Begin an appendix subsection within an appendix. The title appears in the table of contents of a printed manual. In Info, the title is underlined with hyphens. See Section 5.10 [Subsection Commands], page 54.

`@appendixsubsubsec` *title*
Begin an appendix subsubsection within a subappendix. The title appears in the table of contents of a printed manual. In Info, the title is underlined with periods. See Section 5.11 [The 'subsub' Commands], page 55.

`@asis` Used following `@table` and `@ftable` to print the table's first column without highlighting ("as is"). See Section 12.3 [Making a Two-column Table], page 112.

`@author` *author*
Typeset *author* flushleft and underline it. See Section 3.4.3 [The `@title` and `@author` Commands], page 36.

`@b{`*text*`}` Print *text* in **bold** font. No effect in Info. See Section 8.2.3 [Fonts], page 90.

`@bullet{}`
: Generate a large round dot, or the closest possible thing to one. See Section 9.2.2 [`@bullet`], page 93.

`@bye` Terminate TEX processing on the file. TEX does not see any of the contents of the file following the `@bye` command. See Chapter 4 [Ending a File], page 45.

`@c` *comment*
: Begin a comment in Texinfo. The rest of the line does not appear in either the Info file or the printed manual. A synonym for `@comment`. See Section 1.4 [General Syntactic Conventions], page 8.

`@center` *line-of-text*
: Center the line of text following the command. See Section 3.4.2 [`@center`], page 35.

`@chapheading` *title*
: Print a chapter-like heading in the text, but not in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See Section 5.6 [`@majorheading` and `@chapheading`], page 52.

`@chapter` *title*
: Begin a chapter. The chapter title appears in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See Section 5.4 [`@chapter`], page 51.

`@cindex` *entry*
: Add *entry* to the index of concepts. See Section 15.1 [Defining the Entries of an Index], page 131.

`@cite{`*reference*`}`
: Highlight the name of a book or other reference that lacks a companion Info file. See Section 8.1.8 [`@cite`], page 88.

`@code{`*sample-code*`}`
: Highlight text that is an expression, a syntactically complete token of a program, or a program name. See Section 8.1.1 [`@code`], page 82.

`@comment` *comment*
: Begin a comment in Texinfo. The rest of the line does not appear in either the Info file or the printed manual. See Section 1.4 [General Syntactic Conventions], page 8.

`@contents`
: Print a complete table of contents. Has no effect in Info, which uses menus instead. See Section 4.2 [Generating a Table of Contents], page 46.

`@copyright{}`
: Generate a copyright symbol. See Section 9.3.2 [`@copyright`], page 94.

`@defcodeindex` *index-name*
: Define a new index and its indexing command. Print entries in an `@code` font. See Section 15.5 [Defining New Indices], page 135.

`@defcv` *category class name*

Format a description for a variable associated with a class in object-oriented programming. Takes three arguments: the category of thing being defined, the class to which it belongs, and its name. See Chapter 13 [Definition Commands], page 115.

`@deffn` *category name arguments...*

Format a description for a function, interactive command, or similar entity that may take arguments. `@deffn` takes as arguments the category of entity being described, the name of this particular entity, and its arguments, if any. See Chapter 13 [Definition Commands], page 115.

`@defindex` *index-name*

Define a new index and its indexing command. Print entries in a roman font. See Section 15.5 [Defining New Indices], page 135.

`@defivar` *class instance-variable-name*

Format a description for an instance variable in object-oriented programming. The command is equivalent to '`@defcv {Instance Variable}` ...'. See Chapter 13 [Definition Commands], page 115.

`@defmac` *macro-name arguments...*

Format a description for a macro. The command is equivalent to '`@deffn Macro` ...'. See Chapter 13 [Definition Commands], page 115.

`@defmethod` *class method-name arguments...*

Format a description for a method in object-oriented programming. The command is equivalent to '`@defop Method` ...'. Takes as arguments the name of the class of the method, the name of the method, and its arguments, if any. See Chapter 13 [Definition Commands], page 115.

`@defop` *category class name arguments...*

Format a description for an operation in object-oriented programming. `@defop` takes as arguments the overall name of the category of operation, the name of the class of the operation, the name of the operation, and its arguments, if any. See Chapter 13 [Definition Commands], page 115.

`@defopt` *option-name*

Format a description for a user option. The command is equivalent to '`@defvr {User Option}` ...'. See Chapter 13 [Definition Commands], page 115.

`@defspec` *special-form-name arguments...*

Format a description for a special form. The command is equivalent to '`@deffn {Special Form}` ...'. See Chapter 13 [Definition Commands], page 115.

`@deftp` *category name-of-type attributes...*

Format a description for a data type. `@deftp` takes as arguments the category, the name of the type (which is a word like 'int' or 'float'), and then the names of attributes of objects of that type. See Chapter 13 [Definition Commands], page 115.

`@deftypefn` *classification data-type name arguments...*

Format a description for a function or similar entity that may take arguments and that is typed. `@deftypefn` takes as arguments the classification of entity being described, the type, the name of the entity, and its arguments, if any. See Chapter 13 [Definition Commands], page 115.

`@deftypefun` *data-type function-name arguments...*
Format a description for a function in a typed language. The command is equivalent to '`@deftypefn Function` ...'. See Chapter 13 [Definition Commands], page 115.

`@deftypevr` *classification data-type name*
Format a description for something like a variable in a typed language—an entity that records a value. Takes as arguments the classification of entity being described, the type, and the name of the entity. See Chapter 13 [Definition Commands], page 115.

`@deftypevar` *data-type variable-name*
Format a description for a variable in a typed language. The command is equivalent to '`@deftypevr Variable` ...'. See Chapter 13 [Definition Commands], page 115.

`@defun` *function-name arguments...*
Format a description for functions. The command is equivalent to '`@deffn Function` ...'. See Chapter 13 [Definition Commands], page 115.

`@defvar` *variable-name*
Format a description for variables. The command is equivalent to '`@defvr Variable` ...'. See Chapter 13 [Definition Commands], page 115.

`@defvr` *category name*
Format a description for any kind of variable. `@defvr` takes as arguments the category of the entity and the name of the entity. See Chapter 13 [Definition Commands], page 115.

`@dfn{`*term*`}`
Highlight the introductory or defining use of a term. See Section 8.1.7 [`@dfn`], page 87.

`@display` Begin a kind of example. Indent text, do not fill, do not select a new font. Pair with `@end display`. See Section 10.7 [`@display`], page 100.

`@dots{}` Insert an ellipsis: '...'. See Section 9.2.1 [`@dots`], page 93.

`@emph{`*text*`}`
Highlight *text*. See Section 8.2 [Emphasizing Text], page 88.

`@enumerate`
Begin a numbered list, using `@item` for each entry. Pair with `@end enumerate`. See Section 12.2 [`@enumerate`], page 111.

`@equiv{}` Indicate the exact equivalence of two forms to the reader with a special glyph: '≡'. See Section 11.5 [Equivalence], page 105.

`@error{}` Indicate to the reader with a special glyph that the following text is an error message: 'error '. See Section 11.4 [Error Special Glyph], page 105.

`@evenfooting` [*left*] `@|` [*center*] `@|` [*right*]

Specify footings for even-numbered (left-hand) pages. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@evenheading` [*left*] `@|` [*center*] `@|` [*right*]
Specify headings for even-numbered (left-hand) pages. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@everyfooting` [*left*] `@|` [*center*] `@|` [*right*]
Specify footings for every page. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@everyheading` [*left*] `@|` [*center*] `@|` [*right*]
Specify headings for every page. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@example` Begin an example. Indent text, do not fill, select fixed-width font. Pair with `@end example`. See Section 10.3 [`@example`], page 96.

`@exdent` *line-of-text*
Remove any indentation a line might have. See Section 10.9 [Undoing the Indentation of a Line], page 101.

`@expansion{}`
Indicate the result of a macro expansion to the reader with a special glyph: '↦'. See Section 11.2 [expansion], page 103.

`@file{`*filename*`}`
Highlight the name of a file or directory. See Section 8.1.6 [`@file`], page 87.

`@finalout`
Prevent TEX from printing large black warning rectangles beside over-wide lines. See Section 18.7 [Overfull Hboxes], page 150.

`@findex` *entry*
Add *entry* to the index of functions. See Section 15.1 [Defining the Entries of an Index], page 131.

`@flushleft`
Left justify every line but leave the right end ragged. Leave font as is. Pair with `@end flushleft`. See Section 10.10 [`@flushleft` and `@flushright`], page 101.

`@flushright`
Right justify every line but leave the left end ragged. Leave font as is. Pair with `@end flushright`. See Section 10.10 [`@flushleft` and `@flushright`], page 101.

`@footnote{`*text-of-footnote*`}`
Enter a footnote. Footnote text is printed at the bottom of the page by TEX; Info may format in either 'End Node' or 'Make Node' style. See Chapter 14 [Footnotes], page 129.

`@footnotestyle` *style*
Specify an Info file's footnote style, either '`end`' for the end node style or '`separate`' for the separate node style. See Chapter 14 [Footnotes], page 129.

`@format` Begin a kind of example. Like `@example` or `@display`, but do not narrow the margins and do not select the fixed-width font. Pair with `@end format`. See Section 10.3 [`@example`], page 96.

`@ftable` *formatting-command*
Begin a two-column table, using `@item` for each entry. Automatically enter each of the items in the first column into the index of functions. Pair with `@end ftable`. The same as `@table`, except for indexing. See Section 12.3.1 [`@ftable`], page 113.

`@group` Hold text together that must appear on one printed page. Pair with `@end group`. Not relevant to Info. See Section 16.5 [`@group`], page 139.

`@heading` *title*
Print an unnumbered section-like heading in the text, but not in the table of contents of a printed manual. In Info, the title is underlined with equal signs. See Section 5.8 [Section Commands], page 53.

`@headings` *on-off-single-double*
Turn headings on or off, or specify single-sided or double-sided headings for printing. `@headings on` is synonymous with `@headings double`. See Section 3.4.6 [The `@headings` Command], page 38.

`@i{`*text*`}` Print *text* in *italic* font. No effect in Info. See Section 8.2.3 [Fonts], page 90.

`@ifinfo` Begin a stretch of text that will be ignored by TEX when it typesets the printed manual. The text appears only in the Info file. Pair with `@end ifinfo`. See Chapter 17 [Conditionally Visible Text], page 141.

`@iftex` Begin a stretch of text that will not appear in the Info file, but will be processed only by TEX. Pair with `@end iftex`. See Chapter 17 [Conditionally Visible Text], page 141.

`@ignore` Begin a stretch of text that will not appear in either the Info file or the printed output. Pair with `@end ignore`. See Section 1.5 [Comments and Ignored Text], page 9.

`@include` *filename*
Incorporate the contents of the file *filename* into the Info file or printed document. See Appendix B [Include Files], page 177.

`@inforef{`*node-name*`, [`*entry-name*`], `*info-file-name*`}`
Make a cross reference to an Info file for which there is no printed manual. See Section 7.7 [Cross references using `@inforef`], page 78.

`\input` *macro-definitions-file*
Use the specified macro definitions file. This command is used only in the first line of a Texinfo file to cause TEX to make use of the '`texinfo`' macro definitions file. The backslash in `\input` is used instead of an `@` because TEX does not properly recognize `@` until after it has read the definitions file. See Section 3.2 [The Texinfo File Header], page 28.

`@item` Indicate the beginning of a marked paragraph for `@itemize` and `@enumerate`; indicate the beginning of the text of a first column entry for `@table` and `@ftable`. See

Chapter 12 [Lists and Tables], page 109.

`@itemize` *mark-generating-character-or-command*
: Produce a sequence of indented paragraphs, with a mark inside the left margin at the beginning of each paragraph. Pair with `@end itemize`. See Section 12.1 [`@itemize`], page 110.

`@itemx`
: Like `@item` but do not generate extra vertical space above the item text. See Section 12.3.2 [`@itemx`], page 113.

`@kbd{`*keyboard-characters*`}`
: Indicate text that consists of characters of input to be typed by users. See Section 8.1.2 [`@kbd`], page 83.

`@key{`*key-name*`}`
: Highlight *key-name*, a conventional name for a key on a keyboard. See Section 8.1.3 [`@key`], page 84.

`@kindex` *entry*
: Add *entry* to the index of keys. See Section 15.1 [Defining the Entries of an Index], page 131.

`@lisp`
: Begin an example of Lisp code. Indent text, do not fill, select fixed-width font. Pair with `@end lisp`. See Section 10.5 [`@lisp`], page 99.

`@majorheading` *title*
: Print a chapter-like heading in the text, but not in the table of contents of a printed manual. Generate more vertical whitespace before the heading than the `@chapheading` command. In Info, the chapter heading line is underlined with asterisks. See Section 5.6 [`@majorheading` and `@chapheading`], page 52.

`@menu`
: Mark the beginning of a menu of nodes in Info. No effect in a printed manual. Pair with `@end menu`. See Section 6.4 [`@menu`], page 61.

`@minus{}`
: Generate a minus sign. See Section 9.4 [`@minus`], page 94.

`@need` *n*
: Start a new page in a printed manual if fewer than *n* mils (thousandths of an inch) remain on the current page. See Section 16.6 [`@need`], page 140.

`@node` *name, next, previous, up*
: Define the beginning of a new node in Info, and serve as a locator for references for TEX. See Section 6.2 [`@node`], page 58.

`@noindent`
: Prevent text from being indented as if it were a new paragraph. See Section 10.4 [`@noindent`], page 98.

`@oddfooting [`*left*`] @| [`*center*`] @| [`*right*`]`
: Specify footings for odd-numbered (right-hand) pages. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@oddheading [`*left*`] @| [`*center*`] @| [`*right*`]`

Specify headings for odd-numbered (right-hand) pages. Not relevant to Info. See Section C.3 [How to Make Your Own Headings], page 185.

`@page` Start a new page in a printed manual. No effect in Info. See Section 16.4 [`@page`], page 139.

`@paragraphindent` *indent*
Indent paragraphs by *indent* number of spaces; delete indentation if the value of *indent* is 0; and do not change indentation if *indent* is `asis`. See Section 3.2.6 [Paragraph Indenting], page 32.

`@pindex` *entry*
Add *entry* to the index of programs. See Section 15.1 [Defining the Entries of an Index], page 131.

`@point{}` Indicate the position of point in a buffer to the reader with a special glyph: '$\star$'. See Section 11.6 [Indicating Point in a Buffer], page 106.

`@print{}` Indicate printed output to the reader with a special glyph: ' $\dashv$ '. See Section 11.3 [Print Special Glyph], page 104.

`@printindex` *index-name*
Print an alphabetized two-column index in a printed manual or generate an alphabetized menu of index entries for Info. See Section 4.1 [Printing Indices & Menus], page 45.

`@pxref{`*node-name*`,` [*entry*]`,` [*topic*]`,` [*info-file*]`,` [*manual*]`}`
Make a reference that starts with a lower case 'see' in a printed manual. Use within parentheses only. Do not follow command with a punctuation mark. The Info formatting commands automatically insert terminating punctuation as needed, which is why you do not need to insert punctuation. Only the first argument is mandatory. See Section 7.6 [`@pxref`], page 77.

`@quotation`
Narrow the margins to indicate text that is quoted from another real or imaginary work. Write command on a line of its own. Pair with `@end quotation`. See Section 10.2 [`@quotation`], page 96.

`@r{`*text*`}` Print *text* in roman font. No effect in Info. See Section 8.2.3 [Fonts], page 90.

`@ref{`*node-name*`,` [*entry*]`,` [*topic*]`,` [*info-file*]`,` [*manual*]`}`
Make a reference. In a printed manual, the reference does not start with a 'See'. Follow command with a punctuation mark. Only the first argument is mandatory. See Section 7.5 [`@ref`], page 76.

`@refill` In Info, refill and indent the paragraph after all the other processing has been done. No effect on TEX, which always refills, and not needed on an Info file formatted with `makeinfo`, which also automatically refills. See Appendix E [Refilling Paragraphs], page 199.

`@result{}`

Indicate the result of an expression to the reader with a special glyph: '⇒'. See Section 11.1 [@result], page 103.

@samp{*text*}
: Highlight *text* that is a literal example of a sequence of characters. Used for single characters, for statements and often for entire shell commands. See Section 8.1.4 [@code], page 85.

@sc{*text*} Set *text* in a printed output in THE SMALL CAPS FONT and set text in the Info file in uppercase letters. See Section 8.2.2 [Smallcaps], page 89.

@section *title*
: Begin a section within a chapter. In a printed manual, the section title is numbered and appears in the table of contents. In Info, the title is underlined with equal signs. See Section 5.7 [@section], page 53.

@setchapternewpage *on-off-odd*
: Specify whether chapters start on new pages, and if so, whether on odd-numbered (right-hand) new pages. See Section 3.2.5 [@setchapternewpage], page 31.

@setfilename *info-file-name*
: Provide a name for the Info file. See Section 1.4 [General Syntactic Conventions], page 8.

@settitle *title*
: Provide a title for page headers in a printed manual. See Section 1.4 [General Syntactic Conventions], page 8.

@shortcontents
: Print a short table of contents. Not relevant to Info, which uses menus rather than tables of contents. A synonym for @summarycontents. See Section 4.2 [Generating a Table of Contents], page 46.

@smallbook
: Cause TEX to produce a printed manual in a 7 by 9.25 inch format rather than the regular 8.5 by 11 inch format. See Section 18.8 [Printing Small Books], page 150. Also, see Section 10.6 [@smallexample and @smalllisp], page 99.

@smallexample
: Indent text to indicate an example. Do not fill, select fixed-width font. In @smallbook format, print text in a smaller font than with @example. Pair with @end smallexample. See Section 10.6 [@smallexample and @smalllisp], page 99.

@smalllisp
: Begin an example of Lisp code. Indent text, do not fill, select fixed-width font. In @smallbook format, print text in a smaller font. Pair with @end smalllisp. See Section 10.6 [@smallexample and @smalllisp], page 99.

@sp *n* Skip *n* blank lines. See Section 16.3 [@sp], page 139.

@strong *text*
: Emphasize *text* by typesetting it in a **bold** font for the printed manual and by sur-

rounding it with asterisks for Info. See Section 8.2.1 [Emphasizing Text], page 88.

`@subheading` *title*

Print an unnumbered subsection-like heading in the text, but not in the table of contents of a printed manual. In Info, the title is underlined with hyphens. See Section 5.10 [`@unnumberedsubsec @appendixsubsec @subheading`], page 54.

`@subsection` *title*

Begin a subsection within a section. In a printed manual, the subsection title is numbered and appears in the table of contents. In Info, the title is underlined with hyphens. See Section 5.9 [`@subsection`], page 54.

`@subsubheading` *title*

Print an unnumbered subsubsection-like heading in the text, but not in the table of contents of a printed manual. In Info, the title is underlined with periods. See Section 5.11 [The 'subsub' Commands], page 55.

`@subsubsection` *title*

Begin a subsubsection within a subsection. In a printed manual, the subsubsection title is numbered and appears in the table of contents. In Info, the title is underlined with periods. See Section 5.11 [The 'subsub' Commands], page 55.

`@subtitle` *title*

In a printed manual, set a subtitle in a normal sized font flush to the right-hand side of the page. Not relevant to Info, which does not have title pages. See Section 3.4.3 [`@title @subtitle` and `@author` Commands], page 36.

`@summarycontents`

Print a short table of contents. Not relevant to Info, which uses menus rather than tables of contents. A synonym for `@shortcontents`. See Section 4.2 [Generating a Table of Contents], page 46.

`@syncodeindex` *from-index into-index*

Merge the index named in the first argument into the index named in the second argument, printing the entries from the first index in `@code` font. See Section 15.4 [Combining Indices], page 134.

`@synindex` *from-index into-index*

Merge the index named in the first argument into the index named in the second argument. Do not change the font of *from-index* entries. See Section 15.4 [Combining Indices], page 134.

`@t{`*text*`}` Print *text* in a `fixed-width` font. No effect in Info. See Section 8.2.3 [Fonts], page 90.

`@table` *formatting-command*

Begin a two column table, using `@item` for each entry. Write each first column entry on the same line as `@item`. First column entries are printed in the font resulting from *formatting-command*. See Section 12.3 [Making a Two-column Table], page 112. Also see Section 12.3.1 [`@ftable`], page 113, and Section 12.3.2 [`@itemx`], page 113.

`@TeX{}` Insert the logo TEX. See Section 9.3 [Inserting TEX and ©], page 94.

`@tex` Enter TEX completely. Pair with `@end tex`. See Section 17.1 [Using Ordinary TEX Commands], page 141.

`@thischapter`
In a heading or footing, stands for the number and name of the current chapter, in the format 'Chapter 1: First Chapter'. See Section C.3 [How to Make Your Own Headings], page 185.

`@thischaptername`
In a heading or footing, stands for the name of the current chapter. See Section C.3 [How to Make Your Own Headings], page 185.

`@thisfile`
In a heading or footing, stands for the name of the current `@include` file. Does not insert anything if not within an `@include` file. See Section C.3 [How to Make Your Own Headings], page 185.

`@thispage`
In a heading or footing, stands for the current page number. See Section C.3 [How to Make Your Own Headings], page 185.

`@thistitle`
In a heading or footing, stands for the name of the document, as specified by the `@settitle` command. See Section C.3 [How to Make Your Own Headings], page 185.

`@tindex` *entry*
Add *entry* to the index of data types. See Section 15.1 [Defining the Entries of an Index], page 131.

`@title` *title*
In a printed manual, set a title flush to the left-hand side of the page in a larger than normal font and underline it with a black rule. Not relevant to Info, which does not have title pages. See Section 3.4.3 [The `@title` `@subtitle` and `@author` Commands], page 36.

`@titlefont{`*text*`}`
In a printed manual, print *text* in a larger than normal font. Not relevant to Info, which does not have title pages. See Section 3.4.2 [The `@titlefont` `@center` and `@sp` Commands], page 35.

`@titlepage`
Indicate to Texinfo the beginning of the title page. Write command on a line of its own. Pair with `@end titlepage`. Nothing between `@titlepage` and `@end titlepage` appears in Info. See Section 3.4.1 [`@titlepage`], page 34.

`@today{}` Insert the current date, in '1 Jan 1900' style. See Section C.3 [How to Make Your Own Headings], page 185.

`@top` *title* In a Texinfo file to be formatted with `makeinfo`, identify the topmost node line in the file, which must be written on the line immediately preceding the `@top` command. Used for `makeinfo`'s node pointer insertion feature. The title is underlined with asterisks.

Both the node line and the `@top` line normally should be enclosed by `@ifinfo` and `@end ifinfo`. In TEX and `texinfo-format-buffer`, the `@top` command is merely a synonym for `@unnumbered`. See Section 6.3 [Creating Pointers with `makeinfo`], page 60.

`@unnumbered` *title*

In a printed manual, begin a chapter that appears without chapter numbers of any kind. The title appears in the table of contents of a printed manual. In Info, the title is underlined with asterisks. See Section 5.5 [`@unnumbered` and `@appendix`], page 52.

`@unnumberedsec` *title*

In a printed manual, begin a section that appears without section numbers of any kind. The title appears in the table of contents of a printed manual. In Info, the title is underlined with equal signs. See Section 5.8 [Section Commands], page 53.

`@unnumberedsubsec` *title*

In a printed manual, begin an unnumbered subsection within a chapter. The title appears in the table of contents of a printed manual. In Info, the title is underlined with hyphens. See Section 5.10 [`@unnumberedsubsec` `@appendixsubsec` `@subheading`], page 54.

`@unnumberedsubsubsec` *title*

In a printed manual, begin an unnumbered subsubsection within a chapter. The title appears in the table of contents of a printed manual. In Info, the title is underlined with periods. See Section 5.11 [The 'subsub' Commands], page 55.

`@var{`*metasyntactic-variable*`}`

Highlight a metasyntactic variable, which is something that stands for another piece of text. Thus, in this entry, the word *metasyntactic-variable* is highlighted with `@var`. See Section 8.1.5 [Indicating Metasyntactic Variables], page 86.

`@vindex` *entry*

Add *entry* to the index of variables. See Section 15.1 [Defining the Entries of an Index], page 131.

`@vskip` *amount*

in a printed manual, insert whitespace so as to push text on the remainder of the page towards the bottom of the page. Used in formatting the copyright page with the argument '`0pt plus 1filll`'. (Note spelling of '`filll`'.) `@vskip` may be used only in contexts ignored for Info. See Section 3.4.4 [The Copyright Page and Printed Permissions], page 37.

`@w{`*text*`}` Prevent *text* from being split across two lines. Do not end a paragraph that uses `@w` with an `@refill` command. In the Texinfo file, keep *text* on one line. See Section 16.2 [`@w`], page 138.

`@xref{`*node-name*`,` [*entry*]`,` [*topic*]`,` [*info-file*]`,` [*manual*]`}`

Make a reference that starts with 'See' in a printed manual. Follow command with a punctuation mark. Only the first argument is mandatory. See Section 7.3 [`@xref`], page 69.

Appendix B Include Files

When TEX or an Info formatting command sees an `@include` command in a Texinfo file, it processes the contents of the file named by the command and incorporates them into the DVI or Info file being created. Index entries from the included file are incorporated into the indices of the output file.

B.1 How to Use Include Files

To include another file within a Texinfo file, write the `@include` command at the beginning of a line and follow it on the same line by the name of a file to be included. For example:

```
@include chap47.texinfo
```

An included file should simply be a segment of text that you expect to be included as-is into the overall or *outer* Texinfo file; it should not contain the standard beginning and end parts of a Texinfo file. In particular, you should not start an included file with a line saying '`\input texinfo`'; if you do, that phrase is inserted into the output file as is. Likewise, you should not end an included file with an `@bye` command; that command will stop TEX processing immediately.

In the past, you did need to write an `@setfilename` line at the beginning of an included file, but no longer; now, it does not matter whether you write such a line. If an `@setfilename` line exists in an included file, it is ignored.

Conventionally, an included file begins with an `@node` line that is followed by an `@chapter` line. Each included file is one chapter. This makes it easy to use the regular node and menu creating and updating commands to create the node pointers and menus within the included file. However, the simple Emacs node and menu creating and updating commands do not work with multiple Texinfo files. Thus you cannot use these commands to fill in the 'Next', 'Previous', and 'Up' pointers of the `@node` line that begins the included file. Also, you cannot use the regular commands to create a master menu for the whole file. Either you must insert the menus and the 'Next', 'Previous', and 'Up' pointers by hand, or you must use the `texinfo-multiple-files-update` command that is designed for `@include` files.

B.2 `texinfo-multiple-files-update`

The `texinfo-multiple-files-update` command creates or updates 'Next', 'Previous', and 'Up' pointers of included files as well as those in the outer or over all Texinfo file, and it creates or updates a main menu in the outer file. Depending whether you call it with optional arguments, it updates only the pointers in the first `@node` line of the included files or all of them.

`M-x texinfo-multiple-files-update`
: Called without any arguments:
  - Create or update the 'Next', 'Previous', and 'Up' pointers of the first `@node` line in each file included in an outer or overall Texinfo file.
  - Create or update the 'Top' level node pointers of the outer or overall file.
  - Create or update a main menu in the outer file.

`C-u M-x texinfo-multiple-files-update`
: Called with a prefix argument (a non-`nil` *make-master-menu* argument and a non-`nil` *make-master-menu* argument, if called from a program), create and insert a master menu in the outer file in addition to creating or updating pointers in the first `@node` line in each included file and creating or updating the 'Top' level node pointers of the outer file. The master menu is made from all the menus in all the included files.

`C-u 8 M-x texinfo-multiple-files-update`
: Called with a numeric prefix argument (a non-`nil` *update-everything* argument, if called from a program):
  - Create or update *all* the 'Next', 'Previous', and 'Up' pointers of all the included files.
  - Create or update *all* the menus of all the included files.
  - And then create a master menu in the outer file. This is similar to invoking `texinfo-master-menu` with an argument when you are working with just one file.

Note the use of the prefix argument in interactive use: with a regular prefix argument, just `C-u`, the `texinfo-multiple-files-update` command inserts a master menu; with a numeric prefix argument, such as `C-u 8`, the command updates every pointer and menu in all the files and then inserts a master menu.

B.3 Sample File with `@include`

If you plan to use the `texinfo-multiple-files-update` command, the outer Texinfo file that lists included files within it should contain nothing but the beginning and end parts of a Texinfo

file, and a number of `@include` commands listing the included files. It should not even include indices, which should be listed in an included file of their own.

Moreover, each of the included files must contain exactly one highest level node (conventionally, an `@chapter` node or equivalent), and this node must be the first node in the included file. Furthermore, each of these highest level nodes in each included file must be at the same hierarchical level in the file structure. Usually, each is an `@chapter`, an `@appendix`, or an `@unnumbered` node. Thus, normally, each included file contains one, and only one, chapter or equivalent-level node.

The outer file should *not* contain any nodes besides the single 'Top' node. The `texinfo-multiple-files-update` command will not process them.

Here is an example of an outer Texinfo file with `@include` files within it before running `texinfo-multiple-files-update`, which would insert a main or master menu:

```
\input texinfo @c -*-texinfo-*-
@setfilename  include-example.info
@settitle Include Example

@setchapternewpage odd
@titlepage
@sp 12
@center @titlefont{Include Example}
@sp 2
@center by Whom Ever

@page
Copyright @copyright{} 1990 Free Software Foundation, Inc.
@end titlepage

@node Top, First, (dir), (dir)
@ifinfo
@chapter Master Menu
@end ifinfo

@include foo.texinfo
@include bar.texinfo
@include index.texinfo

@summarycontents
@contents

@bye
```

An included file, such as '`foo.texinfo`', might look like this:

```
@node First, Second, , Top
@chapter First Chapter

Contents of first chapter ...
```

The full contents of '`index.texinfo`' might be as simple as this:

```
@unnumbered Concept Index, , Second, Top
@printindex cp
```

The outer Texinfo source file for the *GNU Emacs Lisp Reference Manual* is named '`elisp.texi`'. This outer file contains a master menu with 417 items in it and a list of 41 `@include` files.

B.4 Evolution of Include Files

When Info was first created, it was customary to create many small Info files on one subject. Each Info file was formatted from its own Texinfo source file. This custom meant that Emacs did not need to make a large buffer to hold the whole of a large Info file when someone wanted information; instead, Emacs allocated just enough memory for the small Info file that contained the particular information sought. This way, Emacs could avoid wasting memory.

References from one file to another were made by referring to the file name as well as the node name. (See Section 6.5 [Referring to Other Info Files], page 64. Also, see Section 7.3.4 [`@xref` with Four and Five Arguments], page 73.)

Include files were designed primarily as a way to create a single, large printed manual out of several smaller Info files. In a printed manual, all the references were within the same document, so TEX could automatically determine the references' page numbers. The Info formatting commands used include files only for creating joint indices; each of the individual Texinfo files had to be formatted for Info individually. (Each, therefore, required its own `@setfilename` line.)

However, because large Info files are now split automatically, it is no longer necessary to keep them small.

Nowadays, multiple Texinfo files are used mostly for large documents, such as the *GNU Emacs Lisp Reference Manual*, and for projects in which several different people write different sections of a document simultaneously.

In addition, the Info formatting commands have been extended to work with the `@include` command so as to create a single large Info file that is split if necessary into smaller files. This

means that you can write menus and cross references without naming the different Texinfo files.

Appendix C Page Headings

Most printed manuals contain headings along the top of every page except the title and copyright pages. Some manuals also contain footings. (Headings and footings have no meaning to Info, which is not paginated.)

Texinfo provides standard heading formats for manuals that are printed on one side of each sheet of paper and for manuals that are printed on both sides of the paper. Usually, you will use one or other of these formats, but you can specify your own format, if you wish.

In addition, you can specify whether chapters should begin on a new page, or merely continue the same page as the previous chapter; and if chapters begin on new pages, you can specify whether they must be odd-numbered pages.

By convention, a book is printed on both sides of each sheet of paper. When you open a book flat, the right-hand page is odd-numbered, and chapters begin on right-hand pages—a preceding left-hand page is left blank if necessary. Reports, however, are often printed on just one side of paper, and chapters begin on a fresh page immediately following the end of the preceding chapter. In short or informal reports, chapters often do not begin on a new page at all, but are separated from the preceding text by a small amount of whitespace.

The `@setchapternewpage` command controls whether chapters begin on new pages, and whether one of the standard heading formats is used. In addition, Texinfo has several heading and footing commands that you can use to generate your own heading and footing formats.

In Texinfo, headings and footings are single lines at the tops and bottoms of pages; you cannot create multiline headings or footings. Each header or footer line is divided into three parts: a left part, a middle part, and a right part. Any part, or a whole line, may be left blank. Text for the left part of a header or footer line is set flushleft; text for the middle part is centered; and, text for the right part is set flushright.

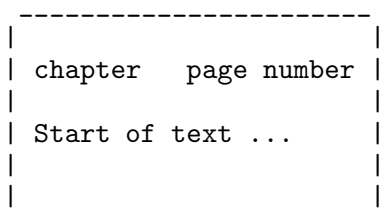
C.1 Standard Heading Formats

Texinfo provides two standard heading formats, one for manuals printed on one side of each sheet of paper, and the other for manuals printed on both sides of the paper.

By default, nothing is specified for the footing of a Texinfo file, so the footings remains blank.

The standard format for single-sided printing consists of a header line in which the left-hand part contains the name of the chapter, the central part is blank, and the right-hand part contains the page number.

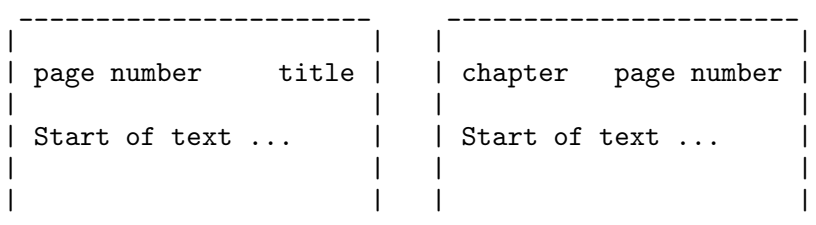
The single-sided page looks like this:



The standard format for two-sided printing depends on whether the page number is even or odd. By convention, even-numbered pages are on the left- and odd-numbered pages are on the right. (TEX will adjust the widths of the left- and right-hand margins. Usually, widths are correct, but during double-sided printing, it is wise to check that pages will bind properly—sometimes a printer will produce output in which the even-numbered pages have a larger right-hand margin than the odd-numbered pages.)

In the standard double-sided format, the left part of the left-hand (even-numbered) page contains the page number, the central part is blank, and the right part contains the title (specified by the `@settitle` command). The left part of the right-hand (odd-numbered) page contains the name of the chapter, the central part is blank, and the right part contains the page number.

The two pages, side by side as in an open book, look like this:



The chapter name is preceded by the word '`Chapter`', the chapter number and a colon. This makes it easier to keep track of where you are in the manual.

C.2 Specifying the Type of Heading

TEX does not begin to generate page headings for a standard Texinfo file until it reaches the `@end titlepage` command. Thus, the title and copyright pages are not numbered. The `@end titlepage` command causes TEX to begin to generate page headings according to a standard format specified by the `@setchapternewpage` command that precedes the `@titlepage` section.

There are four possibilities:

No `@setchapternewpage` command
: Cause TEX to specify the single-sided heading format, with chapters on new pages. This is the same as `@setchapternewpage on`.

`@setchapternewpage on`
: Specify the single-sided heading format, with chapters on new pages.

`@setchapternewpage off`
: Cause TEX to start new chapters on the same page as the the last page of the preceding chapter, after skipping some vertical whitespace. It also causes TEX to typeset for double-sided printing.

`@setchapternewpage odd`
: Specify the double-sided heading format, with chapters on new pages.

Texinfo lacks an `@setchapternewpage even` command.

C.3 How to Make Your Own Headings

You can use the standard headings provided with Texinfo or specify your own.

Texinfo provides six commands for specifying headings and footings. The `@everyheading` and `@everyfooting` commands generate headers and footers that are the same for both even- and odd-numbered pages. The `@evenheading` and `@evenfooting` commands generate headers and footers for even-numbered (left-hand) pages; and the `@oddheading` and `@oddfooting` commands generate headers and footers for odd-numbered (right-hand) pages.

Write custom heading specifications in the Texinfo file immediately after the `@end titlepage` command. Enclose your specifications between `@iftex` and `@end iftex` commands since the `texinfo-format-buffer` command may not recognize them. Also, you must cancel the predefined heading commands with the `@headings off` command before defining your own specifications.

Here is how to tell TEX to place the chapter name at the left, the page number in the center, and the date at the right of every header for both even- and odd-numbered pages:

```
@iftex
@headings off
@everyheading @thischapter @| @thispage @| @today{}
@end iftex
```

You need to divide the left part from the central part and the central part from the right had part by inserting '`@|`' between parts. Otherwise, the specification command will not be able to tell where the text for one part ends and the next part begins.

Each part can contain text or '`@this`...' commands. The text is printed as if the part were within an ordinary paragraph in the body of the page. The '`@this`...' commands replace themselves with the page number, date, chapter name, or whatever.

Here are the six heading and footing commands:

`@everyheading` *left* `@|` *center* `@|` *right*
`@everyfooting` *left* `@|` *center* `@|` *right*

The 'every' commands specify the format for both even- and odd-numbered pages. These commands are for documents that are printed on one side of each sheet of paper, or for documents in which you want symmetrical headers or footers.

`@evenheading` *left* `@|` *center* `@|` *right*
`@oddheading` *left* `@|` *center* `@|` *right*
`@evenfooting` *left* `@|` *center* `@|` *right*
`@oddfooting` *left* `@|` *center* `@|` *right*

The 'even' and 'odd' commands specify the format for even-numbered pages and odd-numbered pages. These commands are for books and manuals that are printed on both sides of each sheet of paper.

Use the '`@this`...' commands to provide the names of chapters and sections and the page number. You can use the '`@this`...' commands in the left, center, or right portions of headers and footers, or anywhere else in a Texinfo file so long as they are between `@iftex` and `@end iftex` commands.

Here are the '`@this`...' commands:

`@thispage`

Expands to the current page number.

`@thischaptername`
: Expands to the name of the current chapter.

`@thischapter`
: Expands to the number and name of the current chapter, in the format 'Chapter 1: First Chapter'.

`@thistitle`
: Expands to the name of the document, as specified by the `@settitle` command.

`@thisfile`
: For `@include` files only: expands to the name of the current `@include` file. If the current Texinfo source file is not an `@include` file, this command has no effect. This command does *not* provide the name of the current Texinfo source file unless it is an `@include` file. (See Appendix B [Include Files], page 177, for more information about `@include` files.)

You can also use the `@today{}` command, which expands to the current date, in '1 Jan 1900' format.

Other `@`-commands and text are printed in a header or footer just as if they were in the body of a page. It is useful to incorporate text, particularly when you are writing drafts:

```
@iftex
@headings off
@everyheading @emph{Draft!} @| @thispage @| @thischapter
@everyfooting @| @| Version: 0.27: @today{}
@end iftex
```

Beware of overlong titles: they may overlap another part of the header or footer and blot it out.

Appendix D Formatting Mistakes

Besides mistakes in the content of your documentation, there are two kinds of mistake you can make with Texinfo: you can make mistakes with `@`-commands, and you can make mistakes with the structure of the nodes and chapters.

Emacs has two tools for catching the `@`-command mistakes and two for catching structuring mistakes.

For finding problems with `@`-commands, you can run TEX or a region formatting command on the region that has a problem; indeed, you can run these commands on each region as you write it.

For finding problems with the structure of nodes and chapters, you can use `C-c C-s` (`texinfo-show-structure`) (and the related `occur` command) and you can use the `M-x Info-validate` command.

The `makeinfo` program does an excellent job of catching errors and reporting them—far better than `texinfo-format-region` or `texinfo-format-buffer`. In addition, the various functions for automatically creating and updating node pointers and menus remove many opportunities for human error.

If you can, use the updating commands to create and insert pointers and menus. These prevent many errors. Then use `makeinfo` (or its Texinfo mode manifestations, `makeinfo-region` and `makeinfo-buffer`) to format your file and check for other errors. This is the best way to work with Texinfo. But if you cannot use `makeinfo`, or your problem is very puzzling, then you may want to use the tools described in this appendix.

D.1 Catching Errors with Info Formatting

After you have written part of a Texinfo file, you can use the `texinfo-format-region` or the `makeinfo-region` command to see whether the region formats properly.

Most likely, however, you are reading this section because for some reason you cannot use the `makeinfo-region` command; therefore, the rest of this section presumes that you are using `texinfo-format-region`.

If you have made a mistake with an `@`-command, `texinfo-format-region` will stop processing

at or after the error and display an error message. To see where in the buffer the error occurred, switch to the '`*Info Region*`' buffer; the cursor will be in a position that is after the location of the error. Also, the text will not be formatted after the place where the error occurred (or more precisely, where it was detected).

For example, if you accidentally end a menu with the command `@end menus` with an 's' on the end, instead of with `@end menu`, you will see an error message that says:

```
@end menus is not handled by texinfo
```

The cursor will stop at the point in the buffer where the error occurs, or not long after it. The buffer will look like this:

```
---------- Buffer: *Info Region* ----------
* Menu:

* Using texinfo-show-structure::  How to use
                                  'texinfo-show-structure'
                                  to catch mistakes.
* Running Info-Validate::         How to check for
                                  unreferenced nodes.
@end menus
*
---------- Buffer: *Info Region* ----------
```

The `texinfo-format-region` command sometimes provides slightly odd error messages. For example,

```
(@xref{Catching Mistakes, for more info.)
```

In this case, `texinfo-format-region` detects the missing closing brace but displays a message that says '`Unbalanced parentheses`' rather than '`Unbalanced braces`'. This is because the formatting command looks for mismatches between braces as if they were parentheses.

Sometimes `texinfo-format-region` fails to detect mistakes. For example, in the following, the closing brace is swapped with the closing parenthesis:

```
(@xref{Catching Mistakes), for more info.}
```

Formatting produces:

```
(*Note for more info.: Catching Mistakes)
```

The only way for you to detect this error is to realize that the reference should have looked like this:

```
(*Note Catching Mistakes::, for more info.)
```

Incidently, if you are reading this node in Info and type `f RET` (`Info-follow-reference`), you will generate an error message that says:

```
No such node: "Catching Mistakes) The only way ...
```

This is because Info perceives the example of the error as the first cross reference in this node and if you type a `RET` immediately after typing the Info `f` command, Info will attempt to go to the referenced node. If you type `f catch TAB RET`, Info will complete the node name of the correctly written example and take you to the 'Catching Mistakes' node. (If you try this, you can return from the 'Catching Mistakes' node by typing `l` (`Info-last`).)

D.2 Catching Errors with TEX Formatting

You can also catch mistakes when you format a file with TEX.

Usually, you will want to do this after you have run `texinfo-format-buffer` (or, better, `makeinfo-buffer`) on the same file, because `texinfo-format-buffer` sometimes displays error messages that make more sense than TEX. (See Section D.1 [Debugging with Info], page 189, for more information.)

For example, TEX was run on a Texinfo file, part of which is shown here:

```
---------- Buffer: texinfo.texi ----------
name of the texinfo file as an extension.  The
@samp{??} are 'wildcards' that cause the shell to
substitute all the raw index files.  (@xref{sorting
indices, for more information about sorting
indices.)@refill
---------- Buffer: texinfo.texi ----------
```

(The cross reference lacks a closing brace.) TEX produced the following output, after which it

stopped:

```
---------- Buffer: *texinfo-tex-shell* ----------
Runaway argument?
{sorting indices, for more information about sorting
indices.) @refill @ETC.
! Paragraph ended before \xref was complete.
<to be read again>
                   \par
l.27

?
---------- Buffer: *texinfo-tex-shell* ----------
```

In this case, TEX produced an accurate and understandable error message:

```
Paragraph ended before \xref was complete.
```

(Note, however, that TEX translated the '`@`' into a '\'.) Also, '`\par`' is an internal TEX command of no relevance to Texinfo.)

Unfortunately, TEX is not always so helpful, and sometimes you have to be truly a Sherlock Holmes to discover what went wrong.

In any case, if you run into a problem like this, you can do one of two things.

1. You can tell TEX to continue running and to ignore errors as best it can by typing `r RET` at the '`?`' prompt.

   This is often the best thing to do. However, beware: the one error may produce a cascade of additional error messages as its consequences are felt through the rest of the file. (To stop TEX when it is producing such an avalanche of error messages, type `C-d` (or `C-c C-d`, if you running a shell inside of Emacs.))
2. You can tell TEX to stop this run by typing `x RET` at the '`?`' prompt.

Sometimes TEX will format a file without producing error messages even though there is a problem. This usually occurs if a command is not ended but TEX is able to continue processing anyhow. For example, if you fail to end an itemized list with the `@end itemize` command, TEX will write a DVI file that you can print out. The only error message that TEX will give you is the somewhat mysterious comment that

```
(\end occurred inside a group at level 1)
```

However, if you print the DVI file, you will find that the text of the file that follows the itemized list is entirely indented as if it were part of the last item in the itemized list. The error message is the way TEX says that it expected to find an `@end` command somewhere in the file; but that it could not determine where it was needed.

Another source of notoriously hard-to-find errors is a missing `@end group` command. If you ever are stumped by incomprehensible errors, look for a missing `@end group` command first.

If you do not have the header lines in the file, TEX may stop in the beginning of its run and display output that looks like the following. The '`*`' indicates that TEX is waiting for input.

```
This is TeX, Version 2.0 for Berkeley UNIX
(preloaded format=plain-cm 87.10.25)
(test.texinfo [1])
*
```

In this case, simply type `\end RET` after the asterisk. Then put the header lines into the Texinfo file and run the TEX command again.

D.3 Using `texinfo-show-structure`

It is not always easy to keep track of the nodes, chapters, sections, and subsections of a Texinfo file. This is especially true if you are revising or adding to a Texinfo file that someone else has written.

In GNU Emacs, in Texinfo mode, the `texinfo-show-structure` command lists all the lines that begin with the `@`-commands that specify the structure: `@chapter`, `@section`, `@appendix`, and so on. With an argument (prefix, if interactive), the command also shows the `@node` lines. The `texinfo-show-structure` command is bound to `C-c C-s` in Texinfo mode, by default.

The lines are displayed in a buffer called the '`*Occur*`' buffer. For example, when `texinfo-show-structure` was run on an earlier version of this appendix, it produced the following:

```
Lines matching "^@\\(chapter \\|sect\\|sub\\|unnum\\|major\\|
heading \\|appendix\\)" in buffer texinfo.texi.
  4:@appendix Formatting Mistakes
```

```
 52:@appendixsec Catching Errors with Info Formatting
222:@appendixsec Catching Errors with @TeX{} Formatting
338:@appendixsec Using @code{texinfo-show-structure}
407:@appendixsubsec Using @code{occur}
444:@appendixsec Finding Badly Referenced Nodes
513:@appendixsubsec Running @code{Info-validate}
573:@appendixsubsec Splitting a File Manually
```

This says that lines 4, 52, and 222 of '`texinfo.texi`' begin with the `@appendix`, `@appendixsec`, and `@appendixsec` commands respectively. If you move your cursor into the '`*Occur*`' window, you can position the cursor over one of the lines and use the `C-c C-c` command (`occur-mode-goto-occurrence`), to jump to the corresponding spot in the Texinfo file. See section "Using Occur" in *The GNU Emacs Manual*, for more information about `occur-mode-goto-occurrence`.

The first line in the '`*Occur*`' window describes the *regular expression* specified by *texinfo-heading-pattern*. This regular expression is the pattern that `texinfo-show-structure` looks for. See section "Using Regular Expressions" in *The GNU Emacs Manual*, for more information.

When you invoke the `texinfo-show-structure` command, Emacs will display the structure of the whole buffer. If you want to see the structure of just a part of the buffer, of one chapter, for example, use the `C-x n` (`narrow-to-region`) command to mark the region. (See section "Narrowing" in *The GNU Emacs Manual*.) This is how the example used above was generated. (To see the whole buffer again, use `C-x w` (`widen`).)

If you call `texinfo-show-structure` with a prefix argument by typing `C-u C-c C-s`, it will list lines beginning with `@node` as well as the lines beginning with the `@`-sign commands for `@chapter`, `@section`, and the like.

You can remind yourself of the structure of a Texinfo file by looking at the list in the '`*Occur*`' window; and if you have mis-named a node or left out a section, you can correct the mistake.

D.4 Using `occur`

Sometimes the `texinfo-show-structure` command produces too much information. Perhaps you want to remind yourself of the overall structure of a Texinfo file, and are overwhelmed by the detailed list produced by `texinfo-show-structure`. In this case, you can use the `occur` command directly. To do this, type

```
M-x occur
```

and then, when prompted, type a *regexp*, a regular expression for the pattern you want to match. (See section "Regular Expressions" in *The GNU Emacs Manual*.) The `occur` command works from the current location of the cursor in the buffer to the end of the buffer. If you want to run `occur` on the whole buffer, place the cursor at the beginning of the buffer.

For example, to see all the lines that contain the word '`@chapter`' in them, just type '`@chapter`'. This will produce a list of the chapters. It will also list all the sentences with '`@chapter`' in the middle of the line.

If you want to see only those lines that start with the word '`@chapter`', type '`^@chapter`' when prompted by `occur`. If you want to see all the lines that end with a word or phrase, end the last word with a '`$`'; for example, '`catching mistakes$`'. This can be helpful when you want to see all the nodes that are part of the same chapter or section and therefore have the same 'Up' pointer.

See section "Using Occur" in *The GNU Emacs Manual*, for more information.

D.5 Finding Badly Referenced Nodes

You can use the `Info-validate` command to check whether any of the 'Next', 'Previous', 'Up' or other node pointers fail to point to a node. This command checks that every node pointer points to an existing node. The `Info-validate` command works only on Info files, not on Texinfo files.

The `makeinfo` program validates pointers automatically, so you do not need to use the `Info-validate` command if you are using `makeinfo`. You only may need to use `Info-validate` if you are unable to run `makeinfo` and instead must create an Info file using `texinfo-format-region` or `texinfo-format-buffer`, or if you write an Info file from scratch.

D.5.1 Running `Info-validate`

To use `Info-validate`, visit the Info file you wish to check and type:

```
M-x Info-validate
```

(Note that the `Info-validate` command requires an upper case 'I'. You may also need to create a tag table before running `Info-validate`. See Section D.5.3 [Tagifying], page 197.)

If your file is valid, you will receive a message that says "File appears valid". However, if you have a pointer that does not point to a node, error messages will be displayed in a buffer called '`*problems in info file*`'.

For example, `Info-validate` was run on a test file that contained only the first node of this manual. One of the messages said:

```
In node "Overview", invalid Next: Texinfo Mode
```

This meant that the node called '`Overview`' had a 'Next' pointer that did not point to anything (which was true in this case, since the test file had only one node in it).

Now suppose we add a node named '`Texinfo Mode`' to our test case but we don't specify a 'Previous' for this node. Then we will get the following error message:

```
In node "Texinfo Mode", should have Previous: Overview
```

This is because every 'Next' pointer should be matched by a 'Previous' (in the node where the 'Next' points) which points back.

`Info-validate` also checks that all menu items and cross references point to actual nodes.

Note that `Info-validate` requires a tag table and does not work with files that have been split. (The `texinfo-format-buffer` command automatically splits files larger than 100,000 bytes.) In order to use `Info-validate` on a large file, you must run `texinfo-format-buffer` with an argument so that it does not split the Info file; and you must create a tag table for the unsplit file.

D.5.2 Creating an Unsplit File

You can run `Info-validate` only on a single Info file that has a tag table. The command will not work on the indirect subfiles that are generated when a master file is split. If you have a large file (longer than 70,000 bytes or so), you need to run the `texinfo-format-buffer` or `makeinfo-buffer` command in such a way that it does not create indirect subfiles. You will also need to create a tag table for the Info file. After you have done this, you can run `Info-validate` and look for badly referenced nodes.

The first step is to create an unsplit Info file. To prevent `texinfo-format-buffer` from splitting

a Texinfo file into smaller Info files, give a prefix to the `M-x texinfo-format-buffer` command:

```
C-u  M-x texinfo-format-buffer
```

or else

```
C-u C-c C-e C-b
```

When you do this, Texinfo will not split the file and will not create a tag table for it.

D.5.3 Tagifying a File

After creating an unsplit Info file, you must create a tag table for it. Visit the Info file you wish to tagify and type:

```
M-x Info-tagify
```

(Note the upper case `I` in `Info-tagify`.) This creates an Info file with a tag table that you can validate.

The third step is to validate the Info file:

```
M-x Info-validate
```

(Note the upper case `I` in `Info-validate`.) In brief, the steps are:

```
C-u M-x texinfo-format-buffer
M-x Info-tagify
M-x Info-validate
```

After you have validated the node structure, you can rerun `texinfo-format-buffer` in the normal way so it will construct a tag table and split the file automatically, or you can make the tag table and split the file manually.

D.5.4 Splitting a File Manually

You should split a large file or else let the `texinfo-format-buffer` or `makeinfo-buffer` command do it for you automatically. (Generally you will let one of the formatting commands do this job for you. See Chapter 19 [Creating an Info File], page 153.)

The split off files are called the indirect subfiles.

Info files are split to save memory. With smaller files, Emacs does not have make such a large buffer to hold the information.

If an Info file has more than 30 nodes, you should also make a tag table for it. See Section D.5.1 [Using Info-validate], page 195, for information about creating a tag table. (Again, tag tables are usually created automatically by the formatting command; you only need to create a tag table yourself if you are doing the job manually. Most likely, you will do this for a large, unsplit file on which you have run `Info-validate`.)

Visit the file you wish to tagify and split and type the two commands:

```
M-x Info-tagify
M-x Info-split
```

(Note that the '`I`' in '`Info`' is upper case.)

When you use the `Info-split` command, the buffer is modified into a (small) Info file which lists the indirect subfiles. This file should be saved in place of the original visited file. The indirect subfiles are written in the same directory the original file is in, with names generated by appending '`-`' and a number to the original file name.

The primary file still functions as an Info file, but it contains just the tag table and a directory of subfiles.

Appendix E Refilling Paragraphs

The `@refill` command refills and, optionally, indents the first line of a paragraph.[1]

If a paragraph contains long `@`-constructs, the paragraph may look badly filled after being formatted by `texinfo-format-region` or `texinfo-format-buffer`. this is because both `texinfo-format-region` and `texinfo-format-buffer` remove `@`-commands from formatted text but do not refill paragraphs automatically although both TEX and `makeinfo` do. Consequently, some lines become shorter than they were.

To cause the `texinfo-format-region` and `texinfo-format-buffer` commands to refill a paragraph, write `@refill` at the end of the paragraph. This command refills a paragraph in the Info file after all the other processing has been done. `@refill` has no effect on TEX or `makeinfo`, which always fill every paragraph that ought to be filled.

For example, without any indenting, the following

```
To use @code{foo}, pass @samp{xx%$} and
@var{flag} and type @kbd{x} after running
@code{make-foo}.@refill
```

produces (in the Info file)

```
To use ‘foo’, pass ‘xx%$’ and FLAG and type ‘x’ after
running ‘make-foo’.
```

whereas without the `@refill` it would produce

```
To use ‘foo’, pass ‘xx%$’ and
FLAG and type ‘x’ after running
‘make-foo’.
```

with the line broken at the same place as in the Texinfo input file.

---

[1] Perhaps the command should have been called the `@refillandindent` command, but `@refill` is shorter and the name was chosen before indenting was possible.

Write the `@refill` command at the end of the paragraph. Do not put a space before `@refill`; otherwise the command might be put at the beginning of the line when you refill the paragraph in the Texinfo file with `M-q` (`fill-paragraph`). If this were to happen, the `@refill` command might fail to work.

Do not put braces after `@refill`. The `@refill` command is the only 'within-paragraph' `@`-command that does not take braces. Because an `@refill` command is placed at the end of a paragraph and never at the beginning of a line, the braces are not necessary.

As an exception to the general rule, you can write an `@refill` command at the end of a footnote before the footnote's closing brace, even if the footnote text is embedded in a the middle of a paragraph in the Texinfo file. This is because the footnote text is extracted from the surrounding text and formatted on its own.

Also, do not end a paragraph that uses either `@*` or `@w` with an `@refill` command; otherwise, `texinfo-format-buffer` or `texinfo-format-buffer` will refill the paragraph in spite of those commands.

In addition to refilling, the `@refill` command may insert spaces at the beginning of the first line of the paragraph, thereby indenting that line. The argument to the `@paragraphindent` command specifies the amount of indentation: if the value of the argument is 0, an `@refill` command deletes existing indentation. If the value of the argument is greater than 0, an `@refill` command indents the paragraph by that number of spaces. And if the value of the argument is '`asis`', an `@refill` command does not change existing indentation. For more information about the `@paragraphindent` command, Section 3.2.6 [Paragraph Indenting], page 32.

The `@refill` command does not indent entries in a list, table, or definition, nor does `@refill` indent paragraphs preceded by `@noindent`.

Appendix F @-Command Syntax

The character '@' is used to start special Texinfo commands. (It has the same meaning that '\' has in plain TEX.) Texinfo has four types of @-command:

1. Non-alphabetic commands.
   These commands consist of an @ followed by a punctuation mark or other character that is not part of the alphabet. Non-alphabetic commands are almost always part of the text within a paragraph, and never take any argument. The two characters (@ and the other one) are complete in themselves; none is followed by braces. The non-alphabetic commands are: @., @:, @*, @@, @{, and @}.
2. Alphabetic commands that do not require arguments.
   These commands start with @ followed by a word followed by left- and right-hand braces. These commands insert special symbols in the document; they do not require arguments. For example, @dots{} ⇒ '...', @equiv{} ⇒ '≡', @TeX{} ⇒ 'TEX', and @bullet{} ⇒ '•'.
3. Alphabetic commands that do require arguments.
   These commands start with @ followed by a letter or a word, followed by an argument within braces. For example, the command @dfn indicates the introductory or defining use of a term; it is used as follows: '`In Texinfo, @@-commands are @dfn{mark-up} commands.`'
4. Alphabetic commands that occupy an entire line.
   These commands occupy an entire line. The line starts with @, followed by the name of the command (a word) such as @center or @cindex. If no argument is needed, the word is followed by the end of the line. If there is an argument, it is separated from the command name by a space. Braces are not used.

Thus, the alphabetic commands fall into three classes that have different argument syntax. You cannot tell to which class a command belongs by the appearance of its name, but you can tell by the command's meaning: if the command stands for a special glyph, it is in class 2 and does not require an argument; if it makes sense to use the command together with other text as part of a paragraph, the command is in class 3 and must be followed by an argument in braces; otherwise, it is in class 4 and uses the rest of the line as its argument.

The purpose of having a different syntax for commands of classes 3 and 4 is to make Texinfo files easier to read, and also to help the GNU Emacs paragraph and filling commands work properly. There is only one exception to this rule: the command @refill, which is always used at the end of a paragraph immediately following the final period or other punctuation character. @refill takes

no argument and does *not* require braces. `@refill` never confuses the Emacs paragraph commands because it cannot appear at the beginning of a line.

Appendix G Second Edition Features

The second edition of the Texinfo manual describes more than 20 new Texinfo mode commands and more than 50 previously undocumented Texinfo @-commands. This edition is more than twice the length of the first edition.

Here is a brief description of the new commands.

G.1 New Texinfo Mode Commands

Texinfo mode provides commands and features especially designed for working with Texinfo files. More than 20 new commands have been added, including commands for automatically creating and updating nodes and menus, a tedious task when done by hand.

The keybindings are intended to be somewhat mnemonic.

Update Pointers

Create or update 'Next', 'Previous', and 'Up' node pointers.

See Section 2.3 [Updating Nodes and Menus], page 18.

`C-c C-u C-n`	Update a node.
`C-c C-u C-e`	Update every node in the buffer.

Update Menus

Create or update menus.

See Section 2.3 [Updating Nodes and Menus], page 18.

The `texinfo-master-menu` command lacks a keybinding since it is used less frequently than the other commands.

`C-c C-u C-m` Make or update a menu.

`C-c C-u C-a` Make or update all the menus in a buffer; with an argument, first update all the nodes.

`M-x texinfo-master-menu`
Create or update a master menu. With an argument, first create or update all nodes and regular menus.

Format for Info

Provide keybindings both for the Info formatting commands that are written in Emacs Lisp and for `makeinfo` which is written in C.

See Section 2.4 [Info Formatting], page 21.

Use the Emacs lisp `texinfo-format`... commands:

`C-c C-e C-r` Format the region.
`C-c C-e C-b` Format the buffer.

Use `makeinfo`:

`C-c C-m C-r` Format the region.
`C-c C-m C-b` Format the buffer.
`C-c C-m C-l` Recenter the `makeinfo` output buffer.
`C-c C-m C-k` Kill the `makeinfo` formatting job.

Typeset and Print

Typeset and print Texinfo documents from within Emacs.

See Section 2.5 [Formatting and Printing], page 22.

`C-c C-t C-r` Run TEX on the region.
`C-c C-t C-b` Run TEX on the buffer.
`C-c C-t C-i` Run `texindex`.

`C-c C-t C-p` Print the DVI file.
`C-c C-t C-q` Show the print queue.
`C-c C-t C-d` Delete a job from the print queue.
`C-c C-t C-k` Kill the current TEX formatting job.
`C-c C-t C-l` Recenter the output buffer.

Other Updating Commands

The "other updating commands" do not have standard keybindings because they are less frequently used.

See Section 2.3.2 [Other Updating Commands], page 20.

`M-x texinfo-insert-node-lines`
Insert missing node lines using section titles as node names.

`M-x texinfo-multiple-files-update`
Update a multi-file document.

`M-x texinfo-indent-menu-description`
Indent descriptions in menus.

`M-x texinfo-sequentially-update-node`
Insert node pointers in strict sequence.

G.2 New Texinfo @-Commands

The second edition of the Texinfo manual describes more than 50 commands that were not described in the first edition. A third or so of these commands existed in Texinfo but were not documented in the manual; the others are new. Here is a listing, with brief descriptions of them:

Indexing

Commands for creating your own index and for merging two indices:

See Chapter 15 [Indices], page 131.

`@defindex` *index-name*
: Define a new index and its indexing command.

`@synindex` *from-index into-index*
: Merge first index into second index.

Definitions

Many commands to help you describe functions, variables, macros, commands, user options, special forms and other such artifacts in a uniform format.

See Chapter 13 [Definition Commands], page 115.

`@deffn` *category name arguments...*
: Format a description for functions, interactive commands, and similar entities.

`@defvr, @defop,` ...
: 15 other related commands.

Glyphs

Special symbols to indicate the results of evaluation or an expansion, printed output, an error message, equivalence of expressions, and the location of point.

See Chapter 11 [Special Glyphs], page 103.

`@equiv{}` Equivalence: ‘ ≡ ’

`@error{}` Error message: ‘ error ’

`@expansion{}` Macro expansion: ‘ ↦ ’

`@point{}` Position of point: ‘ ⋆ ’

`@print{}` Printed output: ‘ ⊣ ’

`@result{}` Result of an expression: ‘ ⇒ ’

Headings

Commands to customize headings.

See Appendix C [Headings], page 183.

`@headings` *on-off-single-double*
: Headings on or off, single or double-sided.

`@evenfooting` [*left*] `@|` [*center*] `@|` [*right*]
: Footings for even-numbered (left-hand) pages.

`@evenheading, @everyheading, @oddheading,` ...
: Five other related commands.

`@thischapter`
: Insert name of chapter and chapter number.

`@thischaptername, @thisfile, @thistitle, @thispage`
: Related commands.

Formatting

Commands for formatting text:

See Chapter 10 [Quotations and Examples], page 95.

`@exdent` *line-of-text*
: Remove indentation.

`@flushleft` Left justify.

`@flushright` Right justify.

`@format` Do not narrow nor change font.

`@ftable` *formatting-command*
: Two column table with indexing.

`@lisp` For an example of Lisp code.

`@smallexample`
: Like `@example` but for `@smallbook`.

@heading series for Titles

Produce unnumbered headings that do not appear in a table of contents.

See Chapter 5 [Structuring], page 49.

`@heading` *title*
: Unnumbered section-like heading not listed table of contents of a printed manual.

`@chapheading`, `@majorheading`
`@subheading`, `@subsubheading`
: Related commands.

Fonts

Font commands.

See Section 8.2.2 [Smallcaps], page 89, and
Section 8.2.3 [Fonts], page 90.

`@r{`*text*`}` Print in roman font.

`@sc{`*text*`}` Print in SMALL CAPS font.

Miscellaneous

This list inlcudes a useful cross reference command among others.

See Section 7.1 [Cross Reference Commands], page 67,
see Section 18.7 [Overfull Hboxes], page 150,
see Chapter 14 [Footnotes], page 129,
see Section 9.4 [Inserting a Minus Sign], page 94,
see Appendix E [Refilling Paragraphs], page 199, and
see Section 3.4.3 [`@title` `@subtitle` and `@author` Commands], page 36.

`@author` *author*

Typeset author's names.

`@finalout` Produce cleaner printed output.

`@footnotestyle`
Specify footnote style.

`@minus{}` Generate a minus sign.

`@paragraphindent`
Specify paragraph indentation.

`@ref{`*node-name*`,` [*entry*]`,` [*topic*]`,` [*info-file*]`,` [*manual*]`}`
Make a reference. In the printed manual, the reference does not start with a 'See'.

`@title` *title*
Title in the alternative title page format.

`@subtitle` *subtitle*
Subtitle in the alternative title page format.

`@today{}` Insert the current date.

Command and Variable Index

This is an alphabetical list of all the @-commands and several variables. To make the list easier to use, the commands are listed without their preceding '@'.

T

U

V

W

X

Concept Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Short Contents

Table of Contents