GNU Texinfo

Texinfo

The GNU Documentation Format
for Texinfo version 4.8, 29 December 2004

Robert J. Chassell
Richard M. Stallman

This manual is for GNU Texinfo (version 4.8, 29 December 2004), a documentation system that can produce both online information and a printed manual from a single source.

Copyright (C) 1988, 1990, 1991, 1992, 1993, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004 Free Software Foundation, Inc.

> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being "A GNU Manual," and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled "GNU Free Documentation License."
>
> (a) The FSF's Back-Cover Text is: "You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development."

Published by the Free Software Foundation
59 Temple Place Suite 330
Boston, MA 02111-1307
USA
ISBN 1-882114-67-1

Cover art by Etienne Suvasa.

Short Contents

Texinfo Copying Conditions 2
1 Overview of Texinfo 3
2 Using Texinfo Mode 16
3 Beginning a Texinfo File 28
4 Ending a Texinfo File 45
5 Chapter Structuring 47
6 Nodes 53
7 Menus 60
8 Cross References 64
9 Marking Words and Phrases 74
10 Quotations and Examples 85
11 Lists and Tables 94
12 Special Displays 102
13 Indices 108
14 Special Insertions 113
15 Forcing and Preventing Breaks 123
16 Definition Commands 127
17 Conditionally Visible Text 140
18 Internationalization 147
19 Defining New Texinfo Commands 150
20 Formatting and Printing Hardcopy 156
21 Creating and Installing Info Files 168
22 Generating HTML 182
A @-Command List 190
B Tips and Hints 211
C Sample Texinfo Files 216
D Include Files 222
E Page Headings 226
F Formatting Mistakes 230
G Copying This Manual 237
Command and Variable Index 244
Concept Index 248

Table of Contents

Texinfo Copying Conditions 2

1 Overview of Texinfo 3
- 1.1 Reporting Bugs 3
- 1.2 Using Texinfo 3
- 1.3 Output Formats 4
- 1.4 Info Files 5
- 1.5 Printed Books 7
- 1.6 @-commands 8
- 1.7 General Syntactic Conventions 9
- 1.8 Comments 10
- 1.9 What a Texinfo File Must Have 10
- 1.10 Six Parts of a Texinfo File 11
- 1.11 A Short Sample Texinfo File 12
- 1.12 History 14

2 Using Texinfo Mode 16
- 2.1 Texinfo Mode Overview 16
- 2.2 The Usual GNU Emacs Editing Commands 16
- 2.3 Inserting Frequently Used Commands 17
- 2.4 Showing the Section Structure of a File 19
- 2.5 Updating Nodes and Menus 19
  - 2.5.1 The Updating Commands 19
  - 2.5.2 Updating Requirements 22
  - 2.5.3 Other Updating Commands 22
- 2.6 Formatting for Info 23
- 2.7 Printing 24
- 2.8 Texinfo Mode Summary 25

3 Beginning a Texinfo File 28
- 3.1 Sample Texinfo File Beginning 28
- 3.2 Texinfo File Header 29
  - 3.2.1 The First Line of a Texinfo File 30
  - 3.2.2 Start of Header 30
  - 3.2.3 `@setfilename`: Set the output file name 30
  - 3.2.4 `@settitle`: Set the document title 31
  - 3.2.5 End of Header 32
- 3.3 Document Permissions 32
  - 3.3.1 `@copying`: Declare Copying Permissions 32
  - 3.3.2 `@insertcopying`: Include Permissions Text 33
- 3.4 Title and Copyright Pages 33
  - 3.4.1 `@titlepage` 34

3.4.2 `@titlefont`, `@center`, and `@sp` ... 34
3.4.3 `@title`, `@subtitle`, and `@author` ... 35
3.4.4 Copyright Page ... 36
3.4.5 Heading Generation ... 37
3.4.6 The `@headings` Command ... 37
3.5 Generating a Table of Contents ... 38
3.6 The 'Top' Node and Master Menu ... 39
3.6.1 Top Node Example ... 40
3.6.2 Parts of a Master Menu ... 40
3.7 Global Document Commands ... 41
3.7.1 `@documentdescription`: Summary Text ... 41
3.7.2 `@setchapternewpage`: ... 41
3.7.3 `@paragraphindent`: Paragraph Indenting ... 42
3.7.4 `@firstparagraphindent`: Indenting After Headings ... 43
3.7.5 `@exampleindent`: Environment Indenting ... 43
3.8 Software Copying Permissions ... 44

4 Ending a Texinfo File ... 45
4.1 Printing Indices and Menus ... 45
4.2 `@bye` File Ending ... 46

5 Chapter Structuring ... 47
5.1 Tree Structure of Sections ... 47
5.2 Structuring Command Types ... 47
5.3 `@top` ... 48
5.4 `@chapter` ... 48
5.5 `@unnumbered` and `@appendix` ... 49
5.6 `@majorheading`, `@chapheading` ... 49
5.7 `@section` ... 49
5.8 `@unnumberedsec`, `@appendixsec`, `@heading` ... 50
5.9 The `@subsection` Command ... 50
5.10 The `@subsection`-like Commands ... 50
5.11 The 'subsub' Commands ... 51
5.12 `@raisesections` and `@lowersections` ... 51

6 Nodes ... 53
6.1 Two Paths ... 53
6.2 Node and Menu Illustration ... 53
6.3 The `@node` Command ... 55
6.3.1 Choosing Node and Pointer Names ... 55
6.3.2 How to Write an `@node` Line ... 56
6.3.3 `@node` Line Tips ... 56
6.3.4 `@node` Line Requirements ... 57
6.3.5 The First Node ... 57
6.3.6 The `@top` Sectioning Command ... 58
6.4 Creating Pointers with `makeinfo` ... 58
6.5 `@anchor`: Defining Arbitrary Cross-reference Targets ... 59

7 Menus 60

7.1 Menu Location 60
7.2 Writing a Menu 60
7.3 The Parts of a Menu 61
7.4 Less Cluttered Menu Entry 61
7.5 A Menu Example 61
7.6 Referring to Other Info Files 62

8 Cross References 64

8.1 What References Are For 64
8.2 Different Cross Reference Commands 64
8.3 Parts of a Cross Reference 65
8.4 `@xref` 66
8.4.1 What a Reference Looks Like and Requires 66
8.4.2 `@xref` with One Argument 66
8.4.3 `@xref` with Two Arguments 67
8.4.4 `@xref` with Three Arguments 67
8.4.5 `@xref` with Four and Five Arguments 68
8.5 Naming a 'Top' Node 70
8.6 `@ref` 70
8.7 `@pxref` 71
8.8 `@inforef` 72
8.9 `@url`, `@uref{`*`url`*`[, `*`text`*`][, `*`replacement`*`]}` 72

9 Marking Words and Phrases 74

9.1 Indicating Definitions, Commands, etc. 74
9.1.1 Highlighting Commands are Useful 74
9.1.2 `@code{`*sample-code*`}` 75
9.1.3 `@kbd{`*keyboard-characters*`}` 76
9.1.4 `@key{`*key-name*`}` 77
9.1.5 `@samp{`*text*`}` 78
9.1.6 `@verb{`<char>*text*<char>`}` 78
9.1.7 `@var{`*metasyntactic-variable*`}` 79
9.1.8 `@env{`*environment-variable*`}` 79
9.1.9 `@file{`*file-name*`}` 80
9.1.10 `@command{`*command-name*`}` 80
9.1.11 `@option{`*option-name*`}` 80
9.1.12 `@dfn{`*term*`}` 80
9.1.13 `@cite{`*reference*`}` 81
9.1.14 `@abbr{`*abbreviation*[, *meaning*]`}` 81
9.1.15 `@acronym{`*acronym*[, *meaning*]`}` 81
9.1.16 `@indicateurl{`*uniform-resource-locator*`}` 82
9.1.17 `@email{`*email-address*[, *displayed-text*]`}` 82
9.2 Emphasizing Text 83
9.2.1 `@emph{`*text*`}` and `@strong{`*text*`}` 83
9.2.2 `@sc{`*text*`}`: The Small Caps Font 83
9.2.3 Fonts for Printing, Not Info 84

10 Quotations and Examples 85

10.1 Block Enclosing Commands 85
10.2 `@quotation`: Block quotations 86
10.3 `@example`: Example Text 86
10.4 `@verbatim`: Literal Text 87
10.5 `@verbatiminclude` *file*: Include a File Verbatim 88
10.6 `@lisp`: Marking a Lisp Example 88
10.7 `@small...` Block Commands 89
10.8 `@display` and `@smalldisplay` 89
10.9 `@format` and `@smallformat` 90
10.10 `@exdent`: Undoing a Line's Indentation 90
10.11 `@flushleft` and `@flushright` 90
10.12 `@noindent`: Omitting Indentation 91
10.13 `@indent`: Forcing Indentation 92
10.14 `@cartouche`: Rounded Rectangles Around Examples 92

11 Lists and Tables 94

11.1 Introducing Lists 94
11.2 `@itemize`: Making an Itemized List 95
11.3 `@enumerate`: Making a Numbered or Lettered List 96
11.4 Making a Two-column Table 97
11.4.1 Using the `@table` Command 98
11.4.2 `@ftable` and `@vtable` 99
11.4.3 `@itemx` 99
11.5 `@multitable`: Multi-column Tables 99
11.5.1 Multitable Column Widths 100
11.5.2 Multitable Rows 100

12 Special Displays 102

12.1 Floats 102
12.1.1 `@float` [*type*][,*label*]: Floating material 102
12.1.2 `@caption` & `@shortcaption` 103
12.1.3 `@listoffloats`: Tables of contents for floats 103
12.2 Inserting Images 104
12.2.1 Image Syntax 104
12.2.2 Image Scaling 105
12.3 Footnotes 106
12.3.1 Footnote Commands 106
12.3.2 Footnote Styles 107

13 Indices 108

13.1 Making Index Entries 108
13.2 Predefined Indices 108
13.3 Defining the Entries of an Index 109
13.4 Combining Indices 110
13.4.1 `@syncodeindex` 110
13.4.2 `@synindex` 111
13.5 Defining New Indices 111

14 Special Insertions 113

14.1 Inserting @ and {} and , ... 113
14.1.1 Inserting '@' with @@ 113
14.1.2 Inserting '{' and '}' with @{ and @} 113
14.1.3 Inserting ',' with @comma{} 113
14.2 Inserting Space 114
14.2.1 Not Ending a Sentence 114
14.2.2 Ending a Sentence 114
14.2.3 Multiple Spaces 115
14.2.4 @dmn{*dimension*}: Format a Dimension 115
14.3 Inserting Accents 116
14.4 Inserting Ellipsis and Bullets 117
14.4.1 @dots{} (. . .) and @enddots{} (. . .) 117
14.4.2 @bullet{} (•) 117
14.5 Inserting TEX and Legal Symbols: ©, ® 117
14.5.1 @TeX{} (TEX) and @LaTeX{} (LATEX) 117
14.5.2 @copyright{} (©) 117
14.5.3 @registeredsymbol{} (®) 118
14.6 @euro{} (€): Euro currency symbol 118
14.7 @pounds{} (£): Pounds Sterling 118
14.8 @minus{} (−): Inserting a Minus Sign 118
14.9 @math: Inserting Mathematical Expressions 118
14.10 Glyphs for Examples 119
14.10.1 Glyphs Summary 119
14.10.2 @result{} (⇒): Indicating Evaluation 120
14.10.3 @expansion{} (↦): Indicating an Expansion 120
14.10.4 @print{} (⊣): Indicating Printed Output 120
14.10.5 @error{} (error): Indicating an Error Message 121
14.10.6 @equiv{} (≡): Indicating Equivalence 121
14.10.7 @point{} (⋆): Indicating Point in a Buffer 121

15 Forcing and Preventing Breaks 123

15.1 Break Commands 123
15.2 @* and @/: Generate and Allow Line Breaks 123
15.3 @- and @hyphenation: Helping TEX Hyphenate 124
15.4 @w{*text*}: Prevent Line Breaks 124
15.5 @tie{}: Inserting an Unbreakable Space 125
15.6 @sp *n*: Insert Blank Lines 125
15.7 @page: Start a New Page 125
15.8 @group: Prevent Page Breaks 125
15.9 @need *mils*: Prevent Page Breaks 126

16 Definition Commands 127

16.1 The Template for a Definition 127
16.2 Definition Command Continuation Lines 128
16.3 Optional and Repeated Arguments 129
16.4 Two or More 'First' Lines 129
16.5 The Definition Commands 130
16.5.1 Functions and Similar Entities 130
16.5.2 Variables and Similar Entities 131
16.5.3 Functions in Typed Languages 132
16.5.4 Variables in Typed Languages 133
16.5.5 Data Types 134
16.5.6 Object-Oriented Programming 135
16.5.6.1 Object-Oriented Variables 135
16.5.6.2 Object-Oriented Methods 136
16.6 Conventions for Writing Definitions 137
16.7 A Sample Function Definition 138

17 Conditionally Visible Text 140

17.1 Conditional Commands 140
17.2 Conditional Not Commands 141
17.3 Raw Formatter Commands 141
17.4 `@set`, `@clear`, and `@value` 142
17.4.1 `@set` and `@value` 143
17.4.2 `@ifset` and `@ifclear` 144
17.4.3 `@value` Example 145
17.5 Conditional Nesting 146

18 Internationalization 147

18.1 `@documentlanguage` *cc*: Set the Document Language 147
18.2 `@documentencoding` *enc*: Set Input Encoding 148

19 Defining New Texinfo Commands 150

19.1 Defining Macros 150
19.2 Invoking Macros 151
19.3 Macro Details and Caveats 152
19.4 '`@alias` *new=existing*' 153
19.5 '`definfoenclose`': Customized Highlighting 154

20 Formatting and Printing Hardcopy 156

20.1 Use TEX 156
20.2 Format with `tex` and `texindex` 156
20.3 Format with `texi2dvi` 157
20.4 Shell Print Using `lpr -d` 158
20.5 From an Emacs Shell 159
20.6 Formatting and Printing in Texinfo Mode 159
20.7 Using the Local Variables List 161
20.8 TEX Formatting Requirements Summary 161
20.9 Preparing for TEX 162
20.10 Overfull "hboxes" 163
20.11 Printing "Small" Books 164
20.12 Printing on A4 Paper 164
20.13 `@pagesizes` [*width*][, *height*]: Custom Page Sizes 165
20.14 Cropmarks and Magnification 165
20.15 PDF Output 166
20.16 How to Obtain TEX 166

21 Creating and Installing Info Files 168

21.1 Creating an Info File 168
21.1.1 `makeinfo` Preferred 168
21.1.2 Running `makeinfo` from a Shell 168
21.1.3 Options for `makeinfo` 168
21.1.4 Pointer Validation 172
21.1.5 Running `makeinfo` Within Emacs 173
21.1.6 The `texinfo-format...` Commands 174
21.1.7 Batch Formatting 175
21.1.8 Tag Files and Split Files 175
21.2 Installing an Info File 176
21.2.1 The Directory File '`dir`' 176
21.2.2 Listing a New Info File 177
21.2.3 Info Files in Other Directories 177
21.2.4 Installing Info Directory Files 179
21.2.5 Invoking `install-info` 180

22 Generating HTML 182

22.1 HTML Translation 182
22.2 HTML Splitting 182
22.3 HTML CSS 183
22.4 HTML Cross-references 184
22.4.1 HTML Cross-reference Link Basics 184
22.4.2 HTML Cross-reference Node Name Expansion 185
22.4.3 HTML Cross-reference Command Expansion 186
22.4.4 HTML Cross-reference 8-bit Character Expansion 188
22.4.5 HTML Cross-reference Mismatch 188

Appendix A @-Command List 190
A.1 @-Command Syntax 209

Appendix B Tips and Hints 211

Appendix C Sample Texinfo Files 216
C.1 Short Sample 216
C.2 GNU Sample Texts 217
C.3 Verbatim Copying License 220
C.4 All-permissive Copying License 221

Appendix D Include Files 222
D.1 How to Use Include Files 222
D.2 `texinfo-multiple-files-update` 222
D.3 Include Files Requirements 223
D.4 Sample File with `@include` 224
D.5 Evolution of Include Files 224

Appendix E Page Headings 226
E.1 Headings Introduced 226
E.2 Standard Heading Formats 226
E.3 Specifying the Type of Heading 227
E.4 How to Make Your Own Headings 228

Appendix F Formatting Mistakes 230
F.1 `makeinfo` Find Errors 230
F.2 Catching Errors with Info Formatting 230
F.3 Catching Errors with TEX Formatting 231
F.4 Using `texinfo-show-structure` 233
F.5 Using `occur` 234
F.6 Finding Badly Referenced Nodes 234
F.6.1 Running `Info-validate` 235
F.6.2 Creating an Unsplit File 235
F.6.3 Tagifying a File 236
F.6.4 Splitting a File Manually 236

Appendix G Copying This Manual 237
G.1 GNU Free Documentation License 237
G.1.1 ADDENDUM: How to use this License for your documents 243

Command and Variable Index 244

Concept Index 248

Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing. —Dick Brandon

Texinfo Copying Conditions

The programs currently being distributed that relate to Texinfo include `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. This manual specifically is covered by the GNU Free Documentation License (see Section G.1 [GNU Free Documentation License], page 237).

1 Overview of Texinfo

Texinfo[1] is a documentation system that uses a single source file to produce both online information and printed output. This means that instead of writing two different documents, one for the online information and the other for a printed work, you need write only one document. Therefore, when the work is revised, you need revise only that one document.

1.1 Reporting Bugs

We welcome bug reports and suggestions for any aspect of the Texinfo system, programs, documentation, installation, anything. Please email them to `bug-texinfo@gnu.org`. You can get the latest version of Texinfo from `ftp://ftp.gnu.org/gnu/texinfo/` and its mirrors worldwide.

For bug reports, please include enough information for the maintainers to reproduce the problem. Generally speaking, that means:

- the version number of Texinfo and the program(s) or manual(s) involved.
- hardware and operating system names and versions.
- the contents of any input files necessary to reproduce the bug.
- a description of the problem and samples of any erroneous output.
- any unusual options you gave to `configure`.
- anything else that you think would be helpful.

When in doubt whether something is needed or not, include it. It's better to include too much than to leave out something important.

Patches are most welcome; if possible, please make them with '`diff -c`' (see section "Overview" in *Comparing and Merging Files*) and include '`ChangeLog`' entries (see section "Change Log" in *The GNU Emacs Manual*).

When sending patches, if possible please do not encode or split them in any way; it's much easier to deal with one plain text message, however large, than many small ones. GNU shar (`ftp://ftp.gnu.org/gnu/sharutils/`) is a convenient way of packaging multiple and/or binary files for email.

1.2 Using Texinfo

Using Texinfo, you can create a printed document (via the TEX typesetting system) the normal features of a book, including chapters, sections, cross references, and indices. From the same Texinfo source file, you can create an Info file with special features to make documentation browsing easy. You can also create from that same source file an HTML output file suitable for use with a web browser, or an XML file. See the next section (see Section 1.3 [Output Formats], page 4) for details and the exact commands to generate output from the source.

[1] The first syllable of "Texinfo" is pronounced like "speck", not "hex". This odd pronunciation is derived from, but is not the same as, the pronunciation of TEX. In the word TEX, the '`X`' is actually the Greek letter "chi" rather than the English letter "ex". Pronounce TEX as if the '`X`' were the last sound in the name 'Bach'; but pronounce Texinfo as if the '`x`' were a 'k'. Spell "Texinfo" with a capital "T" and the other letters in lower case.

TEX works with virtually all printers; Info works with virtually all computer terminals; the HTML output works with virtually all web browsers. Thus Texinfo can be used by almost any computer user.

A Texinfo source file is a plain ASCII file containing text interspersed with `@`-*commands* (words preceded by an '`@`') that tell the typesetting and formatting programs what to do. You can 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 16.)

You can use Texinfo to create both online help and printed manuals; moreover, Texinfo is freely redistributable. For these reasons, Texinfo is the official documentation format of the GNU project. More information is available at the GNU documentation web page (`http://www.gnu.org/doc/`).

1.3 Output Formats

Here is a brief overview of the output formats currently supported by Texinfo.

Info (Generated via `makeinfo`.) This format is essentially a plain text transliteration of the Texinfo source. It adds a few control characters to separate nodes and provide navigational information for menus, cross-references, indices, and so on. See the next section (see Section 1.4 [Info Files], page 5) for more details on this format. The Emacs Info subsystem (see section "Getting Started" in *Info*), and the standalone `info` program (see section "Info Standalone" in *GNU Info*), among others, can read these files. See Chapter 21 [Creating and Installing Info Files], page 168.

Plain text (Generated via `makeinfo --no-headers`.) This is almost the same as Info output, except the navigational control characters are omitted. Also, standard output is used by default.

HTML (Generated via `makeinfo --html`.) This is the Hyper Text Markup Language that has become the most commonly used language for writing documents on the World Wide Web. Web browsers, such as Mozilla, Lynx, and Emacs-W3, can render this language online. There are many versions of HTML; `makeinfo` tries to use a subset of the language that can be interpreted by any common browser. For details of the HTML language and much related information, see `http://www.w3.org/MarkUp/`. See Chapter 22 [Generating HTML], page 182.

DVI (Generated via `texi2dvi`.) This DeVice Independent binary format is output by the TEX typesetting program (`http://tug.org`). This is then read by a DVI 'driver', which writes the actual device-specific commands that can be viewed or printed, notably Dvips for translation to PostScript (see section "Invoking Dvips" in *Dvips*) and Xdvi for viewing on an X display (`http://sourceforge.net/projects/xdvi/`). See Chapter 20 [Hardcopy], page 156.

Be aware that the Texinfo language is very different from and much stricter than TEX's usual languages, plain TEX and LATEX. For more information on TEX in general, please see the book *TEX for the Impatient*, available from `http://savannah.gnu.org/projects/teximpatient`.

PDF (Generated via `texi2dvi --pdf` or `texi2pdf`.) This was developed by Adobe Systems for portable document interchange, based on their previous PostScript language. It can represent the exact appearance of a document, including fonts, and supporting arbitrary scaling. It is intended to be platform-independent and easily viewable, among other design goals; for a discussion, see `http://tug.org/TUGboat/Articles/tb22-3/tb72beebeI.pdf`. Texinfo uses the `pdftex` program, a variant of TEX, to output PDF; see `http://tug.org/applications/pdftex`. See Section 20.15 [PDF Output], page 166.

XML (Generated via `makeinfo --xml`.) XML is a generic syntax specification usable for any sort of content (see, for example, `http://www.w3.org/XML/`). The `makeinfo` xml output, unlike all the formats above, interprets very little of the Texinfo source. Rather, it merely translates the Texinfo markup commands into XML syntax, for processing by further XML tools. The particular syntax output is defined in the file '`texinfo.dtd`' included in the Texinfo source distribution.

Docbook (Generated via `makeinfo --docbook`.) This is an XML-based format developed some years ago, primarily for technical documentation. It therefore bears some resemblance, in broad outlines, to Texinfo. See `http://www.docbook.org`. If you want to convert from Docbook *to* Texinfo, please see `http://docbook2X.sourceforge.net`.

From time to time, proposals are made to generate traditional Unix man pages from Texinfo source. However, because man pages have a very strict conventional format, generating a good man page requires a completely different source than the typical Texinfo applications of writing a good user tutorial and/or a good reference manual. This makes generating man pages incompatible with the Texinfo design goal of not having to document the same information in different ways for different output formats. You might as well just write the man page directly.

Man pages still have their place, and if you wish to support them, you may find the program `help2man` to be useful; it generates a traditional man page from the '`--help`' output of a program. In fact, this is currently used to generate man pages for the programs in the Texinfo distribution. It is GNU software written by Brendan O'Dea, available from `ftp://ftp.gnu.org/gnu/help2man/`.

If you are a programmer and would like to contribute to the GNU project by implementing additional output formats for Texinfo, that would be excellent. But please do not write a separate translator texi2foo for your favorite format foo! That is the hard way to do the job, and makes extra work in subsequent maintenance, since the Texinfo language is continually being enhanced and updated. Instead, the best approach is modify `makeinfo` to generate the new format.

1.4 Info Files

An Info file is a Texinfo file formatted so that the Info documentation reading program can operate on it. (`makeinfo` and `texinfo-format-buffer` are two commands that convert a Texinfo file into an Info file.)

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 other related nodes.

See section "Top" in *GNU Info*, for more information about using Info.

Each node of an Info file may have any number of child nodes that describe subtopics of the node's topic. The names of child nodes 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, an Info 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. The 'Next' pointer provides a link to the next section, and the 'Previous' pointer provides a link to the previous section. 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 book or report.

In addition to menus and to 'Next', 'Previous', and 'Up' pointers, 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 output. But occasionally there are times when this is not right for the material being discussed. Therefore, Texinfo uses separate commands to specify the node structure for the Info file and the section structure for the printed output.

Generally, you enter an Info file through a node that by convention is named '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. Alternatively, with the standalone Info program, you can specify specific menu items on the command line (see section "Top" in *Info*).

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

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

The '`dir`' file in the '`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.

If you wish to refer to an Info file in a URI, you can use the (unofficial) syntax exemplified in the following. This works with Emacs/W3, for example:

```
info:///usr/info/emacs#Dissociated%20Press
info:emacs#Dissociated%20Press
info://localhost/usr/info/emacs#Dissociated%20Press
```

The `info` program itself does not follow URI's of any kind.

1.5 Printed Books

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

A Texinfo-based book is similar to any other typeset, printed work: it can 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 online information. You can use Texinfo for writing a printed novel, and even to write a printed 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 '`texinfo.tex`' that contains information (definitions or *macros*) that TEX uses when it typesets a Texinfo file. ('`texinfo.tex`' tells 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. You can get the latest version of '`texinfo.tex`' from `ftp://ftp.gnu.org/gnu/texinfo/texinfo.tex`.

In the United States, documents are most often printed on 8.5 inch by 11 inch pages (216 mm by 280 mm); this is the default size. But you can also print for 7 inch by 9.25 inch pages (178 mm by 235 mm, the `@smallbook` size; or on A4 or A5 size paper (`@afourpaper`, `@afivepaper`). (See Section 20.11 [Printing "Small" Books], page 164. Also, see Section 20.12 [Printing on A4 Paper], page 164.)

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 superset of Pascal called WEB and can be compiled either in Pascal or (by using a conversion program that comes with the TEX

[3] You can also use the `texi2roff` (`ftp://tug.org/texi2roff.tar.gz`) program if you do not have TEX; since Texinfo is designed for use with TEX, `texi2roff` is not described here. `texi2roff` is not part of the standard GNU distribution and is not maintained or up-to-date with all the Texinfo features described in this manual.

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.

To get a copy of TEX, see Section 20.16 [How to Obtain TEX], page 166.

1.6 `@`-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.

Note: Almost all `@` command names are entirely 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:

- Write a command such as `@quotation` at the beginning of a line as the only text on the line. (`@quotation` begins an indented environment.)
- 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. (`@chapter` creates chapter titles.)
- Write a command such as `@dots{}` wherever you wish but usually within a sentence. (`@dots{}` creates dots ...)
- 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. (`@code` marks text as being code.)
- Write a command such as `@example` on a line of its own; write the body-text on following lines; and write the matching `@end` command, `@end example` in this case, on a line of its own after the body-text. (`@example` ... `@end example` indents and typesets body-text as an example.) It's usually ok to indent environment commands like this, but in complicated and hard-to-define circumstances the extra spaces cause extra space to appear in the output, so beware.

As a general rule, a command requires braces if it mingles among other text; but it does not need braces if it starts a line of its own. The non-alphabetic commands, such as `@:`, are exceptions to the rule; they do not need braces.

[4] The word *argument* comes from the way it is used in mathematics and does not refer to a dispute 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 dispute.

As you gain experience with Texinfo, you will rapidly learn how to write the different commands: the different ways to write commands actually make it easier to write and read Texinfo files than if all commands followed exactly the same syntax. See Section A.1 [@-Command Syntax], page 209, for all the details.

1.7 General Syntactic Conventions

This section describes the general conventions used in all Texinfo documents.

- All printable ASCII characters except '`@`', '`{`' and '`}`' can appear in a Texinfo file and stand for themselves. '`@`' is the escape character which introduces commands, while '`{`' and '`}`' are used 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 '`@}`'.
- Separate paragraphs with one or more blank lines. Currently Texinfo only recognizes newline characters as end of line, not the CRLF sequence used on some systems; so a *blank line* means exactly two consecutive newlines. Sometimes blank lines are useful or convenient in other cases as well; you can use the `@noindent` to inhibit paragraph indentation if required (see Section 10.12 [@noindent], page 91).
- Use doubled single-quote characters to begin and end quotations: ``` ``...'' ```. TEX converts two single quotes to left- and right-hand doubled quotation marks, "like this", and Info converts doubled single-quote characters to ASCII double-quotes: ``` ``...'' ``` becomes `"..."`.

  You may occasionally need to produce two consecutive single quotes; for example, in documenting a computer language such as Maxima where `''` is a valid command. You can do this with the input `'@w{}'`; the empty `@w` command stops the combination into the double-quote characters.

  The left quote character (`` ` ``, ASCII code 96) used in Texinfo is a grave accent in ANSI and ISO character set standards. We use it as a quote character because that is how TEX is set up, by default. We hope to eventually support the various quotation characters in Unicode.
- Use three hyphens in a row, '`---`', to produce a long dash—like this (called an *em dash*), used for punctuation in sentences. Use two hyphens, '`--`', to produce a medium dash (called an *en dash*), used primarily for numeric ranges, as in "June 25–26". Use a single hyphen, '`-`', to produce a standard hyphen used in compound words. For display on the screen, Info reduces three hyphens to two and two hyphens to one (not transitively!). Of course, any number of hyphens in the source remain as they are in literal contexts, such as `@code` and `@example`.
- **Caution:** Last and most important, do not use tab characters in a Texinfo file (except in verbatim modes)! TEX uses variable-width fonts, which means that it is impractical at best to define a tab to work in all circumstances. Consequently, TEX treats tabs like single spaces, and that is not what they look like in the source. Furthermore, `makeinfo` does nothing special with tabs, and thus a tab character in your input file will usually appear differently in the output.

  To avoid this problem, Texinfo mode causes GNU Emacs to insert multiple spaces when you press the ⟨TAB⟩ key.

Also, you can run `untabify` in Emacs to convert tabs in a region to multiple spaces, or use the `unexpand` command from the shell.

1.8 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 revises 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.

Often, you can write the `@comment` or `@c` in the middle of a line, and only the text that follows after the `@comment` or `@c` command does not appear; but some commands, such as `@settitle` and `@setfilename`, work on a whole line. You cannot use `@comment` or `@c` in a line beginning with such a command.

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.

Text enclosed by `@ignore` or by failing `@ifset` or `@ifclear` conditions is ignored in the sense that it will not contribute to the formatted output. However, TEX and makeinfo must still parse the ignored text, in order to understand when to *stop* ignoring text from the source file; that means that you may still get error messages if you have invalid Texinfo commands within ignored text.

1.9 What a Texinfo File Must Have

By convention, the namea of a Texinfo file ends with (in order of preference) one of the extensions '`.texinfo`', '`.texi`', '`.txi`', or '`.tex`'. The longer extensions are preferred since they describe 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
```

Here's an explanation:

- 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. (Note the use of the backslash, '\'; this is correct for TEX.)
- The `@setfilename` line provides a name for the Info file and tells TEX to open auxiliary files. **All text before `@setfilename` is ignored!**

- The `@settitle` line specifies a title for the page headers (or footers) of the printed manual, and the default document description for the '`<head>`' in HTML format. Strictly speaking, `@settitle` is optional—if you don't mind your document being titled 'Untitled'.
- The `@bye` line at the end of the file on a line of its own tells the formatters that the file is ended and to stop formatting.

Typically, you will not 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 30.)

Furthermore, you will usually provide a Texinfo file with a title page, indices, and the like, all of which are explained in this manual. 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.

1.10 Six Parts of a Texinfo File

Generally, a Texinfo file contains more than the minimal beginning and end described in the previous section—it usually contains the six parts listed below. These are described fully in the following sections.

1. Header The *Header* names the file, tells TEX which definitions file to use, and other such housekeeping tasks.

2. Summary and Copyright
   The *Summary and Copyright* segment describes the document and contains the copyright notice and copying permissions. This is done with the `@copying` command.

3. Title and Copyright
   The *Title and Copyright* segment contains the title and copyright pages for the printed manual. The segment must be enclosed between `@titlepage` and `@end titlepage` commands. The title and copyright page appear only in the printed manual.

4. 'Top' Node and Master Menu
   The 'Top' node starts off the online output; it does not appear in the printed manual. We recommend including the copying permissions here as well as the segments above. And it contains at least a top-level menu listing the chapters, and possibly a *Master Menu* listing all the nodes in the entire document.

5. Body The *Body* of the document is typically structured like a traditional book or encyclopedia, but it may be free form.

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

1.11 A Short Sample Texinfo File

Here is a very short but complete Texinfo file, in the six conventional parts enumerated in the previous section, so you can see how Texinfo source appears in practice. 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 writing a manual, you simply change the names as appropriate.

See Chapter 3 [Beginning a File], page 28, for full documentation on the commands listed here. See Section C.2 [GNU Sample Texts], page 217, for the full texts to be used in GNU manuals.

In the following, the sample text is *indented*; comments on it are not. The complete file, without interspersed comments, is shown in Section C.1 [Short Sample Texinfo File], page 216.

Part 1: Header

The header does not appear in either the Info file or the printed output. It sets various parameters, including the name of the Info file and the title used in the header.

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename sample.info
@settitle Sample Manual 1.0
@c %**end of header
```

Part 2: Summary Description and Copyright

A real manual includes more text here, according to the license under which it is distributed. See Section C.2 [GNU Sample Texts], page 217.

```
@copying
This is a short example of a complete Texinfo file, version 1.0.

Copyright @copyright{} 2004 Free Software Foundation, Inc.
@end copying
```

Part 3: Titlepage, Contents, Copyright

The titlepage segment does not appear in the online output, only in the printed manual. We use the `@insertcopying` command to include the permission text from the previous section, instead of writing it out again; it is output on the back of the title page. The `@contents` command generates a table of contents.

```
@titlepage
@title Sample Title
```

```
@c The following two commands start the copyright page.
@page
@vskip 0pt plus 1filll
@insertcopying
@end titlepage

@c Output the table of contents at the beginning.
@contents
```

Part 4: 'Top' Node and Master Menu

The 'Top' node contains the master menu for the Info file. Since the printed manual uses a table of contents rather than a menu, it excludes the 'Top' node. We also include the copying text again for the benefit of online readers. Since the copying text begins with a brief description of the manual, no other text is needed in this case. The '`@top`' command itself helps `makeinfo` determine the relationships between nodes.

```
@ifnottex
@node Top
@top Short Sample

@insertcopying
@end ifnottex

@menu
* First Chapter::    The first chapter is the
                       only chapter in this sample.
* Index::            Complete index.
@end menu
```

Part 5: The Body of the Document

The body segment contains all the text of the document, but not the indices or table of contents. This example illustrates a node and a chapter containing an enumerated list.

```
@node First Chapter
@chapter First Chapter

@cindex chapter, first

This is the first chapter.
@cindex index entry, another
```

```
Here is a numbered list.

@enumerate
@item
This is the first item.

@item
This is the second item.
@end enumerate
```

Part 6: The End of the Document

The end segment contains commands for generating an index in a node and unnumbered chapter of its own, and the `@bye` command that marks the end of the document.

```
@node Index
@unnumbered Index

@printindex cp

@bye
```

Some Results

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

> This is the first chapter.
>
> Here is a numbered list.
>
> 1. This is the first item.
> 2. This is the second item.

1.12 History

Richard M. Stallman invented the Texinfo format, wrote the initial processors, and created Edition 1.0 of this manual. Robert J. Chassell greatly revised and extended the manual, starting with Edition 1.1. Brian Fox was responsible for the standalone Texinfo distribution until version 3.8, and wrote the standalone `makeinfo` and `info` programs. Karl Berry has continued maintenance since Texinfo 3.8 (manual edition 2.22).

Our thanks go out to all who helped improve this work, particularly the indefatigable Eli Zaretskii and Andreas Schwab, who have provided patches beyond counting. François Pinard and David D. Zuhn, tirelessly recorded and reported mistakes and obscurities. Zack Weinberg did the impossible by implementing the macro syntax in '`texinfo.tex`'. Special thanks go to Melissa Weisshaus for her frequent reviews of nearly similar editions. Dozens of others have contributed patches and suggestions, they are gratefully acknowledged in the '`ChangeLog`' file. Our mistakes are our own.

A bit of history: in the 1970's at CMU, Brian Reid developed a program and format named Scribe to mark up documents for printing. It used the `@` character to introduce commands, as Texinfo does. Much more consequentially, it strived to describe document contents rather than formatting, an idea wholeheartedly adopted by Texinfo.

Meanwhile, people at MIT developed another, not too dissimilar format called Bolio. This then was converted to using TEX as its typesetting language: BoTEX. The earliest BoTEX version seems to have been 0.02 on October 31, 1984.

BoTEX could only be used as a markup language for documents to be printed, not for online documents. Richard Stallman (RMS) worked on both Bolio and BoTEX. He also developed a nifty on-line help format called Info, and then combined BoTEX and Info to create Texinfo, a mark up language for text that is intended to be read both online and as printed hard copy.

2 Using Texinfo Mode

You may edit a Texinfo file with any text editor you choose. A Texinfo file is no different from any other ASCII file. However, GNU Emacs comes with a special mode, called Texinfo mode, that provides Emacs commands and tools to help ease your work.

This chapter describes features of GNU Emacs' Texinfo mode but not any features of the Texinfo formatting language. So if you are reading this manual straight through from the beginning, you may want to skim through this chapter briefly and come back to it after reading succeeding chapters which describe the Texinfo formatting language in detail.

2.1 Texinfo Mode Overview

Texinfo mode provides special features for working with Texinfo files. You can:

- Insert frequently used `@`-commands.
- 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.

Perhaps the two most helpful features are those for inserting frequently used `@`-commands and for creating node pointers and menus.

2.2 The Usual GNU Emacs Editing Commands

In most cases, the usual Text mode commands work the same in Texinfo mode as they do in Text mode. Texinfo mode adds new editing commands and tools to GNU Emacs' general purpose editing features. The major difference concerns filling. 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.

In addition, Texinfo mode sets the `page-delimiter` variable to the value of `texinfo-chapter-level-regexp`; by default, this is a regular expression matching the commands for chapters and their equivalents, such as appendices. With this value for the page delimiter, you can jump from chapter title to chapter title with the *C-x]* (`forward-page`) and *C-x [* (`backward-page`) commands and narrow to a chapter with the *C-x p* (`narrow-to-page`) command. (See section "Pages" in *The GNU Emacs Manual*, for details about the page commands.)

You may name a Texinfo file however you wish, but the convention is to end a Texinfo file name with one of the extensions '`.texinfo`', '`.texi`', '`.txi`', 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 a '`.texinfo`', '`.texi`' or '`.txi`' extension. Also, Emacs switches

to Texinfo mode when you visit 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.

2.3 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` and attempt to insert the correct following word, such as '`example`' or '`table`'. (This command does not handle nested lists correctly, but inserts the word appropriate to the immediately preceding list.)

`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 t`
`M-x texinfo-insert-@table`
: Insert `@table` followed by a ⟨SPC⟩ and leave the cursor after the ⟨SPC⟩.

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 }
C-c C-c]
M-x up-list
: Move from between a pair of braces forward past the closing brace. Typing *C-c C-c]* is easier than typing *C-c C-c }*, which is, however, more mnemonic; hence the two keybindings. (Also, you can move out from between braces by typing *C-f*.)

To put a command such as `@code{...}` around an *existing* word, position the cursor in front of the word and type *C-u 1 C-c C-c c*. This makes it easy to edit existing plain text. The value of the prefix argument tells Emacs how many words following point to include between braces—'`1`' for one word, '`2`' for two words, and so on. Use a negative argument to enclose the previous word or words. If you do not specify a prefix argument, Emacs inserts the `@`-command string and positions the cursor between the braces. This feature works only for those `@`-commands that operate on a word or words within one line, such as `@kbd` and `@var`.

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`'.

C-c C-c C-d (`texinfo-start-menu-description`) is an insert command that works differently from the other insert commands. It inserts a node's section or chapter title in the space for the description in a menu entry line. (A menu entry has three parts, the entry name, the node name, and the description. Only the node name is required, but a description helps explain what the node is about. See Section 7.3 [The Parts of a Menu], page 61.)

To use `texinfo-start-menu-description`, position point in a menu entry line and type *C-c C-c C-d*. The command looks for and copies the title that goes with the node name, and inserts the title as a description; it positions point at beginning of the inserted text so you can edit it. The function does not insert the title if the menu entry line already contains a description.

This command is only an aid to writing descriptions; it does not do the whole job. You must edit the inserted text since a title tends to use the same words as a node name but a useful description uses different words.

2.4 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. It 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. You can use `texinfo-show-structure` with a prefix argument to check whether the 'Next', 'Previous', and 'Up' pointers of an `@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 by using the *C-x n n* (`narrow-to-region`) command and `texinfo-show-structure` will work on only that region. To see the whole buffer again, use *C-x n w* (`widen`). (See section "Narrowing" in *The GNU Emacs Manual*, for more information about the narrowing commands.)

In addition to providing the `texinfo-show-structure` command, Texinfo mode sets the value of the page delimiter variable to match the chapter-level `@`-commands. This enables you to use the *C-x]* (`forward-page`) and *C-x [* (`backward-page`) commands to move forward and backward by chapter, and to use the *C-x p* (`narrow-to-page`) command to narrow to a chapter. See section "Pages" in *The GNU Emacs Manual*, for more information about the page commands.

2.5 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 an `@node` line that has none and to create menus in a file that has none.

If you do not use the updating commands, you need to write menus and node pointers by hand, which is a tedious task.

2.5.1 The Updating Commands

You can use the updating commands to:

- insert or update the 'Next', 'Previous', and 'Up' pointers of a node,

- insert or update the menu for a section, and
- 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.

The updating commands work only with conventional Texinfo files, which are structured hierarchically like books. In such files, a structuring command line must follow closely after each `@node` line, except for the 'Top' `@node` line. (A *structuring command line* is a line beginning with `@chapter`, `@section`, or other similar command.)

You can write the structuring command line on the line that follows immediately after an `@node` line or else on the line that follows after a single `@comment` line or a single `@ifinfo` line. You cannot interpose more than one line between the `@node` line and the structuring command line; and you may interpose only an `@comment` line or an `@ifinfo` line.

Commands which work on a whole buffer require that the 'Top' node be followed by a node with an `@chapter` or equivalent-level command. The menu updating commands will not create a main or master menu for a Texinfo file that has only `@chapter`-level nodes! The menu updating commands only create menus *within* nodes for lower level nodes. To create a menu of chapters, you must provide a 'Top' node.

The menu updating commands remove menu entries that refer to other Info files since they do not refer to nodes within the current buffer. This is a deficiency. Rather than use menu entries, you can use cross references to refer to other Info files. None of the updating commands affect cross references.

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, *C-u,* if interactive), first create or update all the nodes and all the regular menus in the buffer before constructing the master menu. (See Section 3.6 [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 and at least one subsequent node.

After extensively editing a Texinfo file, you can type the following:

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

or

```
C-u C-c C-u m
```

This updates 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 that 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, *C-u*, 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 (*C-u* as prefix argument, if interactive), the command makes or updates menus for the nodes which are either within or a 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.

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 (*C-u* as 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.)

  When working on a document that does not merit a master menu, you can type the following:

  ```
  C-u C-c C-u C-a
  ```

  or

  ```
  C-u M-x texinfo-all-menus-update
  ```

  This updates all the nodes and menus.

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 via customization (see section "Changing an Option" 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 menu descriptions to a specified column. Finally, if you wish, you can use the `texinfo-insert-node-lines` command to insert missing `@node` lines into a file. (See Section 2.5.3 [Other Updating Commands], page 22, for more information.)

2.5.2 Updating Requirements

To use the updating commands, you must organize the Texinfo file hierarchically with chapters, sections, subsections, and the like. When you construct the hierarchy of the manual, do not 'jump down' more than one level at a time: you can follow the 'Top' node with a chapter, but not with a section; you can follow a chapter with a section, but not with a subsection. However, you may 'jump up' any number of levels at one time—for example, from a subsection to a chapter.

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
```

or like this (without the explicit node pointers):

```
@node Comments
@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. (Instead of an `@comment` line, you may also write an `@ifinfo` line.)

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

The menu updating commands create a menu of sections within a chapter, a menu of subsections within a section, and so on. This means that you must have a 'Top' node if you want a menu of chapters.

Incidentally, the `makeinfo` command will create an Info file for a 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 Section 21.1 [Creating an Info File], page 168, for more information about `makeinfo`.) However, both `makeinfo` and the `texinfo-format-...` commands require that you insert menus in the file.

2.5.3 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` lines before the `@chapter`, `@section`, and other sectioning commands wherever they are missing throughout a region in a Texinfo file.

  With an argument (*C-u* as 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. In addition, it inserts the titles as node names in pre-existing `@node` lines that lack names. Since node names should be more concise than section or chapter titles, you must manually edit node names so inserted.

  For example, the following marks a whole buffer as a region and inserts `@node` lines and titles throughout:

  ```
  C-x h C-u M-x texinfo-insert-node-lines
  ```

  This command inserts titles as node names in `@node` lines; the `texinfo-start-menu-description` command (see Section 2.3 [Inserting], page 17) inserts titles as descriptions in menu entries, a different action. However, in both cases, you need to edit the inserted text.

M-x texinfo-multiple-files-update
: Update nodes and menus in a document built from several separate files. With *C-u* as a prefix argument, create and insert a master menu in the outer file. With a numeric prefix argument, such as *C-u 2*, 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 Section D.2 [texinfo-multiple-files-update], page 222.

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 (*C-u* as 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-sequential-node-update
: 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 ** 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-sequential-node-update` command sequentially updates all the nodes in the region.

2.6 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.

For example, after writing a Texinfo file, you can type the following:

```
C-u C-c C-u m
```

or

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

This updates all the nodes and menus. Then type the following to create an Info file:

```
C-c C-m C-b
```

or

```
M-x makeinfo-buffer
```

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

See Section 21.1 [Creating an Info File], page 168, for details about Info formatting.

2.7 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 print the file. Optionally, you may also create indices. To do this, you must run the `texindex` command after first running the `tex` typesetting command; and then you must run the `tex` command again. Or else run the `texi2dvi` command which automatically creates indices as needed (see Section 20.3 [Format with texi2dvi], page 157).

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-b
M-x texinfo-tex-buffer
: Run `texi2dvi` on the buffer. In addition to running TEX on the buffer, this command automatically creates or updates indices as needed.

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

C-c C-t C-i
M-x texinfo-texindex
: Run `texindex` to sort the indices of a Texinfo file formatted with `texinfo-tex-region`. The `texinfo-tex-region` command does not run `texindex` automatically; it only runs the `tex` typesetting command. You must run the `texinfo-tex-region` command a second time after sorting the raw index files with the `texindex` command. (Usually, you do not format an index when you format a region, only when you format a buffer. Now that the `texi2dvi` command exists, there is little or no need for this command.)

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. The file must end with `@bye` on a line by itself. (When you use `texinfo-tex-region`, you must surround the `@settitle` line with start-of-header and end-of-header lines.)

See Chapter 20 [Hardcopy], page 156, for a description of the other TEX related commands, such as `tex-show-print-queue`.

2.8 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 are somewhat mnemonic.

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 ]`
`C-c C-c }` Move out of enclosing braces.

`C-c C-c C-d` Insert a node's section title in the space for the description in a menu entry line.

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.

`C-u C-c C-u m` With `C-u` as a prefix argument, first create or update all nodes and regular menus, and then create a master menu.

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

Invoke the update menu commands by typing `C-c C-u` and then either `C-m` for `texinfo-make-menu` or `C-a` for `texinfo-all-menus-update`. To update both nodes and menus at the same time, precede `C-c C-u C-a` with `C-u`.

`C-c C-u C-m` Make or update a menu.

`C-c C-u C-a` Make or update all menus in a buffer.

`C-u C-c C-u C-a` With `C-u` as a prefix argument, first create or update all nodes and then create or update all menus.

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.

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 `texi2dvi` 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-x` Quit a currently stopped TEX formatting job.
`C-c C-t C-l` Recenter the output buffer.

Other Updating Commands

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

`M-x texinfo-insert-node-lines`
Insert missing `@node` lines in region.
With *C-u* as a prefix argument,
use section titles as node names.

`M-x texinfo-multiple-files-update`
Update a multi-file document.
With *C-u 2* as a prefix argument,
create or update all nodes and menus
in all included files first.

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

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

3 Beginning a Texinfo File

Certain pieces of information must be provided at the beginning of a Texinfo file, such as the name for the output file(s), the title of the document, and the Top node. A table of contents is also generally produced here.

This chapter expands on the minimal complete Texinfo source file previously given (see Section 1.10 [Six Parts], page 11). It describes the numerous commands for handling the traditional frontmatter items in Texinfo.

Straight text outside of any command before the Top node should be avoided. Such text is treated differently in the different output formats: visible in TEX and HTML, by default not shown in Info readers, and so on.

3.1 Sample Texinfo File Beginning

The following sample shows what is needed. The elements given here are explained in more detail in the following sections. Other commands are often included at the beginning of Texinfo files, but the ones here are the most critical.

See Section C.2 [GNU Sample Texts], page 217, for the full texts to be used in GNU manuals.

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

@copying
This manual is for program, version version.

Copyright @copyright{} years copyright-owner.

@quotation
Permission is granted to ...
@end quotation
@end copying

@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
@insertcopying
```

```
Published by ...
@end titlepage

@c So the toc is printed at the start.
@contents

@ifnottex
@node Top
@top title

@insertcopying
@end ifnottex

@menu
* First Chapter::    Getting started ...
* Second Chapter::          ...
 ...
* Copying::          Your rights and freedoms.
@end menu

@node First Chapter
@chapter First Chapter

@cindex first chapter
@cindex chapter, first
...
```

3.2 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.

Also, if you want to format just part of the Texinfo file, you must write the `@settitle` and `@setfilename` lines between start-of-header and end-of-header lines. The start- and end-of-header lines are optional, but they do no harm, so you might as well always include them.

Any command that affects document formatting as a whole makes sense to include in the header. `@synindex` (see Section 13.4.2 [synindex], page 111), for instance, is another command often included in the header. See Section C.2 [GNU Sample Texts], page 217, for complete sample texts.

Thus, the beginning of a Texinfo file generally looks like this:

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename sample.info
@settitle Sample Manual 1.0
@c %**end of header
```

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-*-
```

This 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 should have been installed on your system along with either the TEX or Texinfo software. TEX uses the backslash, '\', to mark the beginning of a command, exactly as Texinfo uses '`@`'. The '`texinfo.tex`' file causes the switch from '\' to '`@`'; before the switch occurs, TEX requires '\', which is why it appears at the beginning of the file.
2. When the file is edited in GNU Emacs, the '`-*-texinfo-*-`' mode specification tells Emacs to use Texinfo mode.

3.2.2 Start of Header

A start-of-header line is a Texinfo comment that looks like this:

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

Write the 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, with other commands that globally affect the document formatting, such as `@synindex` or `@footnotestyle`; and then by an end-of-header line (see Section 3.2.5 [End of Header], page 32).

The start- and end-of-header lines allow you to format only part of a Texinfo file for Info or printing. See Section 21.1.6 [texinfo-format commands], page 174.

The odd string of characters, '`%**`', is to ensure that no other comment is accidentally taken for a start-of-header line. You can change it if you wish by setting the `tex-start-of-header` and/or `tex-end-of-header` Emacs variables. See Section 20.6 [Texinfo Mode Printing], page 159.

3.2.3 `@setfilename`: Set the output file name

In order to serve as the primary input file for either `makeinfo` or TEX, 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 and follow it on the same line by the Info file name. Do not write anything else on the line; anything on the line after the command is considered part of the file name, including what would otherwise be a comment.

The Info formatting commands ignore everything written before the `@setfilename` line, which is why the very first line of the file (the `\input` line) does not show up in the output.

The `@setfilename` line specifies the name of the output file to be generated. This name must be different from the name of the Texinfo file. There are two conventions for choosing the name: you can either remove the extension (such as '`.texi`') entirely from the input file name, or, preferably, replace it with the '`.info`' extension.

Although an explicit '`.info`' extension is preferable, some operating systems cannot handle long file names. You can run into a problem even when the file name you specify is itself short enough. This occurs because the Info formatters split a long Info file into short indirect subfiles, and name them by appending '`-1`', '`-2`', ..., '`-10`', '`-11`', and so on, to the original file name. (See Section 21.1.8 [Tag and Split Files], page 175.) The subfile name '`texinfo.info-10`', for example, is too long for old systems with a 14-character limit on filenames; so the Info file name for this document is '`texinfo`' rather than '`texinfo.info`'. When `makeinfo` is running on operating systems such as MS-DOS which impose severe limits on file names, it may remove some characters from the original file name to leave enough space for the subfile suffix, thus producing files named '`texin-10`', '`gcc.i12`', etc.

When producing HTML output, `makeinfo` will replace any extension with '`html`', or add '`.html`' if the given name has no extension.

The `@setfilename` line produces no output when you typeset a manual with TEX, but it is nevertheless essential: it opens the index, cross-reference, and other auxiliary files used by Texinfo, and also reads '`texinfo.cnf`' if that file is present on your system (see Section 20.9 [Preparing for TEX], page 162).

3.2.4 `@settitle`: Set the document title

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 and follow it on the same line by the title. This tells TEX the title to use in a header or footer. Do not write anything else on the line; anything on the line after the command is considered part of the title, including what would otherwise be a comment.

The `@settitle` command should precede everything that generates actual output. The best place for it is right after the `@setfilename` command (see the previous section).

In the HTML file produced by `makeinfo`, *title* serves as the document '`<title>`'. It also becomes the default document description in the '`<head>`' part (see Section 3.7.1 [documentdescription], page 41).

The title in the `@settitle` command does not affect the title as it appears on the title page. Thus, the two do not need not match exactly. A practice we recommend is to include the version or edition number of the manual in the `@settitle` title; on the title page, the version number generally appears as a `@subtitle` so it would be omitted from the `@title`. See Section 3.4.1 [titlepage], page 34.

Conventionally, when TEX formats a Texinfo file for double-sided output, the title is printed in the left-hand (even-numbered) page headings and the current chapter title is printed in the right-hand (odd-numbered) page headings. (TEX learns the title of each chapter from each `@chapter` command.) By default, no page footer is 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.

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 37, for more information.)

You may, if you wish, create your own, customized headings and footings. See Appendix E [Headings], page 226, for a detailed discussion of this.

3.2.5 End of Header

Follow the header lines with an end-of-header line, which is a Texinfo comment that looks like this:

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

See Section 3.2.2 [Start of Header], page 30.

3.3 Document Permissions

The copyright notice and copying permissions for a document need to appear in several places in the various Texinfo output formats. Therefore, Texinfo provides a command (`@copying`) to declare this text once, and another command (`@insertcopying`) to insert the text at appropriate points.

3.3.1 `@copying`: Declare Copying Permissions

The `@copying` command should be given very early in the document; the recommended location is right after the header material (see Section 3.2 [Texinfo File Header], page 29). It conventionally consists of a sentence or two about what the program is, identification of the documentation itself, the legal copyright line, and the copying permissions. Here is a skeletal example:

```
@copying
This manual is for program (version version, updated
date), which ...

Copyright @copyright{} years copyright-owner.

@quotation
Permission is granted to ...
@end quotation
@end copying
```

The `@quotation` has no legal significance; it's there to improve readability in some contexts.

See Section C.2 [GNU Sample Texts], page 217, for the full text to be used in GNU manuals. See Section G.1 [GNU Free Documentation License], page 237, for the license itself under which GNU and other free manuals are distributed. You need to include the license as an appendix to your document.

The text of `@copying` is output as a comment at the beginning of Info, HTML, and XML output files. It is *not* output implicitly in plain text or TEX; it's up to you to use `@insertcopying` to emit the copying information. See the next section for details.

The `@copyright{}` command generates a '`c`' inside a circle in output formats that support this (print and HTML). In the other formats (Info and plain text), it generates '`(C)`'. The copyright notice itself has the following legally defined sequence:

```
Copyright © years copyright-owner.
```

The word 'Copyright' must always be written in English, even if the document is otherwise written in another language. This is due to international law.

The list of years should include all years in which a version was completed (even if it was released in a subsequent year). Ranges are not allowed; each year must be written out individually and in full, separated by commas.

The copyright owner (or owners) is whoever holds legal copyright on the work. In the case of works assigned to the FSF, the owner is 'Free Software Foundation, Inc.'.

The copyright 'line' may actually be split across multiple lines, both in the source document and in the output. This often happens for documents with a long history, having many different years of publication.

See section "Copyright Notices" in *GNU Maintenance Instructions*, for additional information.

3.3.2 `@insertcopying`: Include Permissions Text

The `@insertcopying` command is simply written on a line by itself, like this:

```
@insertcopying
```

This inserts the text previously defined by `@copying`. To meet legal requirements, it must be used on the copyright page in the printed manual (see Section 3.4.4 [Copyright], page 36).

We also strongly recommend using `@insertcopying` in the Top node of your manual (see Section 3.6 [The Top Node], page 39), although it is not required legally. Here's why:

The `@copying` command itself causes the permissions text to appear in an Info file *before* the first node. The text is also copied into the beginning of each split Info output file, as is legally necessary. This location implies a human reading the manual using Info does *not* see this text (except when using the advanced Info command `g *`). Therefore, an explicit `@insertcopying` in the Top node makes it apparent to readers that the manual is free.

Similarly, the `@copying` text is automatically included at the beginning of each HTML output file, as an HTML comment. Again, this text is not visible (unless the reader views the HTML source). And therefore again, the `@insertcopying` in the Top node is valuable because it makes the copying permissions visible and thus promotes freedom.

The permissions text defined by `@copying` also appears automatically at the beginning of the XML output file.

3.4 Title and Copyright Pages

In hard copy output, the manual's name and author are usually printed on a title page. Copyright information is usually 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 appears in the printed manual.

You may wish to include titlepage-like information for plain text output. Simply place any such leading material between `@ifplaintext` and `@end ifplaintext`; `makeinfo` includes this when writing plain text ('`--no-headers`'), along with an `@insertcopying`.

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 E [Page Headings], page 226, for details about how to generate page headings.) All the material that you want to appear on unnumbered pages should be put between the `@titlepage` and `@end titlepage` commands. You can force the table of contents to appear there with the `@setcontentsaftertitlepage` command (see Section 3.5 [Contents], page 38).

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!)

When you write a manual about a computer program, you should write the version of the program to which the manual applies on the title page. If the manual changes more frequently than the program or is independent of it, you should also include an edition number[1] for the manual. This helps readers keep track of which manual is for which version of the program. (The 'Top' node should also contain this information; see Section 3.6 [The Top Node], page 39.)

Texinfo provides two main 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 specify the text you want, and Texinfo does the formatting.

You may use either method, or you may combine them; see the examples in the sections below.

For extremely simple documents, and for the bastard title page in traditional book frontmatter, Texinfo also provides a command `@shorttitlepage` which takes the rest of the line as the title. The argument is typeset on a page by itself and followed by a blank page.

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. (This is the first of the two methods for creating a title page in Texinfo.)

Use the `@titlefont` command to select a large font suitable for the title itself. You can use `@titlefont` more than once if you have an especially long title.

For HTML output, each `@titlefont` command produces an `<h1>` heading, but the HTML document `<title>` is not affected. For that, you must put an `@settitle` command before the `@titlefont` command (see Section 3.2.4 [settitle], page 31).

[1] We have found that it is helpful to refer to versions of independent manuals as 'editions' and versions of programs as 'versions'; otherwise, we find we are liable to confuse each other in conversation by referring to both the documentation and the software with the same words.

For example:

```
@titlefont{Texinfo}
```

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

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

centers the title, which in this example is "Texinfo" 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 15.6 [@sp], page 125, for more information about the `@sp` command.)

A template for this method looks like this:

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

The spacing of the example fits an 8.5 by 11 inch manual.

You can in fact use these commands anywhere, not just on a title page, but since they are not logical markup commands, we don't recommend them.

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 automatically. 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. These commands are only effective in TEX output; it's an error to use them anywhere except within `@titlepage`.

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. Only a single line is allowed; the `@*` command may not be used to break the title into two lines. To handle very long titles, you may find it profitable to use both `@title` and `@titlefont`; see the final example in this section.

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.)

A template for this method looks like this:

```
@titlepage
@title name-of-manual-when-printed
@subtitle subtitle-if-any
@subtitle second-subtitle
@author author
@page
...
@end titlepage
```

You may also combine the `@titlefont` method described in the previous section and `@title` method described in this one. This may be useful if you have a very long title. Here is a real-life example:

```
@titlepage
@titlefont{GNU Software}
@sp 1
@title for MS-Windows and MS-DOS
@subtitle Edition @value{e} for Release @value{cde}
@author by Daniel Hagerty, Melissa Weisshaus
@author and Eli Zaretskii
```

(The use of `@value` here is explained in Section 17.4.3 [value Example], page 145.

3.4.4 Copyright Page

By international treaty, the copyright notice for a book must be either on the title page or on the back of the title page. 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, use the following incantantion after `@page`:

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

This is a TEX command that is not supported by the Info formatting commands. The `@vskip` command inserts whitespace. The '`0pt plus 1filll`' means to put in zero points of mandatory whitespace, and as much optional whitespace 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 correct.

To insert the copyright text itself, write `@insertcopying` next (see Section 3.3 [Document Permissions], page 32):

```
@insertcopying
```

Follow the copying text by the publisher, ISBN numbers, cover art credits, and other such information.

Here is an example putting all this together:

```
@titlepage
...
@page
@vskip 0pt plus 1filll
@insertcopying

Published by ...

Cover art by ...
@end titlepage
```

3.4.5 Heading Generation

Like all `@end` commands (see Chapter 10 [Quotations and Examples], page 85), the `@end titlepage` command must be written at the beginning of a line by itself, with only one space between the `@end` and the `titlepage`. It not only marks the end of the title and copyright pages, but also causes TEX to start generating page headings and page numbers.

To repeat what is said elsewhere, Texinfo has two standard page heading formats, one for documents which are printed on one side of each sheet of paper (single-sided printing), and the other for documents which are 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 page headings in the manner desired. (See Section 3.7.2 [setchapternewpage], page 41.)
- Alternatively, you can use the `@headings` command to prevent page headings from being generated or to start them for either single or double-sided printing. (Write an `@headings` command immediately after the `@end titlepage` command. See Section 3.4.6 [The `@headings` Command], page 37, for more information.)
- Or, you may specify your own page heading and footing format. See Appendix E [Page Headings], page 226, for detailed information about page 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 `@setchapternewpage` command. You need the `@headings` command only if the `@setchapternewpage` command does not do what you want, or if you want to turn off pre-defined page headings prior to defining your own. Write an `@headings` command immediately after the `@end titlepage` command.

You can use `@headings` as follows:

`@headings off`
: Turn off printing of page headings.

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

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

`@headings singleafter`
`@headings doubleafter`
: Turn on `single` or `double` headings, respectively, after the current page is output.

`@headings on`
: Turn on page headings: `single` if '`@setchapternewpage on`', `double` otherwise.

For example, suppose you write `@setchapternewpage off` before the `@titlepage` command to tell TEX to start a new chapter on the same page as the end of the last chapter. This command also causes TEX to typeset page headers for single-sided printing. To cause TEX to typeset for double sided printing, write `@headings double` after the `@end titlepage` command.

You can stop TEX from generating any page headings at all by writing `@headings off` on a line of its own immediately after the line containing the `@end titlepage` command, like this:

```
@end titlepage
@headings off
```

The `@headings off` command overrides the `@end titlepage` command, which would otherwise cause TEX to print page headings.

You can also specify your own style of page heading and footing. See Appendix E [Page Headings], page 226, for more information.

3.5 Generating a Table of Contents

The `@chapter`, `@section`, and other structuring commands (see Chapter 5 [Structuring], page 47) 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/or `@summarycontents` command(s).

`@contents`
: Generates a table of contents in a printed manual, including all chapters, sections, subsections, etc., as well as appendices and unnumbered chapters. Headings generated by `@majorheading`, `@chapheading`, and the other `@...heading` commands do not appear in the table of contents (see Section 5.2 [Structuring Command Types], page 47).

`@shortcontents`
`@summarycontents`

(`@summarycontents` is a synonym for `@shortcontents`.)

Generates a short or summary table of contents that lists only the chapters, appendices, and unnumbered chapters. Sections, subsections and subsubsections are omitted. Only a long manual needs a short table of contents in addition to the full table of contents.

Both contents commands should be written on a line by themselves, and are best placed near the beginning of the file, after the `@end titlepage` (see Section 3.4.1 [titlepage], page 34). The contents commands automatically generate a chapter-like heading at the top of the first table of contents page, so don't include any sectioning command such as `@unnumbered` before them.

Since an Info file uses menus instead of tables of contents, the Info formatting commands ignore the contents commands. But the contents are included in plain text output (generated by `makeinfo --no-headers`), unless `makeinfo` is writing its output to standard output.

When `makeinfo` writes a short table of contents while producing html output, the links in the short table of contents point to corresponding entries in the full table of contents rather than the text of the document. The links in the full table of contents point to the main text of the document.

In the past, the contents commands were sometimes placed at the end of the file, after any indices and just before the `@bye`, but we no longer recommend this.

However, since many existing Texinfo documents still do have the `@contents` at the end of the manual, if you are a user printing a manual, you may wish to force the contents to be printed after the title page. You can do this by specifying `@setcontentsaftertitlepage` and/or `@setshortcontentsaftertitlepage`. The first prints only the main contents after the `@end titlepage`; the second prints both the short contents and the main contents. In either case, any subsequent `@contents` or `@shortcontents` is ignored (unless, erroneously, no `@end titlepage` is ever encountered).

You need to include the `@set...contentsaftertitlepage` commands early in the document (just after `@setfilename`, for example). We recommend using `texi2dvi` (see Section 20.3 [Format with texi2dvi], page 157) to specify this without altering the source file at all. For example:

```
texi2dvi --texinfo=@setcontentsaftertitlepage foo.texi
```

3.6 The 'Top' Node and Master Menu

The 'Top' node is the node in which a reader enters an Info manual. As such, it should begin with the `@insertcopying` command (see Section 3.3 [Document Permissions], page 32) to provide a brief description of the manual (including the version number) and copying permissions, and end with a master menu for the whole manual. Of course you should include any other general information you feel a reader would find helpful.

It is also conventional to write an `@top` sectioning command line containing the title of the document immediately after the `@node Top` line (see Section 6.3.6 [The `@top` Sectioning Command], page 58).

The contents of the 'Top' node should appear only in the online output; none of it should appear in printed output, so enclose it between `@ifnottex` and `@end ifnottex` commands. (TEX does not print either an `@node` line or a menu; they appear only in Info; strictly speaking, you are not required to enclose these parts between `@ifnottex` and `@end ifnottext`, but it is simplest to do so. See Chapter 17 [Conditionally Visible Text], page 140.)

3.6.1 Top Node Example

Here is an example of a Top node.

```
@ifnottex
@node Top
@top Sample Title

@insertcopying

Additional general information.

@menu
* First Chapter::
* Second Chapter::
...
* Index::
@end menu
```

3.6.2 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. If you do use them, put `@detailmenu` before the first one, and `@end detailmenu` after the last; otherwise, `makeinfo` will get confused.

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 entry. (See Section 7.2 [Writing a Menu], page 60, for more information.)

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

```
@menu
* Copying Conditions::  Your rights.
* Overview::            Texinfo in brief.
...
* Command and Variable Index::
* Concept Index::

@detailmenu
--- The Detailed Node Listing ---

Overview of Texinfo

* Reporting Bugs:: ...
...

Beginning a Texinfo File

* Sample Beginning:: ...
...
@end detailmenu
@end menu
```

3.7 Global Document Commands

Besides the basic commands mentioned in the previous sections, here are additional commands which affect the document as a whole. They are generally all given before the Top node, if they are given at all.

3.7.1 `@documentdescription`: Summary Text

When producing HTML output for a document, `makeinfo` writes a '`<meta>`' element in the '`<head>`' to give some idea of the content of the document. By default, this *description* is the title of the document, taken from the `@settitle` command (see Section 3.2.4 [settitle], page 31). To change this, use the `@documentdescription` environment, as in:

```
@documentdescription
descriptive text.
@end documentdescription
```

This will produce the following output in the '`<head>`' of the HTML:

```
<meta name=description content="descriptive text.">
```

`@documentdescription` must be specified before the first node of the document.

3.7.2 `@setchapternewpage`:

In an officially bound book, text is usually 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 format headers 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 with the `@setchapternewpage` command:

`@setchapternewpage off`
: Cause TEX to typeset a new chapter on the same page as the last chapter, after skipping some vertical whitespace. Also, cause TEX to format page headers for single-sided printing.

`@setchapternewpage on`
: Cause TEX to start new chapters on new pages and to format page headers for single-sided printing. This is the form most often used for short reports or personal printing. This is the default.

`@setchapternewpage odd`
: Cause TEX to start new chapters on new, odd-numbered pages (right-handed pages) and to typeset for double-sided printing. This is the form most often used for books and manuals.

Texinfo does not have an `@setchapternewpage even` command, because there is no printing tradition of starting chapters or books on an even-numbered page.

If you don't like the default headers that `@setchapternewpage` sets, you can explicit control them with the `@headings` command. See Section 3.4.6 [The `@headings` Command], page 37.

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 and frontmatter 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.

We recommend not including any `@setchapternewpage` command in your manual sources at all, since the desired output is not intrinsic to the document. For a particular hard copy run, if you don't want the default option (no blank pages, same headers on all pages) use the '`--texinfo`' option to `texi2dvi` to specify the output you want.

3.7.3 `@paragraphindent`: Paragraph Indenting

The Texinfo processors may insert whitespace at the beginning of the first line of each paragraph, thereby indenting that paragraph. You can use the `@paragraphindent` command to specify this indentation. Write an `@paragraphindent` command at the beginning of a line followed by either '`asis`' or a number:

```
@paragraphindent indent
```

The indentation is according to the value of *indent*:

`asis` Do not change the existing indentation (not implemented in TEX).

`none`
0 Omit all indentation.

n Indent by *n* space characters in Info output, by *n* ems in TEX.

The default value of *indent* is 3. `@paragraphindent` is ignored for HTML output.

It is best to write the `@paragraphindent` command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See Section 3.2.2 [Start of Header], page 30.

A peculiarity of the `texinfo-format-buffer` and `texinfo-format-region` commands is that they do not indent (nor fill) paragraphs that contain `@w` or `@*` commands.

3.7.4 `@firstparagraphindent`: Indenting After Headings

As you can see in the present manual, the first paragraph in any section is not indented by default. Typographically, indentation is a paragraph separator, which means that it is unnecessary when a new section begins. This indentation is controlled with the `@firstparagraphindent` command:

```
@firstparagraphindent word
```

The first paragraph after a heading is indented according to the value of *word*:

`none` Prevents the first paragraph from being indented (default). This option is ignored by `makeinfo` if `@paragraphindent asis` is in effect.

`insert` Include normal paragraph indentation. This respects the paragraph indentation set by a `@paragraphindent` command (see Section 3.7.3 [paragraphindent], page 42).

For HTML and XML output, the `@firstparagraphindent` setting is ignored.

It is best to write the `@paragraphindent` command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See Section 3.2.2 [Start of Header], page 30.

3.7.5 `@exampleindent`: Environment Indenting

The Texinfo processors indent each line of `@example` and similar environments. You can use the `@exampleindent` command to specify this indentation. Write an `@exampleindent` command at the beginning of a line followed by either '`asis`' or a number:

```
@exampleindent indent
```

`@exampleindent` is ignored for HTML output. Otherwise, the indentation is according to the value of *indent*:

`asis` Do not change the existing indentation (not implemented in TEX).

0 Omit all indentation.

n Indent environments by *n* space characters in Info output, by *n* ems in TEX.

The default value of *indent* is 5 spaces in Info, and 0.4 in in TEX, which is somewhat less. (The reduction is to help TEX fit more characters onto physical lines.)

It is best to write the `@exampleindent` command before the end-of-header line at the beginning of a Texinfo file, so the region formatting commands indent paragraphs as specified. See Section 3.2.2 [Start of Header], page 30.

3.8 Software Copying Permissions

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, we recommend placing this right after 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.

4 Ending a Texinfo File

The end of a Texinfo file should include commands to create indices, and the `@bye` command to mark the last line to be processed.

For example:

```
@node Index
@unnumbered Index

@printindex cp

@bye
```

4.1 Printing Indices and Menus

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 20 [Hardcopy], page 156) to sort the raw data to produce a sorted index file. The sorted index file is what is actually used to print the index.

Texinfo offers six separate types of predefined index, which suffice in most cases. See Chapter 13 [Indices], page 108, for information on this, as well defining your own new indices, combining indices, and, most importantly advice on writing the actual index entries. This section focuses on printing indices, which is done with the `@printindex` command.

`@printindex` takes one argument, a two-letter index abbreviation. It reads the corresponding sorted index file (for printed output), and formats it appropriately into an index.

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

For example:

```
@node Variable Index
@unnumbered Variable Index

@printindex vr

@node Concept Index
@unnumbered Concept Index

@printindex cp
```

If you have more than one index, we recommend placing the concept index last.

- In printed output, `@printindex` produces a traditional two-column index, with dot leaders between the index terms and page numbers.

- In Info output, `@printindex` produces a special menu containing the line number of the entry, relative to the start of the node. Info readers can use this to go to the exact line of an entry, not just the containing node. (Older Info readers will just go to the node.) Here's an example:

  ```
  * First index entry:   Top.   (line  7)
  ```

  The actual number of spaces is variable, to right-justify the line number; it's been reduced here to make the line fit in the printed manual.
- In plain text output, `@printindex` produces the same menu, but the line numbers are relative to the start of the file, since that's more convenient for that format.
- In HTML and Docbook output, `@printindex` produces links to the index entries.
- In XML output, it simply records the index to be printed.

It's not possible to generate an index when writing to standard output; `makeinfo` generates a warning in this case.

4.2 `@bye` File Ending

An `@bye` command terminates Texinfo processing. None of the formatters read anything following `@bye`. The `@bye` command should be on a line by itself.

If you wish, you may follow the `@bye` line with notes. These notes will not be formatted and will not appear in either Info or a printed manual; it is as if text after `@bye` were within `@ignore ... @end ignore`. Also, you may follow the `@bye` line with a local variables list for Emacs. See Section 20.7 [Using Local Variables and the Compile Command], page 161, 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 3.5 [Generating a Table of Contents], page 38).

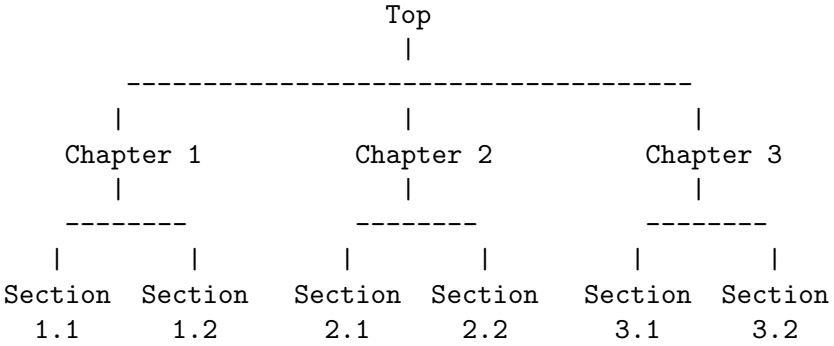
The chapter structuring commands do not create an Info node structure, so normally you should put an `@node` command immediately before each chapter structuring command (see Chapter 6 [Nodes], page 53). The only time you are likely to use the chapter structuring commands without using the node structuring 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 need 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.

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 following chapters. (See Chapter 6 [Nodes], page 53, and see Chapter 7 [Menus], page 60.)

5.2 Structuring Command Types

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 48). An `@unnumbered` section should be associated with a node and be a normal part of the document structure.
- The `@heading` series of commands produce simple unnumbered headings that do not appear in a table of contents, are not associated with nodes, and cannot be cross-referenced. The heading commands never start a new page.
- The `@majorheading` command is similar to `@chapheading`, except that it 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	*Unnumbered*	*Lettered/numbered*	No new page *Unnumbered*
In contents	In contents	In contents	Not in contents
	`@top`		`@majorheading`
`@chapter`	`@unnumbered`	`@appendix`	`@chapheading`
`@section`	`@unnumberedsec`	`@appendixsec`	`@heading`
`@subsection`	`@unnumberedsubsec`	`@appendixsubsec`	`@subheading`
`@subsubsection`	`@unnumberedsubsubsec`	`@appendixsubsubsec`	`@subsubheading`

5.3 `@top`

The `@top` command is a special sectioning command that you use only after an '`@node Top`' line at the beginning of a Texinfo file. The `@top` command tells the `makeinfo` formatter which node is the 'Top' node, so it can use it as the root of the node tree if your manual uses implicit node pointers. It has the same typesetting effect as `@unnumbered` (see Section 5.5 [`@unnumbered` and `@appendix`], page 49). For detailed information, see Section 6.3.6 [The `@top` Command], page 58.

The `@top` node and its menu (if any) is conventionally wrapped in an `@ifnottex` conditional so that it will appear only in Info and HTML output, not TEX.

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
*******************
```

Texinfo also provides a command `@centerchap`, which is analogous to `@unnumbered`, but centers its argument in the printed output. This kind of stylistic choice is not usually offered by Texinfo.

5.5 `@unnumbered` and `@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 48.)

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

In Info, the `@majorheading` and `@chapheading` 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 48.)

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 '`=`'.

This section is headed with an `@section` command and looks like this in the Texinfo file:

```
@section @code{@@section}
```

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 49.)

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

`@appendixsec`
`@appendixsection`
: `@appendixsection` is a longer spelling of the `@appendixsec` command; the two are synonymous.

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

`@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 49.) 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 50.)

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; both of these commands produce headings that appear in the table of contents.

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 50.) 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. They also appear in the table of contents. 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
.......................
```

5.12 `@raisesections` and `@lowersections`

The `@raisesections` and `@lowersections` commands implicitly raise and lower the hierarchical level of following chapters, sections and the other sectioning commands.

That is, the `@raisesections` command changes sections to chapters, subsections to sections, and so on. Conversely, the `@lowersections` command changes chapters to sections, sections to subsections, and so on. Thus, an `@lowersections` command cancels an `@raisesections` command, and vice versa.

You can use `@lowersections` to include text written as an outer or standalone Texinfo file in another Texinfo file as an inner, included file. Typical usage looks like this:

```
@lowersections
@include somefile.texi
@raisesections
```

(Without the `@raisesections`, all the subsequent sections in the document would be lowered.)

If the included file being lowered has a `@top` node, you'll need to conditionalize its inclusion with a flag (see Section 17.4.1 [set value], page 143).

Another difficulty can arise with documents that use the (recommended) feature of `makeinfo` for implicitly determining node pointers. Since `makeinfo` must assume a hierarchically organized document to determine the pointers, you cannot just arbitrarily sprinkle `@raisesections` and `@lowersections` commands in the document. The final result has to have menus that take the raising and lowering into account. Therefore, as a practical matter, you generally only want to raise or lower large chunks, usually in external files as shown above.

Repeated use of the commands continue to raise or lower the hierarchical level a step at a time. An attempt to raise above 'chapter' reproduces chapter commands; an attempt to lower below 'subsubsection' reproduces subsubsection commands. Also, lowered subsubsections and raised chapters will not work with `makeinfo`'s feature of implicitly determining node pointers, since the menu structure won't be correct.

Write each `@raisesections` and `@lowersections` command on a line of its own.

6 Nodes

Nodes are the primary segments of a Texinfo file. They do not in and of themselves impose a hierarchical or any other kind of structure on a file. Nodes contain *node pointers* that name other nodes, and can contain *menus* which are lists of nodes. In Info, the movement commands can carry you to a pointed-to node or to a node listed in a menu.

Node pointers and menus provide structure for Info files just as chapters, sections, subsections, and the like, provide structure for printed books.

Because node names are used in cross-references, it is not desirable to casually change them. Such name changes invalidate references from other manuals, from mail archives, and so on.

6.1 Two Paths

The node and menu commands and the chapter structuring commands are technically independent of each other:

- In Info, node and menu commands provide structure. The chapter structuring commands generate headings with different kinds of underlining—asterisks for chapters, hyphens for sections, and so on; they do nothing else.
- In TEX, the chapter structuring commands generate chapter and section numbers and tables of contents. The node and menu commands provide information for cross references; they do nothing else.

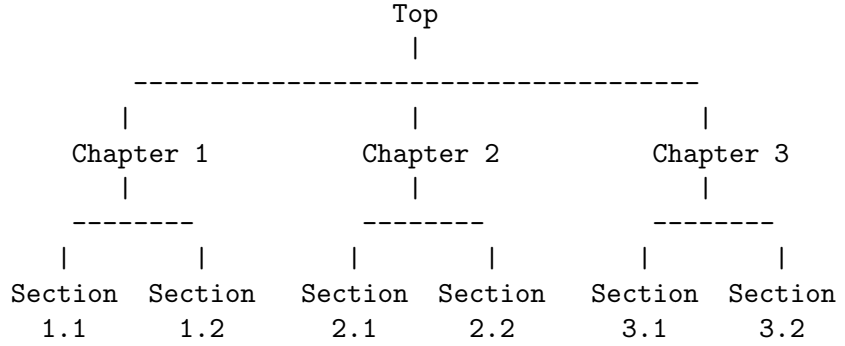
You can use node pointers and menus to structure an Info file any way you want; and you can write a Texinfo file so that its Info output has a different structure than its printed output. However, virtually all Texinfo files are written such that the structure for the Info output corresponds to the structure for the printed output. It is neither convenient nor understandable to the reader to do otherwise.

Generally, printed output is structured in a tree-like hierarchy in which the chapters are the major limbs from which the sections branch out. Similarly, node pointers and menus are organized to create a matching structure in the Info output.

6.2 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.

The “root” is at the top of the diagram and the “leaves” are at the bottom. This is how such a diagram is drawn conventionally; it illustrates an upside-down tree. For this reason, the root node is called the ‘Top’ node, and ‘Up’ node pointers carry you closer to the root.



The fully-written command to start Chapter 2 would be this:

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

This `@node` line says that the name of this node is "Chapter 2", the name of the 'Next' node is "Chapter 3", the name of the 'Previous' node is "Chapter 1", and the name of the 'Up' node is "Top". You can omit writing out these node names if your document is hierarchically organized (see Section 6.4 [makeinfo Pointer Creation], page 58), but the pointer relationships still obtain.

> **Note: Please Note:** 'Next' refers to the next node at the same hierarchical level in the manual, not necessarily to the next node within the Texinfo file. In the Texinfo file, the subsequent node may be at a lower level—a section-level node most often follows a chapter-level node, for example. 'Next' and 'Previous' refer to nodes at the *same* hierarchical level. (The 'Top' node contains the exception to this rule. Since the 'Top' node is the only node at that level, 'Next' refers to the first following node, which is almost always a chapter or chapter-level node.)

To go to Sections 2.1 and 2.2 using Info, you need a menu inside Chapter 2. (See Chapter 7 [Menus], page 60.) You would write the menu just before the beginning of Section 2.1, like this:

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

Write the node for Sect. 2.1 like this:

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

In Info format, the 'Next' and 'Previous' pointers of a node usually lead to other nodes at the same level—from chapter to chapter or from section to section (sometimes, as shown, the 'Previous' pointer points up); an 'Up' pointer usually leads to a node at the level above (closer to the 'Top' node); and a 'Menu' leads to nodes at a level below (closer to 'leaves'). (A cross reference can point to a node at any level; see Chapter 8 [Cross References], page 64.)

Usually, an `@node` command and a chapter structuring command are used in sequence, along with indexing commands. (You may follow the `@node` line with a comment line that reminds you which pointer is which.)

Here is the beginning of the chapter in this manual called "Ending a Texinfo File". This shows an `@node` line followed by a comment line, an `@chapter` line, and then by indexing lines.

```
@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
```

6.3 The `@node` Command

A *node* is a segment of text that begins at an `@node` command and continues until the next `@node` command. The definition of node is different from that for chapter or section. A chapter may contain sections and a section may contain subsections; but a node cannot contain subnodes; the text of a node continues only until the next `@node` command in the file. A node usually contains only one chapter structuring command, the one that follows the `@node` line. On the other hand, in printed output nodes are used only for cross references, so a chapter or section may contain any number of nodes. Indeed, a chapter usually contains several nodes, one for each section, subsection, and subsubsection.

To create a node, write an `@node` command at the beginning of a line, and follow it with up to four arguments, separated by commas, on the rest of the same line. The first argument is required; it is the name of this node. The subsequent arguments are the names of the 'Next', 'Previous', and 'Up' pointers, in that order, and may be omitted if your Texinfo document is hierarchically organized (see Section 6.4 [makeinfo Pointer Creation], page 58).

You may insert spaces before each name if you wish; the spaces are ignored. You must write the name of the node and (if present) the names of the 'Next', 'Previous', and 'Up' pointers all on the same line. Otherwise, the formatters fail. (See Info file '`info`', node '`Top`', for more information about nodes in Info.)

Usually, you write one of the chapter-structuring command lines immediately after an `@node` line—for example, an `@section` or `@subsection` line. (See Section 5.2 [Structuring Command Types], page 47.)

> **Note:** The GNU Emacs Texinfo mode updating commands work only with Texinfo files in which `@node` lines are followed by chapter structuring lines. See Section 2.5.2 [Updating Requirements], page 22.

TEX uses `@node` lines to identify the names to use for cross references. For this reason, you must write `@node` lines in a Texinfo file that you intend to format for printing, even if you do not intend to format it for Info. (Cross references, such as the one at the end of this sentence, are made with `@xref` and related commands; see Chapter 8 [Cross References], page 64.)

6.3.1 Choosing Node and Pointer Names

The name of a node identifies the node. The pointers enable you to reach other nodes and consist simply of the names of those nodes.

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

Usually, the first node of a Texinfo file is the 'Top' node, and its 'Up' and 'Previous' pointers point to the '`dir`' file, which contains the main menu for all of Info.

The 'Top' node itself contains the main or master menu for the manual. Also, it is helpful to include a brief description of the manual in the 'Top' node. See Section 6.3.5 [First Node], page 57, for information on how to write the first node of a Texinfo file.

Even when you explicitly specify all pointers, that does not mean you can write the nodes in the Texinfo source file in an arbitrary order! Because TEX processes the file sequentially, irrespective of node pointers, you must write the nodes in the order you wish them to appear in the printed output.

6.3.2 How to Write an `@node` Line

The easiest way to write an `@node` line is to write `@node` at the beginning of a line and then the name of the node, like this:

```
@node node-name
```

If you are using GNU Emacs, you can use the 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 let `makeinfo` insert node pointers into the Info file it creates. (See Chapter 2 [Texinfo Mode], page 16, and Section 6.4 [makeinfo Pointer Creation], page 58.)

Alternatively, you can 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 comment line is especially useful if you are not familiar with Texinfo.

The template for a fully-written-out node line with 'Next', 'Previous', and 'Up' pointers looks like this:

```
@node node-name, next, previous, up
```

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, we do not recommend this practice. It is better to name the node itself at the same time that you write a segment 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 an `@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.

6.3.3 `@node` Line Tips

Here are three suggestions:

- 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.)
- Try to pick node names that differ from each other near the beginnings of their names. This way, it is easy to use automatic name completion in Info.
- By convention, node names are capitalized just as they would be for section or chapter titles—initial and significant words are capitalized; others are not.

6.3.4 `@node` Line Requirements

Here are several requirements for `@node` lines:

- All the node names for a single Info file must be unique.

  Duplicates confuse the Info movement commands. This means, for example, that if you end every chapter with a summary, you must name each summary node differently. You cannot just call each one "Summary". You may, however, duplicate the titles of chapters, sections, and the like. Thus you can end each chapter in a book with a section called "Summary", so long as the node names for those sections are all different.
- A pointer name must be the name of a node.

  The node to which a pointer points may come before or after the node containing the pointer.
- `@`-commands in node names are not allowed. This includes punctuation characters that are escaped with a '`@`', such as `@` and `{`, and accent commands such as '`@'`'. (For a few cases when this is useful, Texinfo has limited support for using `@`-commands in node names; see Section 21.1.4 [Pointer Validation], page 172.) Perhaps this limitation will be removed some day.
- Unfortunately, you cannot use periods, commas, colons or parentheses within a node name; these confuse the Texinfo processors. Perhaps this limitation will be removed some day, too.

  For example, the following is a section title in this manual:

  ```
  @code{@@unnumberedsec}, @code{@@appendixsec}, @code{@@heading}
  ```

  But the corresponding node name lacks the commas and the `@`'s:

  ```
  unnumberedsec appendixsec heading
  ```
- Case is significant in node names.

  Spaces before and after names on the '`@node`' line are ignored, but spaces "inside" a name are significant. For example:

  ```
  @node  foo bar,
  @node foo bar ,
  @node  foo bar ,
  ```

  all define the same node, '`foo bar`'. References to the node should all use that name, without the leading or trailing spaces, but with the internal spaces.

6.3.5 The First Node

The first node of a Texinfo file is the *Top* node, except in an included file (see Appendix D [Include Files], page 222). The Top node should contain a short summary, copying permis-

sions, and a master menu. See Section 3.6 [The Top Node], page 39, for more information on the Top node contents and examples.

Here is a description of the node pointers to be used in the Top node:

- The Top node (which must be named '`top`' or '`Top`') should have as its 'Up' node the name of a node in another file, where there is a menu that leads to this file. Specify the file name in parentheses.

  Usually, all Info files are installed in the same Info directory tree; in this case, 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 the Info system as a whole.
- On the other hand, do not define the 'Previous' node of the Top node to be '`(dir)`', as it causes confusing behavior for users: if you are in the Top node and hits ⟨DEL⟩ to go backwards, you wind up in the middle of the some other entry in the '`dir`' file, which has nothing to do with what you were reading.
- The 'Next' node of the Top node should be the first chapter in your document.

See Section 21.2 [Installing an Info File], page 176, for more information about installing an Info file in the '`info`' directory.

For concreteness, here is an example with explicit pointers (which you can maintain automatically with the texinfo mode commands):

Or you can leave the pointers off entirely and let the tools implicitly define them. This is recommended. Thus:

```
@node Top
```

6.3.6 The `@top` Sectioning Command

A special sectioning command, `@top` should be used with the `@node Top` line. The `@top` sectioning 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` command 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` sectioning command causes the title to appear on a line by itself, with a line of asterisks inserted underneath, as other sectioning commands do.

In TEX and `texinfo-format-buffer`, the `@top` sectioning 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 `@unnumbered` after the `@node Top` line when you use these formatters. Also, you can use `@chapter` or `@unnumbered` when you use the Texinfo updating commands to create or update pointers and menus.

Thus, in practice, a Top node starts like this:

```
@node Top
@top Your Manual Title
```

6.4 Creating Pointers with `makeinfo`

The `makeinfo` program has a feature for automatically determining node pointers for a hierarchically organized document.

When you take advantage of this feature, you do not need 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 (except that comment lines may intervene).

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 48.

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 implicit node pointer insertion feature in `makeinfo` relieves you from the need to update menus and pointers manually or with Texinfo mode commands. (See Section 2.5 [Updating Nodes and Menus], page 19.)

In most cases, you will want to take advantage of this feature and not redundantly specify node pointers. However, Texinfo documents are not required to be organized hierarchically or in fact to contain sectioning commands at all (for example, if you never intend the document to be printed). The special procedure for handling the short text before a menu (see Chapter 7 [Menus], page 60) also disables this feature, for that group of nodes. In those cases, you will need to explicitly specify the pointers.

6.5 `@anchor`: Defining Arbitrary Cross-reference Targets

An *anchor* is a position in your document, labeled so that cross-references can refer to it, just as they can to nodes. You create an anchor with the `@anchor` command, and give the label as a normal brace-delimited argument. For example:

```
This marks the @anchor{x-spot}spot.
...
@xref{x-spot,,the spot}.
```

produces:

```
This marks the spot.
...
See [the spot], page 1.
```

As you can see, the `@anchor` command itself produces no output. This example defines an anchor 'x-spot' just before the word 'spot'. You can refer to it later with an `@xref` or other cross-reference command, as shown. See Chapter 8 [Cross References], page 64, for details on the cross-reference commands.

It is best to put `@anchor` commands just before the position you wish to refer to; that way, the reader's eye is led on to the correct text when they jump to the anchor. You can put the `@anchor` command on a line by itself if that helps readability of the source. Spaces are always ignored after `@anchor`.

Anchor names and node names may not conflict. Anchors and nodes are given similar treatment in some ways; for example, the `goto-node` command in standalone Info takes either an anchor name or a node name as an argument. (See section "goto-node" in *GNU Info*.)

7 Menus

Menus contain pointers to subordinate nodes. In online output, you use menus to go to such nodes. Menus have no effect in printed manuals and do not appear in them.

A node with a menu should not contain much text. If you find yourself writing a lot of before a menu, we generally recommend moving most of the text into a new subnode—all but a paragraph or two. Otherwise, a reader with a terminal that displays only a few lines may miss the menu and its associated text. As a practical matter, it is best to locate a menu within 20 or so lines of the beginning of the node.

7.1 Menu Location

A menu must be located at the end of a node, without any regular text or additional commands between the `@end menu` and the beginning of the next node. (As a consequence, there may be at most one menu in a node.)

This is actually a useful restriction, since a reader who uses the menu could easily miss any such text. Technically, it is necessary because in Info format, there is no marker for the end of a menu, so Info-reading programs would have no way to know when the menu ends and normal text resumes.

Technically, menus can carry you to any node, regardless of the structure of the document; even to nodes in a different Info file. However, we do not recommend ever making use of this, because the `makeinfo` implicit pointer creation feature (see Section 6.4 [makeinfo Pointer Creation], page 58) and GNU Emacs Texinfo mode updating commands work only to create menus of subordinate nodes in a hierarchically structured document. Instead, use cross references to refer to arbitrary nodes.

In the past, we recommended using a '`@heading`' command within an `@ifinfo` conditional instead of the normal sectioning commands after a very short node with a menu. This had the advantage of making the printed output look better, because there was no very short text between two headings on the page. But aside from not working with `makeinfo`'s implicit pointer creation, it also makes the XML output incorrect, since it does not reflect the true document structure. So, unfortunately we can no longer recommend this.

7.2 Writing a Menu

A menu consists of an `@menu` command on a line by itself followed by menu entry lines or menu comment lines and then by an `@end menu` command on a line by itself.

A menu looks like this:

```
@menu
Larger Units of Text

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

In a menu, every line that begins with an '`* `' is a *menu entry*. (Note the space after the asterisk.) A line that does not start with an '`* `' may also appear in a menu. Such

a line is not a menu entry but is a menu comment line that appears in the Info file. In the example above, the line '`Larger Units of Text`' is a menu comment line; the two lines starting with '`* `' are menu entries. Space characters in a menu are preserved as-is; this allows you to format the menu as you wish.

7.3 The Parts of a Menu

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

1. The menu entry name (optional).
2. The name of the node (required).
3. A description of the item (optional).

The template for a menu entry looks like this:

```
* menu-entry-name: node-name.   description
```

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

In Info, a user selects a node with the *m* (`Info-menu`) command. The menu entry name is what the user types after the *m* command.

The third part of a menu entry is a descriptive phrase or sentence. Menu entry names and node names are often short; the description explains to the reader what the node is about. A useful description complements the node name rather than repeats it. The description, which is optional, can spread over two or more lines; if it does, some authors prefer to indent the second line while others prefer to align it with the first (and all others). It's up to you.

7.4 Less Cluttered Menu Entry

When the menu entry name and 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::                                   description
```

instead of

```
* Name: Name.                              description
```

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

7.5 A Menu Example

A menu looks like this in Texinfo:

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

This produces:

```
* menu:

* menu entry 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 entry name and the name of the node referred to by that name. '`Multiples`' is the menu entry name; it refers to the node named '`Buffers`'. The line '`Larger Units of Text`' is a comment; it appears in the menu, but is not an entry.

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 7.6 [Referring to Other Info Files], page 62).

7.6 Referring to Other Info Files

You can create a menu entry that enables a reader in Info to go to a node in another Info file by writing the file name in parentheses just before the node name. In this case, you should use the three-part menu entry format, which saves the reader from having to type the file name.

The format looks like this:

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

For example, to refer directly to the '`Outlining`' and '`Rebinding`' nodes in the *Emacs Manual*, you would write a menu like this:

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

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 '`dir`' file that contains the main menu for Info has menu entries that list only file names. These take you directly to the 'Top' nodes of each Info document. (See Section 21.2 [Installing an Info File], page 176.)

For example:

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

(The '`dir`' top level directory for the Info system is an Info file, not a Texinfo file, but a menu entry looks the same in both types of file.)

The GNU Emacs Texinfo mode menu updating commands only work with nodes within the current buffer, so you cannot use them to create menus that refer to other files. You must write such menus by hand.

8 Cross References

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

8.1 What References Are For

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 online 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 results in a page reference, unless it is to another manual altogether, in which case the cross reference names that manual.

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

The various cross reference commands use nodes (or anchors, see Section 6.5 [`@anchor`], page 59) to define cross reference locations. This is evident in Info, in which a cross reference takes you to the specified location. TEX also uses nodes to define cross reference locations, but the action is less obvious. When TEX generates a DVI file, it records each node's page number 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 online, you must nonetheless write `@node` lines to name the places to which you make cross references.

8.2 Different Cross Reference Commands

There are four different cross reference commands:

`@xref` Used to start a sentence in the printed manual saying 'See ...' or an Info cross-reference saying '`*Note` *name*`:` *node*`.`'.

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

`@pxref` Used within parentheses to make a reference that suits both an Info file and a printed book. 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 9.1.13 [`@cite`], page 81.)

8.3 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 cross reference name for Info, a topic description or section title 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 is an example of a full five-part cross reference:

```
@xref{Node name, Cross Reference Name, Particular Topic,
info-file-name, A Printed Manual}, for details.
```

which produces

```
*Note Cross Reference Name: (info-file-name)Node name,
for details.
```

in Info and

> See section "Particular Topic" in *A Printed Manual*, for details.

in a printed book.

The five possible arguments for a cross reference are:

1. The node or anchor name (required). This is the location to which the cross reference takes you. In a printed document, the location of the node provides the page reference only for references within the same document.
2. The cross reference name for the Info reference, if it is to be different from the node name. If you include this argument, it becomes the first part of the cross reference. 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. You need not include any '`.info`' suffix on the file name, since Info readers try appending it automatically.
5. The name of a printed manual from a different Texinfo file.

The template for a full five argument cross reference looks like this:

```
@xref{node-name, cross-reference-name, title-or-topic,
info-file-name, printed-manual-title}.
```

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

Write a node name in a cross reference in exactly the same way as in the `@node` line, including the same capitalization; otherwise, the formatters may not find the reference.

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.

8.4 `@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.

8.4.1 What a Reference Looks Like and Requires

Most often, an Info cross reference looks like this:

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

or like this

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

In TEX, a cross reference looks like this:

> See Section *section-number* [*node-name*], page *page*.

or like this

> See Section *section-number* [*title-or-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 8.7 [`@pxref`], page 71.)

> **Caution:** 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 6.3.2 [Writing a Node], page 56).

`@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.

> **Note:** Commas separate arguments in a cross reference; avoid including them in the title or other part lest the formatters mistake them for separators.

8.4.2 `@xref` with One Argument

The simplest form of `@xref` takes one argument, the name of another node in the same Info file. The Info formatters produce output that the Info readers can use to jump to the reference; TEX produces output that specifies the page and section number for you.

For example,

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

produces

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

and

See Section 3.1 [Tropical Storms], page 24.

(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.
```

and

See Section 3.1 [Tropical Storms], page 24, for more info.

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

8.4.3 `@xref` with Two Arguments

With two arguments, the second is used as the name of the Info cross reference, while the first is still the name of the node to which the cross reference points.

The template is like this:

```
@xref{node-name, cross-reference-name}.
```

For example,

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

produces:

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

and

See Section 5.2 [Electrical Effects], page 57.

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

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

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

which produces

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

and

See Section 5.2 [Electrical Effects], page 57, for more info.

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

8.4.4 `@xref` with Three Arguments

A third argument replaces the node name in the TEX output. The third argument should be the name of the section in the printed output, or else state the topic discussed by that 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 to avoid placing a comma within the title or topic section of a cross reference, or within any other section. The formatters divide cross references into arguments according to the commas; a comma within a title or other section will divide it into two arguments. In a reference, you need to write a title such as "Clouds, Mist, and Fog" without the commas.

Also, remember to write a comma or period after the closing brace of an `@xref` to terminate the cross reference. In the following examples, a clause follows a terminating comma.

The template is like this:

```
@xref{node-name, cross-reference-name, title-or-topic}.
```

For example,

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

produces

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

and

> See Section 5.2 [Thunder and Lightning], page 57, 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 5.2 [Thunder and Lightning], page 57, for details.

As a practical matter, it is often best to write cross references with just the first argument if the node name and the section title are the same, and with the first and third arguments if the node name and title are different.

Here are several examples from *The GNU Awk User's Guide*:

```
@xref{Sample Program}.
@xref{Glossary}.
@xref{Case-sensitivity, ,Case-sensitivity in Matching}.
@xref{Close Output, , Closing Output Files and Pipes},
  for more information.
@xref{Regexp, , Regular Expressions as Patterns}.
```

8.4.5 `@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, cross-reference-name, title-or-topic,
info-file-name, printed-manual-title}.
```

For example,

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

produces

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

The name of 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" in *An Introduction to Meteorology*, for details.

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

The template looks like this:

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

which produces

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

and

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

For example,

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

produces

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

and

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

On rare occasions, you may want to refer to another Info file that 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.

8.5 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 7.6 [Referring to Other Info Files], page 62.) 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" in *The GNU Make Manual*.

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

8.6 `@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{Hurricanes}.
```

produces (in Info):

```
For more information, see *Note Hurricanes::.
```

and (in printed output):

> For more information, see Section 8.2 [Hurricanes], page 123.

The `@ref` command sometimes tempts 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}.
```

looks ok in the printed output:

> Sea surges are described in Section 6.7 [Hurricanes], page 72.

but is awkward to read in Info:

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

As a general rule, you should write a period or comma immediately after an `@ref` command with two or more arguments.

If there is no such following punctuation, `makeinfo` will generate a (grammatically incorrect) period in the Info output; otherwise, the cross-reference would fail completely, due to the current syntax of Info format.

8.7 @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 (or anything else) 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 (or any other text), as you do with `@xref`. In the Info output, such text would follow a period, which is grammatically wrong.

`@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.

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

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

which produces

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

and

... storms cause flooding (see Section 6.7 [Hurricanes], page 72) ...

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

```
... (@pxref{node-name, cross-reference-name}) ...
```

which produces

```
... (*Note cross-reference-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 8.4 [`@xref`], page 66).

> **Caution:** Use `@pxref` only as a parenthetical reference. Do not try to use `@pxref` as a clause in a sentence. It will look bad in either the Info file, the printed output, or both.

Parenthetical cross references look best at the ends of sentences. Although they technically work in the middle of a sentence, that location breaks up the flow of reading.

8.8 @inforef

`@inforef` is used for making cross references to Info documents—even from a printed manual. This might be because you want to refer to conditional `@ifinfo` text (see Chapter 17 [Conditionals], page 140), or because printed output is not available (perhaps because there is no Texinfo source), among other possibilities.

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

1. The node name.
2. The cross reference 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, cross-reference-name, info-file-name},
```

For example,

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

produces (in Info):

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

and (in the printed output):

> See Info file '`info`', node '`Advanced`', for more information.

(This particular example is not realistic, since the Info manual is written in Texinfo, so all formats are available.)

The converse of `@inforef` is `@cite`, which is used to refer to printed works for which no Info form exists. See Section 9.1.13 [`@cite`], page 81.

8.9 @url, @uref{*url*[, *text*][, *replacement*]}

`@uref` produces a reference to a uniform resource locator (url). It takes one mandatory argument, the url, and two optional arguments which control the text that is displayed. In HTML output, `@uref` produces a link you can follow.

`@url` is a synonym for `@uref`. Originally, `@url` had the meaning of `@indicateurl` (see Section 9.1.16 [`@indicateurl`], page 82), but in actual practice it was misused the vast majority of the time. So we've changed the definitions.

The second argument, if specified, is the text to display (the default is the url itself); in Info and DVI output, but not in HTML output, the url is also output.

The third argument, if specified, is the text to display, but in this case the url is *not* output in any format. This is useful when the text is already sufficiently referential, as in a man page. If the third argument is given, the second argument is ignored.

If the url is long enough to cause problems with line breaking, you may find it useful to insert `@/` at places where a line break would be acceptable (after '`/`' characters, for instance). This tells TEX to allow (but not force) a line break at those places. See Section 15.2 [Line Breaks], page 123.

Here is an example of the simple one argument form, where the url is both the target and the text of the link:

```
The official GNU ftp site is @uref{ftp://ftp.gnu.org/gnu}.
```

produces:

The official GNU ftp site is ftp://ftp.gnu.org/gnu.

An example of the two-argument form:

```
The official @uref{ftp://ftp.gnu.org/gnu, GNU ftp site}
holds programs and texts.
```

produces:

The official GNU ftp site (ftp://ftp.gnu.org/gnu)
holds programs and texts.

that is, the Info output is this:

```
The official GNU ftp site (ftp://ftp.gnu.org/gnu)
holds programs and texts.
```

and the HTML output is this:

```
The official <a href="ftp://ftp.gnu.org/gnu">GNU ftp site</a>
holds programs and texts.
```

An example of the three-argument form:

```
The @uref{/man.cgi/1/ls,,ls(1)} program ...
```

produces:

The ls(1) program ...

but with HTML:

```
The <a href="/man.cgi/1/ls">ls(1)</a> program ...
```

To merely indicate a url without creating a link people can follow, use `@indicateurl` (see Section 9.1.16 [indicateurl], page 82).

Some people prefer to display url's in the unambiguous format:

<URL:http://*host*/*path*>

You can use this form in the input file if you wish. We feel it's not necessary to include the '`<URL:`' and '`>`' in the output, since any software that tries to detect url's in text already has to detect them without the '`<URL:`' to be useful.

9 Marking Words and Phrases

In Texinfo, you can mark words and phrases in a variety of ways. The Texinfo formatters use this information to determine how to highlight the text. You can 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, in several different ways.

9.1 Indicating Definitions, Commands, etc.

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

For example, in a printed manual, code is usually illustrated in a typewriter font; `@code` tells TEX to typeset this text in this font. But it would be easy to change the way TEX highlights code to use another font, and this change would not affect 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 need to check every single occurrence to make sure that you were changing code and not something else that should not be changed.

9.1.1 Highlighting Commands are Useful

The highlighting commands can be used to extract 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. See Section 9.1.2 [`@code`], page 75.

`@kbd{`*`keyboard-characters`*`}`
: Indicate keyboard input. See Section 9.1.3 [`@kbd`], page 76.

`@key{`*`key-name`*`}`
: Indicate the conventional name for a key on a keyboard. See Section 9.1.4 [`@key`], page 77.

`@samp{`*`text`*`}`
: Indicate text that is a literal example of a sequence of characters. See Section 9.1.5 [`@samp`], page 78.

`@verb{`*`text`*`}`
: Write a verbatim sequence of characters. See Section 9.1.6 [`@verb`], page 78.

`@var{`*`metasyntactic-variable`*`}`
: Indicate a metasyntactic variable. See Section 9.1.7 [`@var`], page 79.

`@env{`*`environment-variable`*`}`
: Indicate an environment variable. See Section 9.1.8 [`@kenv`], page 79.

`@file{`*`file-name`*`}`
: Indicate the name of a file. See Section 9.1.9 [`@file`], page 80.

`@command{`*`command-name`*`}`
: Indicate the name of a command. See Section 9.1.10 [`@command`], page 80.

`@option{`*`option`*`}`
: Indicate a command-line option. See Section 9.1.11 [`@option`], page 80.

`@dfn{`*`term`*`}`
: Indicate the introductory or defining use of a term. See Section 9.1.12 [`@dfn`], page 80.

`@cite{`*`reference`*`}`
: Indicate the name of a book. See Section 9.1.13 [`@cite`], page 81.

`@abbr{`*`abbreviation`*`}`
: Indicate an abbreviation.

`@acronym{`*`acronym`*`}`
: Indicate an acronym. See Section 9.1.15 [`@acronym`], page 81.

`@indicateurl{`*`uniform-resource-locator`*`}`
: Indicate an example (that is, nonfunctional) uniform resource locator. See Section 9.1.16 [`@indicateurl`], page 82. (Use `@url` (see Section 8.9 [`@url`], page 72) for live url's.)

`@email{`*`email-address`*`[, `*`displayed-text`*`]}`
: Indicate an electronic mail address. See Section 9.1.17 [`@email`], page 82.

9.1.2 `@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 in a programming language.

Use `@code` for command names in languages that resemble programming languages, such as Texinfo. For example, `@code` and `@samp` are produced by writing '`@code{@@code}`' and '`@code{@@samp}`' in the Texinfo source, respectively.

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 different from the identifier `printf`, and most likely is a misspelling of it. Even in languages which are not case sensitive, it is confusing to a human reader to see identifiers spelled in different ways. Pick one spelling and always use that. If you do not want to start a sentence with a command name 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 single quotation marks around the text.

For example,

```
The function returns @code{nil}.
```

produces this in the printed manual:

The function returns `nil`.

and this in the Info file:

```
The function returns ‘nil’.
```

Here are some cases for which it is preferable not to use `@code`:

- For shell command names such as `ls` (use `@command`).
- For shell options such as ‘`-c`’ when such options stand alone (use `@option`).
- Also, an entire shell command often looks better if written using `@samp` rather than `@code`. In this case, the rule is to choose the more pleasing format.
- For environment variable such as `TEXINPUTS` (use `@env`).
- For a string of characters shorter than a syntactic token. For example, if you are writing about ‘`goto-ch`’, which is just a part of the name for the `goto-char` Emacs Lisp function, you should use `@samp`.
- In general, when writing about the characters used in a token; for example, do not 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 keystroke commands of GNU Emacs (use `@kbd` instead) although you may use `@code` for the names of the Emacs Lisp functions that the keystroke commands invoke.

Since `@command`, `@option`, and `@env` were introduced relatively recently, it is acceptable to use `@code` or `@samp` for command names, options, and environment variables. The new commands allow you to express the markup more precisely, but there is no real harm in using the older commands, and of course the long-standing manuals do so.

9.1.3 @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}
```

By default, the `@kbd` command produces a different font (slanted typewriter instead of normal typewriter) in the printed manual, so users can distinguish the characters that they are supposed to type from those that the computer outputs.

In Info output, `@kbd` is usually the same as `@code`, producing ‘quotes’ around its argument. However, in typewriter-like contexts such as the `@example` environment (see Section 10.3 [example], page 86) and `@code` command itself, the quotes are omitted, since Info format cannot use distinguishing fonts.

Since the usage of `@kbd` varies from manual to manual, you can control the font switching with the `@kbdinputstyle` command. This command has no effect on Info output.

Write this command at the beginning of a line with a single word as an argument, one of the following:

'`code`' Always use the same font for `@kbd` as `@code`.

'`example`' Use the distinguishing font for `@kbd` only in `@example` and similar environments.

'`distinct`'
(the default) Always use the distinguishing font for `@kbd`.

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 the '`r`' key and then press the ⟨RETURN⟩ key":

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

This produces: *r* ⟨RET⟩. (The present manual accepts the default for `@kbdinputstyle`.)

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 explicitly want to mention a space character as one of the characters of input, write `@key{SPC}` for it.)

9.1.4 `@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}}
```

Here is a list of the recommended names for keys:

`SPC`	Space
`RET`	Return
`LFD`	Linefeed (however, since most keyboards nowadays do not have a Linefeed key, it might be better to call this character *C-j*.
`TAB`	Tab
`BS`	Backspace
`ESC`	Escape
`DEL`	Delete
`SHIFT`	Shift
`CTRL`	Control
`META`	Meta

There are subtleties to handling words like 'meta' or 'ctrl' that are names of modifier keys. When mentioning a character in which the modifier key is used, such as *Meta-a*, use the `@kbd` command alone; do not use the `@key` command; but when you are referring to the modifier key in isolation, use the `@key` command. For example, write '`@kbd{Meta-a}`' to produce *Meta-a* and '`@key{META}`' to produce ⟨META⟩.

9.1.5 `@samp{`*text*`}`

Use the `@samp` command to indicate text that is a literal example or 'sample' of a sequence of characters in a file, string, pattern, etc. Enclose the text in braces. The argument appears within single quotation marks 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` or `@key` is more appropriate. Also, you may use `@samp` for entire statements in C and for entire shell commands—in this case, `@samp` often looks better than `@code`. 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`'.

9.1.6 `@verb{<char>`*text*`<char>}`

Use the `@verb` command to print a verbatim sequence of characters.

Like LaTeX's `\verb` command, the verbatim text can be quoted using any unique delimiter character. Enclose the verbatim text, including the delimiters, in braces. Text is printed in a fixed-width font:

```
How many @verb{|@|}-escapes does one need to print this
@verb{.@a @b @c.} string or @verb{+@'e?`!`{}\+} this?
```

produces

```
How many @-escapes does one need to print this
@a @b @c string or these @'e?`{}!`\ this?
```

This is in contrast to `@samp` (see the previous section), `@code`, and similar commands; in those cases, the argument is normal Texinfo text, where the three characters `@{}` are special. With `@verb`, nothing is special except the delimiter character you choose.

It is not reliable to use `@verb` inside other Texinfo constructs. In particular, it does not work to use `@verb` in anything related to cross-referencing, such as section titles or figure captions.

9.1.7 `@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 (see Section 9.1.2 [code], page 75). For example, the Emacs Lisp variable `texinfo-tex-command` is not a metasyntactic variable; it is properly formatted using `@code`.

Do not use `@var` for environment variables either; `@env` is correct for them (see the next section).

The effect of `@var` in the Info file is to change the case of the argument to all upper case. In the printed manual and HTML output, the argument is printed in slanted type.

For example,

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

produces

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

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

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

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

This produces:

```
\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 that Texinfo uses. (You can, of course, modify the sources to '`texinfo.tex`' and the Info formatting commands to output the `<...>` format if you wish.)

9.1.8 `@env{`*environment-variable*`}`

Use the `@env` command to indicate environment variables, as used by many operating systems, including GNU. Do not use it for metasyntactic variables; use `@var` instead (see the previous section).

`@env` is equivalent to `@code` in its effects. For example:

```
The @env{PATH} environment variable ...
```

produces

The PATH environment variable ...

9.1.9 @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 file name suffixes. Do not use @file for symbols in a programming language; use @code.

Currently, @file is equivalent to @samp in its effects. 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.

9.1.10 @command{*command-name*}

Use the @command command to indicate command names, such as ls or cc.

@command is equivalent to @code in its effects. For example:

```
The command @command{ls} lists directory contents.
```

produces

The command ls lists directory contents.

You should write the name of a program in the ordinary text font, rather than using @command, if you regard it as a new English word, such as 'Emacs' or 'Bison'.

When writing an entire shell command invocation, as in 'ls -l', you should use either @samp or @code at your discretion.

9.1.11 @option{*option-name*}

Use the @option command to indicate a command-line option; for example, '-l' or '--version' or '--output=*filename*'.

@option is equivalent to @samp in its effects. For example:

```
The option @option{-l} produces a long listing.
```

produces

The option '-l' produces a long listing.

In tables, putting options inside @code produces a more pleasing effect.

9.1.12 @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 does not 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 need to say explicitly that it is a definition, but it should contain the information of a definition—it should make the meaning clear.

9.1.13 `@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 a reader can easily follow such a reference in Info. See Section 8.4 [`@xref`], page 66.

9.1.14 `@abbr{`*abbreviation*`[, `*meaning*`]}`

You can use the `@abbr` command for general abbreviations. The abbreviation is given as the single argument in braces, as in '`@abbr{Comput.}`'. As a matter of style, or for particular abbreviations, you may prefer to omit periods, as in '`@abbr{Mr} Stallman`'.

`@abbr` accepts an optional second argument, intended to be used for the meaning of the abbreviation.

If the abbreviation ends with a lowercase letter and a period, and is not at the end of a sentence, and has no second argument, remember to use the `@.` command (see Section 14.2.1 [Not Ending a Sentence], page 114) to get the correct spacing. However, you do not have to use `@.` within the abbreviation itself; Texinfo automatically assumes periods within the abbreivation do not end a sentence.

In TEX and in the Info output, the first argument is printed as-is; if the second argument is present, it is printed in parentheses after the abbreviation. In HTML and XML, the `<abbr>` tag is used; in Docbook, the `<abbrev>` tag is used. For instance:

```
@abbr{Comput. J., Computer Journal}
```

produces:

Comput. J. (Computer Journal)

For abbreviations consisting of all capital letters, you may prefer to use the `@acronym` command instead. See the next section for more on the usage of these two commands.

9.1.15 `@acronym{`*acronym*`[, `*meaning*`]}`

Use the `@acronym` command for abbreviations written in all capital letters, such as 'NASA'. The abbreviation is given as the single argument in braces, as in '`@acronym{NASA}`'. As a matter of style, or for particular acronyms, you may prefer to use periods, as in '`@acronym{N.A.S.A.}`'.

`@acronym` accepts an optional second argument, intended to be used for the meaning of the acronym.

If the acronym is at the end of a sentence, and if there is no second argument, remember to use the `@.` or similar command (see Section 14.2.2 [Ending a Sentence], page 114) to get the correct spacing.

In TEX, the acronym is printed in slightly smaller font. In the Info output, the argument is printed as-is. In either format, if the second argument is present, it is printed in parentheses after the acronym. In HTML, Docbook, and XML, the `<acronym>` tag is used.

For instance (since GNU is a recursive acronym, we use `@acronym` recursively):

```
@acronym{GNU, @acronym{GNU}'s Not Unix}
```

produces:

GNU (GNU's Not Unix)

In some circumstances, it is conventional to print family names in all capitals. Don't use `@acronym` for this, since a name is not an acronym. Use `@sc` instead (see Section 9.2.2 [Smallcaps], page 83).

`@abbr` and `@acronym` are closely related commands: they both signal to the reader that a shortened form is being used, and possibly give a meaning. When choosing whether to use these two commands, please bear the following in mind.

- In standard English usage, acronyms are a subset of abbreviations: they include pronounceable words like 'NATO', 'radar', and 'snafu', and some sources also include syllable acronyms like 'Usenet', hybrids like 'SIGGRAPH', and unpronounceable initialisms like 'FBI'.
- In Texinfo, an acronym (but not an abbreviation) should consist only of capital letters and periods, no lowercase.
- In TEX, an acronym (but not an abbreviation) is printed in a slightly smaller font.
- Some browsers place a dotted bottom border under abbreviations but not acronyms.
- It's not essential to use these commands for all abbreviations. Text is perfectly readable without them, and for common abbreviations like 'etc.', we consider them to be overkill.

9.1.16 `@indicateurl{`*uniform-resource-locator*`}`

Use the `@indicateurl` command to indicate a uniform resource locator on the World Wide Web. This is analogous to `@file`, `@var`, etc., and is purely for markup purposes. It does not produce a link you can follow in HTML output (use the `@uref` command for that, see Section 8.9 [`@uref`], page 72). It is useful for url's which do not actually exist. For example:

```
For example, the url might be @indicateurl{http://example.org/path}.
```

which produces:

For example, the url might be `http://example.org/path`.

9.1.17 `@email{`*email-address*`[, `*displayed-text*`]}`

Use the `@email` command to indicate an electronic mail address. It takes one mandatory argument, the address, and one optional argument, the text to display (the default is the address itself).

In Info, the address is shown in angle brackets, preceded by the text to display if any. In TEX, the angle brackets are omitted. In HTML output, `@email` produces a '`mailto`' link that usually brings up a mail composition window. For example:

```
Send bug reports to @email{bug-texinfo@@gnu.org},
suggestions to the @email{bug-texinfo@@gnu.org, same place}.
```

produces

Send bug reports to `bug-texinfo@gnu.org`,
suggestions to the same place (`bug-texinfo@gnu.org`).

9.2 Emphasizing Text

Usually, Texinfo changes the font to mark words in the text according to what category the words belong to; an example is the `@code` command. 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 commands to do this. Also, Texinfo has several commands that specify the font in which TEX will typeset text. These commands have no effect on Info and only one of them, the `@r` command, has any regular use.

9.2.1 `@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,

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

produces the following in printed output and HTML:

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.

In the Info output, `@emph` surrounds the text with underscores ('`_`'), and `@strong` puts asterisks around the text.

> **Caution:** Do not use `@strong` with the word '`Note`'; Info will mistake the combination for a cross reference. (It's usually redundant, anyway.) Use a phrase such as **Please notice** or **Caution** instead, or the optional argument to `@quotation`—'`Note`' is allowable there.

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

Use the '`@sc`' command to set text in A SMALL CAPS FONT (where possible). Write the text you want to be in small caps between braces in lower case, like this:

```
Richard @sc{Stallman} founded @acronym{GNU}.
```

This produces:

Richard STALLMAN founded GNU.

As shown here, we recommend using `@acronym` for actual acronyms (see Section 9.1.15 [acronym], page 81), and reserving `@sc` for special cases where you want small caps. The output is not the same (`@acronym` prints in a smaller text font, not the small caps font), but more importantly it describes the actual text more accurately.

Family names are one case where small capitals are sometimes desirable, also as shown here.

TEX typesets any uppercase letters between the braces of an `@sc` command in full-size capitals; only lowercase letters are printed in the small caps font. In the Info output, the argument to `@sc` is printed in all upper case. In HTML, the argument is uppercased and the output marked with the `<small>` tag to reduce the font size.

Since it's redundant to mark all-uppercase text with `@sc`, `makeinfo` warns about such usage.

We recommend using regular mixed case wherever possible.

9.2.3 Fonts for Printing, Not Info

Texinfo provides a number of font commands that specify font changes in the printed manual and (where possible) in the HTML output, but have no effect in the Info file. All the commands apply to an argument that follows, surrounded by braces.

`@b` selects **bold** face;

`@i` selects an *italic* font;

`@r` selects a roman font, which is the usual font in which text is printed. It may or may not be seriffed.

`@sansserif`
selects a sans serif font;

`@slanted` rselects a *slanted* font;

`@t` selects the `fixed-width`, typewriter-style font used by `@code`;

(The commands with longer names were invented much later than the others, when it did not seem desirable to use very short names for such an infrequently needed feature.)

Only the `@r` command has much use: in example-like environments, you can use the `@r` command to write comments in the standard roman font instead of the fixed-width font. This looks better in printed output, and produces a `<lineannotation>` tag in Docbook output.

For example,

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

produces

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

In general, you should avoid using the other font commands. Some of them are only useful when documenting functionality with specific font effects, such as in \TeX\ and related packages.

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 the output.

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 itself, at the beginning of that line, and with only one space between the `@end` and the `example`.

10.1 Block Enclosing Commands

Here are commands for quotations and examples, explained further in the following sections:

`@quotation`
: Indicate text that is quoted. The text is filled, indented (from both margins), and printed in a roman font by default.

`@example`
: Illustrate code, commands, and the like. The text is printed in a fixed-width font, and indented but not filled.

`@verbatim`
: Mark a piece of text that is to be printed verbatim; no character substitutions are made and all commands are ignored, until the next `@end verbatim`. The text is printed in a fixed-width font, and not indented or filled. Extra spaces and blank lines are significant, and tabs are expanded.

`@smallexample`
: Same as `@example`, except that in TEX this command typesets text in a smaller font.

`@lisp`
: Like `@example`, but specifically for illustrating Lisp code. The text is printed in a fixed-width font, and indented but not filled.

`@smalllisp`
: Is to `@lisp` as `@smallexample` is to `@example`.

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

`@smalldisplay`
: Is to `@display` as `@smallexample` is to `@example`.

`@format`
: Like `@display` (the text is not filled and no font is selected), but the text is not indented.

`@smallformat`
: Is to `@format` as `@smallexample` is to `@example`.

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.

You can use the `@cartouche` environment around one of the above constructs to highlight the example or quotation by drawing a box with rounded corners around it. See Section 10.14 [Drawing Cartouches Around Examples], page 92.

10.2 `@quotation`: Block quotations

The text of a quotation is processed normally (regular font, text is filled) except that:

- the margins are closer to the center of the page, so the whole of the quotation is indented;
- and the first lines of paragraphs are indented no more than other lines.

> This is an example of text written between an `@quotation` command and an `@end quotation` command. An `@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 on 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.

`@quotation` takes one optional argument, given on the remainder of the line. This text, if present, is included at the beginning of the quotation in bold or otherwise emphasized, and followed with a ':'. For example:

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

produces

Note: This is a foo.

If the `@quotation` argument is exactly one of these words:

```
Caution  Important  Note  Tip  Warning
```

then the Docbook output uses corresponding special tags (`<note>`, etc.) instead of the default `<blockquote>`. HTML output always uses `<blockquote>`.

10.3 `@example`: Example Text

The `@example` environment is used to indicate an example that is not part of the running text, such as computer input or output. Write an `@example` command at the beginning of a line by itself. Mark the end of the example with an `@end example` command, also written at the beginning of a line by itself.

An `@example` environment has the following characteristics:

- Each line in the input file is a line in the output; that is, the source text is not filled as it normally is.
- Extra spaces and blank lines are significant.

- The output is indented.
- The output uses a fixed-width font.
- Texinfo commands *are* expanded; if you want the output to be the input verbatim, use the `@verbatim` environment instead (see Section 10.4 [`@verbatim`], page 87).

For example,

```
@example
cp foo @var{dest1}; \
 cp foo @var{dest2}
@end example
```

produces

```
cp foo dest1; \
 cp foo dest2
```

The lines containing `@example` and `@end example` will disappear from the output. To make the output look good, you should put a blank line before the `@example` and another blank line after the `@end example`. Blank lines inside the beginning `@example` and the ending `@end example`, on the other hand, do appear in the output.

> **Caution:** Do not use tabs in the lines of an example! (Or anywhere else in Texinfo, except in verbatim environments.) TEX treats tabs as single spaces, and that is not what they look like. In Emacs, you can use *M-x untabify* to convert tabs in a region to multiple spaces.

Examples are often, logically speaking, "in the middle" of a paragraph, and the text that continues afterwards should not be indented, as in the example above. The `@noindent` command prevents a piece of text from being indented as if it were a new paragraph (see Section 10.12 [`@noindent`], page 91.

If you want to embed code fragments within sentences, instead of displaying them, use the `@code` command or its relatives (see Section 9.1.2 [`@code`], page 75).

If you wish to write a "comment" on a line of an example in the normal roman font, you can use the `@r` command (see Section 9.2.3 [Fonts], page 84).

10.4 `@verbatim`: Literal Text

Use the `@verbatim` environment for printing of text that may contain special characters or commands that should not be interpreted, such as computer input or output (`@example` interprets its text as regular Texinfo commands). This is especially useful for including automatically generated output in a Texinfo manual. Here is an example; the output you see is just the same as the input, with a line `@verbatim` before and a line `@end verbatim` after.

```
This is an example of text written in a @verbatim
block.  No character substitutions are made.  All commands
are ignored, until '<at>end verbatim'.

In the printed manual, the text is typeset in a
fixed-width font, and not indented or filled.  All
spaces and blank lines are significant, including tabs.
```

Write a `@verbatim` command at the beginning of a line by itself. This line will disappear from the output. Mark the end of the verbatim block with a `@end verbatim` command, also written at the beginning of a line by itself. The `@end verbatim` will also disappear from the output.

For example:

```
@verbatim
{
⟨TAB⟩@command with strange characters: @'e
expand⟨TAB⟩me
}
@end verbatim
```

produces

```
{
        @command with strange characters: @'e
expand  me
}
```

Since the lines containing `@verbatim` and `@end verbatim` produce no output, typically you should put a blank line before the `@verbatim` and another blank line after the `@end verbatim`. Blank lines between the beginning `@verbatim` and the ending `@end verbatim` will appear in the output.

It is not reliable to use `@verbatim` inside other Texinfo constructs.

10.5 `@verbatiminclude` *file*: Include a File Verbatim

You can include the exact contents of a file in the document with the `@verbatiminclude` command:

```
@verbatiminclude filename
```

The contents of *filename* is printed in a verbatim environment (see Section 10.4 [`@verbatim`], page 87). Generally, the file is printed exactly as it is, with all special characters and white space retained. No indentation is added; if you want indentation, enclose the `@verbatiminclude` within `@example` (see Section 10.3 [`@example`], page 86).

The name of the file is taken literally, with a single exception: `@value{var}` references are expanded. This makes it possible to reliably include files in other directories in a distribution, for instance:

```
@include @value{top_srcdir}/NEWS
```

(You still have to get `top_srcdir` defined in the first place.)

10.6 `@lisp`: Marking a Lisp Example

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` 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.

10.7 `@small...` Block Commands

In addition to the regular `@example` and `@lisp` commands, Texinfo has "small" example-style commands. These are `@smalldisplay`, `@smallexample`, `@smallformat`, and `@smalllisp`.

In Info, the `@small...` commands are equivalent to their non-small companion commands.

In TEX, however, the `@small...` commands typeset text in a smaller font than the non-small example commands. Consequently, many examples containing long lines fit on a page without needing to be shortened.

Mark the end of an `@small...` block with a corresponding `@end small...`. For example, pair `@smallexample` with `@end smallexample`.

Here is an example of the font used by the `@small...` commands (in Info, the output will be the same as usual):

```
... to make sure that you have the freedom to
distribute copies of free software (and charge for
this service if you wish), that you receive source
code or can get it if you want it, that you can
change the software or use pieces of it in new free
programs; and that you know you can do these things.
```

The `@small...` commands make it easier to prepare manuals without forcing you to edit examples by hand to fit them onto narrower pages.

As a general rule, a printed document looks much better if you use only one of (for instance) `@example` or `@smallexample` consistently within a chapter.

10.8 `@display` and `@smalldisplay`

The `@display` command begins a kind of example, where each line of input produces a line of output, and the output is indented. It is thus 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.

Texinfo also provides a command `@smalldisplay`, which is like `@display` but uses a smaller font in `@smallbook` format. See Section 10.7 [small], page 89.

The `@table` command (see Section 11.4.1 [table], page 98) does not work inside `@display`. Since `@display` is line-oriented, it doesn't make sense to use them together. If you want to indent a table, try `@quotation` (see Section 10.2 [quotation], page 86).

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

10.9 `@format` and `@smallformat`

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. As you can see
from this example,
the `@format` command does not fill the text.

Texinfo also provides a command `@smallformat`, which is like `@format` but uses a smaller font in `@smallbook` format. See Section 10.7 [small], page 89.

10.10 `@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. Do not use braces around the text. In a printed manual, the text on an `@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 group
```

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.

10.11 `@flushleft` and `@flushright`

The `@flushleft` and `@flushright` commands line up the 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. For example,

```
@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.

10.12 `@noindent`: Omitting Indentation

An example or other inclusion can break a paragraph into segments. Ordinarily, the formatters indent text that follows an example as a new paragraph. You can prevent this on a case-by-case basis by writing `@noindent` at the beginning of a line, preceding the continuation text. You can also disable indentation for all paragraphs globally with `@paragraphindent` (see Section 3.7.3 [paragraphindent], page 42).

It is best to write `@noindent` on a line by itself, since in most environments, spaces following the command will not be ignored. It's ok to use it at the beginning of a line, with text following, outside of any environment.

For example:

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

@noindent
This line is not 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
@code{@@display} and @code{@@end display}.)
```

produces:

```
This is an example
```

This line is not 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 manual, each line that says 'produces' is preceded by `@noindent`.

Do not put braces after an `@noindent` command; they are not necessary, since `@noindent` is a command used outside of paragraphs (see Section A.1 [Command Syntax], page 209).

10.13 `@indent`: Forcing Indentation

To complement the `@noindent` command (see the previous section), Texinfo provides the `@indent` command that forces a paragraph to be indented. This paragraph, for instance, is indented using an `@indent` command. The first paragraph of a section is the most likely place to use `@indent`, to override the normal behavior of no indentation there (see Section 3.7.3 [paragraphindent], page 42).

It is best to write `@indent` on a line by itself, since in most environments, spaces following the command will not be ignored. The `@indent` line will not generate a blank line in the Info output within an environment.

However, it is ok to use it at the beginning of a line, with text following, outside of any environment.

Do not put braces after an `@indent` command; they are not necessary, since `@indent` is a command used outside of paragraphs (see Section A.1 [Command Syntax], page 209).

10.14 `@cartouche`: Rounded Rectangles Around Examples

In a printed manual, the `@cartouche` command draws a box with rounded corners around its contents. In HTML, a normal rectangle is drawn (that's the best HTML can do). `@cartouche` has no effect in Info output.

You can use this command to further highlight an example or quotation. For instance, you could write a manual in which one type of example is surrounded by a cartouche for emphasis.

For example,

```
@cartouche
@example
% pwd
/usr/local/share/emacs
@end example
@end cartouche
```

surrounds the two-line example with a box with rounded corners, in the printed manual.

The output from the example looks like this (if you're reading this in Info, you'll see the `@cartouche` had no effect):

```
% pwd
/usr/local/info
```

For proper output in HTML, it's necessary to put the `@cartouche` around the `@example`, and not the other way around. This limitation of `makeinfo` may be removed one day.

`@cartouche` also implies `@group` (see Section 15.8 [group], page 125).

11 Lists and Tables

Texinfo has several ways of making lists and tables. Lists can be bulleted or numbered; two-column tables can highlight the items in the first column; multi-column tables are also supported.

11.1 Introducing Lists

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 need 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` or `@itemx` command.

Here is an itemized list of the different kinds of table and lists:

- Itemized lists with and without bullets.
- Enumerated lists, using numbers or letters.
- Two-column tables with highlighting.

Here is an enumerated list with the same items:

1. Itemized lists with and without bullets.
2. Enumerated lists, using numbers or letters.
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`
: Enumerated lists, using numbers or letters.

`@table`
`@ftable`
`@vtable` Two-column tables, optionally with indexing.

11.2 `@itemize`: Making an Itemized List

The `@itemize` command produces sequences of indented paragraphs, with a bullet or other mark inside the left margin at the beginning of each paragraph for which such 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 command or character that results in a single character in the Info file. If you don't want any mark at all, use `@w`. (When you write the mark command such as `@bullet` after an `@itemize` command, you may omit the '`{}`'.) If you don't specify a mark command, the default is `@bullet`.

Write the text of the indented paragraphs themselves after the `@itemize`, up to another line that says `@end itemize`.

At the beginning of each paragraph for which a mark in the margin is desired, write a line that starts with `@item`. It is ok to have text following the `@item`.

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. `@bullet` produces an '`*`' in Info and a round dot in TEX.

```
@itemize @bullet
@item
Some text for foo.

@item
Some text
for bar.
@end itemize
```

This 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
```

This produces:

- First item.
  - Inner item.
  - Second inner item.
- Second outer item.

11.3 `@enumerate`: Making a Numbered or Lettered List

`@enumerate` is like `@itemize` (see Section 11.2 [`@itemize`], page 95), except that the labels on the items are successive integers or letters instead of bullets.

Write the `@enumerate` command at the beginning of a line. The command does not require an argument, but accepts either a number or a letter as an option. Without an argument, `@enumerate` starts the list with the number '`1`'. With a numeric argument, such as '`3`', the command starts the list with that number. With an upper or lower case letter, such as '`a`' or '`A`', the command starts the list with that letter.

Write the text of the enumerated list in the same way as an itemized list: write a line starting with `@item` at the beginning of each paragraph that you want enumerated. It is ok to have text following the `@item`.

You should put a blank line between entries in the list. This generally makes it easier to read the Info file.

Here is an example of `@enumerate` without an argument:

```
@enumerate
@item
Underlying causes.

@item
Proximate causes.
@end enumerate
```

This produces:

1. Underlying causes.

2. Proximate causes.

Here is an example with an argument of *3*:

```
@enumerate 3
@item
Predisposing causes.

@item
Precipitating causes.

@item
Perpetuating causes.
@end enumerate
```

This produces:

3. Predisposing causes.
4. Precipitating causes.
5. Perpetuating causes.

Here is a brief summary of the alternatives. The summary is constructed using `@enumerate` with an argument of `a`.

a. `@enumerate`

   Without an argument, produce a numbered list, starting with the number 1.

b. `@enumerate` *`positive-integer`*

   With a (positive) numeric argument, start a numbered list with that number. You can use this to continue a list that you interrupted with other text.

c. `@enumerate` *`upper-case-letter`*

   With an upper case letter as argument, start a list in which each item is marked by a letter, beginning with that upper case letter.

d. `@enumerate` *`lower-case-letter`*

   With a lower case letter as argument, start a list in which each item is marked by a letter, beginning with that lower case letter.

You can also nest enumerated lists, as in an outline.

11.4 Making a Two-column Table

`@table` is similar to `@itemize` (see Section 11.2 [`@itemize`], page 95), but allows you to specify a name or heading line for each item. The `@table` command is used to produce two-column tables, and is especially useful for glossaries, explanatory exhibits, and command-line option summaries.

11.4.1 Using the `@table` Command

Use the `@table` command to produce two-column tables. It is usually listed for "definition lists" of various sorts, where you have a list of terms and a brief text with each one.

Write the `@table` command at the beginning of a line, after a blank line, and follow it on the same line with an argument that is a Texinfo "indicating" command such as `@code`, `@samp`, `@var`, `@option`, or `@kbd` (see Section 9.1 [Indicating], page 74).

This command will be applied to the text that goes into the first column of each item and thus determines how it will be highlighted. For example, `@table @code` will cause the text in the first column to be output as if it `@code` command.

You may also use the `@asis` command as an argument to `@table`. `@asis` is a command that does nothing; if you use this command after `@table`, the first column entries are output without added highlighting ("as is").

The `@table` command works with other commands besides those explicitly mentioned here. However, you can only use commands that normally take arguments in braces. (In this case, however, you use the command name without an argument, because the subsequent `@item`'s will supply the argument.)

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 do not need 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 the text on the same line as the `@item` will be placed in the first column (including any footnotes).

Normally, you should put a blank line before an `@item` line. This puts a blank line in the Info file. Except when the entries are very brief, a blank line looks better.

End the table with a line consisting of `@end table`, followed by a blank line. TEX will always start a new paragraph after the table, so the blank line is needed for the Info output to be analogous.

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
```

This 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 11.4.3 [`@itemx`], page 99.)

11.4.2 `@ftable` and `@vtable`

The `@ftable` and `@vtable` commands are the same as the `@table` command except that `@ftable` automatically enters each of the items in the first column of the table into the index of functions and `@vtable` automatically enters each of the items in the first column of the table into the index of variables. This simplifies the task of creating 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 13 [Indices], page 108, for more information about indices.

Begin a two-column table using `@ftable` or `@vtable` by writing the `@`-command 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` or `@end vtable` command on a line by itself.

See the example for `@table` in the previous section.

11.4.3 `@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 `@item` for the first entry, and `@itemx` for all subsequent entries; `@itemx` must always follow an `@item` command, with no blank line intervening.

The `@itemx` command works exactly like `@item` except that it does not generate extra vertical space above the first column text. If you have multiple consecutive `@itemx` commands, do not insert any blank lines between them.

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.
@end table
```

This 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.)

11.5 `@multitable`: Multi-column Tables

`@multitable` allows you to construct tables with any number of columns, with each column having any width you like.

You define the column widths on the `@multitable` line itself, and write each row of the actual table following an `@item` command, with columns separated by an `@tab` command. Finally, `@end multitable` completes the table. Details in the sections below.

11.5.1 Multitable Column Widths

You can define the column widths for a multitable in two ways: as fractions of the line length; or with a prototype row. Mixing the two methods is not supported. In either case, the widths are defined entirely on the same line as the `@multitable` command.

1. To specify column widths as fractions of the line length, write `@columnfractions` and the decimal numbers (presumably less than 1; a leading zero is allowed and ignored) after the `@multitable` command, as in:

   ```
   @multitable @columnfractions .33 .33 .33
   ```

   The fractions need not add up exactly to 1.0, as these do not. This allows you to produce tables that do not need the full line length.

2. To specify a prototype row, write the longest entry for each column enclosed in braces after the `@multitable` command. For example:

   ```
   @multitable {some text for column one} {for column two}
   ```

   The first column will then have the width of the typeset 'some text for column one', and the second column the width of 'for column two'.

   The prototype entries need not appear in the table itself.

   Although we used simple text in this example, the prototype entries can contain Texinfo commands; markup commands such as `@code` are particularly likely to be useful.

11.5.2 Multitable Rows

After the `@multitable` command defining the column widths (see the previous section), you begin each row in the body of a multitable with `@item`, and separate the column entries with `@tab`. Line breaks are not special within the table body, and you may break input lines in your source file as necessary.

You can also use `@headitem` instead of `@item` to produce a *heading row*. The TEX output for such a row is in bold, and the HTML, XML, and Docbook output uses the `<thead>` tag.

Here is a complete example of a multi-column table (the text is from *The GNU Emacs Manual*, see section "Splitting Windows" in *The GNU Emacs Manual*):

```
@multitable @columnfractions .15 .45 .4
@headitem Key @tab Command @tab Description
@item C-x 2
@tab @code{split-window-vertically}
@tab Split the selected window into two windows,
with one above the other.
@item C-x 3
@tab @code{split-window-horizontally}
@tab Split the selected window into two windows
positioned side by side.
@item C-Mouse-2
@tab
@tab In the mode line or scroll bar of a window,
split that window.
@end multitable
```

produces:

Key	Command	Description
C-x 2	`split-window-vertically`	Split the selected window into two windows, with one above the other.
C-x 3	`split-window-horizontally`	Split the selected window into two windows positioned side by side.
C-Mouse-2		In the mode line or scroll bar of a window, split that window.

12 Special Displays

The commands in this chapter allow you to write text that is specially displayed (output format permitting), outside of the normal document flow.

One set of such commands is for creating "floats", that is, figures, tables, and the like, set off from the main text, possibly numbered, captioned, and/or referred to from elsewhere in the document. Images are often included in these displays.

Another group of commands is for creating footnotes in Texinfo.

12.1 Floats

A *float* is a display which is set off from the main text. It is typically labelled as being a "Figure", "Table", "Example", or some similar type.

A float is so-named because, in principle, it can be moved to the bottom or top of the current page, or to a following page, in the printed output. (Floating does not make sense in other output formats.) In the present version of Texinfo, however, this floating is unfortunately not yet implemented. Instead, the floating material is simply output at the current location, more or less as if it were an `@group` (see Section 15.8 [`@group`], page 125).

12.1.1 `@float` [*type*][,*label*]: Floating material

To produce floating material, enclose the material you want to be displayed separate between `@float` and `@end float` commands, on lines by themselves.

Floating material uses `@image` to display an already-existing graphic (see Section 12.2 [Images], page 104), or `@multitable` to display a table (see Section 11.5 [Multi-column Tables], page 99). However, the contents of the float can be anything. Here's an example with simple text:

```
@float Figure,fig:ex1
This is an example float.
@end float
```

And the output:

This is an example float.

Figure 12.1

As shown in the example, `@float` takes two arguments (separated by a comma), *type* and *label*. Both are optional.

type Specifies the sort of float this is; typically a word such as "Figure", "Table", etc. If not given, and *label* is, any cross-referencing will simply use a bare number.

label Specifies a cross-reference label for this float. If given, this float is automatically given a number, and will appear in any `@listofloats` output (see Section 12.1.3 [listoffloats], page 103). Cross-references to *label* are allowed.

On the other hand, if *label* is not given, then the float will not be numbered and consequently will not appear in the `@listoffloats` output or be cross-referenceable.

Normally, you specify both *type* and *label*, to get a labeled and numbered float.

In Texinfo, all floats are numbered the same way: with the chapter number (or appendix letter), a period, and the float number, which simply counts 1, 2, 3, ..., and is reset at each chapter. Each float type is counted independently.

Floats within an `@unnumbered` are numbered, or outside of any chapter, are simply numbered consecutively from 1.

These numbering conventions are not, at present, changeable.

12.1.2 `@caption` & `@shortcaption`

You may write an `@caption` anywhere within a `@float` environment, to define a caption for the float. It is not allowed in any other context. `@caption` takes a single argument, enclosed in braces. Here's an example:

```
@float
An example float, with caption.
@caption{Caption for example float.}
@end float
```

The output is:

An example float, with caption.

Caption for example float.

`@caption` can appear anywhere within the float; it is not processed until the `@end float`. The caption text is usually a sentence or two, but may consist of several paragraphs if necessary.

In the output, the caption always appears below the float; this is not currently changeable. It is preceded by the float type and/or number, as specified to the `@float` command (see the previous section).

The `@shortcaption` command likewise may be used only within `@float`, and takes a single argument in braces. The short caption text is used instead of the caption text in a list of floats (see the next section). Thus, you can write a long caption for the main document, and a short title to appear in the list of floats. For example:

```
@float
... as above ...
@shortcaption{Text for list of floats.}
@end float
```

The text for `@caption` and `@shortcaption` may not contain comments (`@c`), verbatim text (`@verb`), environments such as `@example`, or other complex constructs.

12.1.3 `@listoffloats`: Tables of contents for floats

You can write a `@listoffloats` command to generate a list of floats for a given float type (see Section 12.1.1 [float], page 102), analogous to the document's overall table of contents. Typically, it is written in its own `@unnumbered` node to provide a heading and structure, rather like `@printindex` (see Section 4.1 [Printing Indices & Menus], page 45).

`@listoffloats` takes one optional argument, the float type. Here's an example:

```
@node List of Figures
@unnumbered List of Figures
@listoffloats Figure
```

And the output from `@listoffloats`:

Figure 12.1 102

Without any argument, `@listoffloats` generates a list of floats for which no float type was specified, i.e., no first argument to the `@float` command (see Section 12.1.1 [float], page 102).

Each line in the list of floats contains the float type (if any), the float number, and the caption, if any—the `@shortcaption` argument, if it was specified, else the `@caption` argument. In Info, the result is a menu where each float can be selected. In HTML, each line is a link to the float. In printed output, the page number is included.

Unnumbered floats (those without cross-reference labels) are omitted from the list of floats.

12.2 Inserting Images

You can insert an image given in an external file with the `@image` command. Although images can be used anywhere, including the middle of a paragraph, we describe them in this chapter since they are most often part of a displayed figure or example.

12.2.1 Image Syntax

Here is the basic synopsis of the `@image` command:

```
@image{filename[, width[, height[, alttext[, extension]]]]}
```

The *filename* argument is mandatory, and must not have an extension, because the different processors support different formats:

- TEX reads the file '*filename*`.eps`' (Encapsulated PostScript format).
- PDFTEX reads '*filename*`.pdf`' (Adobe's Portable Document Format).
- `makeinfo` includes '*filename*`.txt`' verbatim for Info output (more or less as if it was an `@example`).
- `makeinfo` uses the optional fifth argument *extension* to `@image` for the filename extension, if it is specified. For example:

  ```
  @image{foo,,,,,.xpm}
  ```

  will cause `makeinfo` to look for '`foo.xpm`' before any others.

The *width* and *height* arguments are described in the next section.

For TEX output, if an image is the first thing in a paragraph, for example if you want two images side-by-side, you must precede it with `@noindent` (see Section 10.12 [`@noindent`], page 91). Otherwise it will be displayed on a line by itself. If you want it centered, use `@center` (see Section 3.4.2 [`@titlefont @center @sp`], page 34).

When producing html, `makeinfo` sets the *alt attribute* for inline images to the optional *alttext* (fourth) argument to `@image`, if supplied. If not supplied, `makeinfo` uses the full file name of the image being displayed. If you want an empty `alt` string, use `@-`. The *alttext*

is taken as Texinfo text, so special characters such as '"' and '<' and '&' are escaped in the HTML and XML output.

If you do not supply the optional *extension* (fifth) argument, `makeinfo` first tries '*filename*`.png`'; if that does not exist, it tries '*filename*`.jpg`'. If that does not exist either, it complains. (We cannot support GIF format directly due to software patents.)

In Info output, `makeinfo` writes a reference to the binary image file (trying *filename* suffixed with '*extension*', '*.extension*', '`.png`', or '`.jpg`', in that order) if one exists. It also literally includes the '`.txt`' file if one exists. This way, Info readers which can display images (such as the Emacs Info browser, running under X) can do so, whereas Info readers which can only use text (such as the standalone Info reader) can display the textual version.

The implementation of this is to put the following construct into the Info output:

```
^@^H[image src="binaryfile" text="txtfile"
          alt="alttext ... ^@^H]
```

where '`^@`' and '`^H`' stand for the actual null and backspace control characters. If one of the files is not present, the corresponding argument is omitted.

The reason for mentioning this here is that older Info browsers (this feature was introduced in Texinfo version 4.6) will display the above literally, which, although not pretty, should not be harmful.

12.2.2 Image Scaling

The optional *width* and *height* arguments to the `@image` command (see the previous section) specify the size to scale the image to. They are ignored for Info output. If neither is specified, the image is presented in its natural size (given in the file); if only one is specified, the other is scaled proportionately; and if both are specified, both are respected, thus possibly distorting the original image by changing its aspect ratio.

The *width* and *height* may be specified using any valid TEX dimension, namely:

pt	point (72.27pt = 1in)
pc	pica (1pc = 12pt)
bp	big point (72bp = 1in)
in	inch
cm	centimeter (2.54cm = 1in)
mm	millimeter (10mm = 1cm)
dd	didôt point (1157dd = 1238pt)
cc	cicero (1cc = 12dd)
sp	scaled point (65536sp = 1pt)

For example, the following will scale a file '`ridt.eps`' to one inch vertically, with the width scaled proportionately:

```
@image{ridt,,1in}
```

For `@image` to work with TEX, the file '`epsf.tex`' must be installed somewhere that TEX can find it. (The standard location is '*texmf*`/tex/generic/dvips/epsf.tex`', where

texmf is a root of your TEX directory tree.) This file is included in the Texinfo distribution and is also available from `ftp://tug.org/tex/epsf.tex`, among other places.

`@image` can be used within a line as well as for displayed figures. Therefore, if you intend it to be displayed, be sure to leave a blank line before the command, or the output will run into the preceding text.

Image scaling is presently implemented only in TEX, not in HTML or any other sort of output.

12.3 Footnotes

A *footnote* is for a reference that documents or elucidates the primary text.[1] Footnotes are distracting; use them sparingly, if at all. Standard bibliographical references are better placed in a bibliography at the end of a document than in footnotes throughout.

12.3.1 Footnote Commands

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. Footnotes may be of any length (they will be broken across pages if necessary), but are usually short. The template is:

```
ordinary text@footnote{text of footnote}
```

As shown here, the `@footnote` command should come right after the text being footnoted, with no intervening space; otherwise, the footnote marker might end up starting a line.

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...
```

As you can see, the source includes two punctuation marks next to each other; in this case, '`.};`' is the sequence. This is normal (the first ends the footnote and the second belongs to the sentence being footnoted), so don't worry that it looks odd.

In a printed manual or book, the reference mark for a footnote is a small, superscripted number; the text of the footnote appears 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)`'. The reference mark is followed by a cross-reference link to the footnote's text.

In the HTML output, footnote references are marked with a small, superscripted number which is rendered as a hypertext link to the footnote text.

By the way, footnotes in the argument of an `@item` command for a `@table` must be on the same line as the `@item` (as usual). See Section 11.4 [Two-column Tables], page 97.

---

[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.

12.3.2 Footnote Styles

Info has two footnote styles, which determine where the text of the footnote is located:

- In the 'End' 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:

  ```
  --------- Footnotes ---------

  (1)  Here is a sample footnote.
  ```

- In the 'Separate' node 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 which you use to reach the footnote node.

  The name of the node with the footnotes 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) The first syllable of "Texinfo" is pronounced like "speck", not
  "hex". ...
  ```

Unless your document has long and important footnotes (as in, say, Gibbon's *Decline and Fall . . .*), we recommend the '`end`' style, as it is simpler for readers to follow.

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 use their default style. Currently, `texinfo-format-buffer` and `texinfo-format-region` use the 'separate' style and `makeinfo` uses the 'end' style.

13 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 entry 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. Lastly, you can define your own new indices.

See Section 4.1 [Printing Indices & Menus], page 45, for information on how to print indices.

13.1 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, one reader may 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 readers.

Like typesetting, the construction of an index is a highly skilled, professional art, the subtleties of which are not appreciated until you need to do it yourself.

See Section 4.1 [Printing Indices & Menus], page 45, for information about printing an index at the end of a book or creating an index menu in an Info file.

13.2 Predefined Indices

Texinfo provides six predefined indices. Here are their nominal meanings, abbreviations, and the corresponding index entry commands:

'`cp`' (`@cindex`) concept index, for general concepts.

'`fn`' (`@findex`) function index, for function and function-like names (such as entry points of libraries).

'`ky`' (`@kindex`) keystroke index, for keyboard commands.

'`pg`' (`@pindex`) program index, for names of programs.

'`tp`' (`@tindex`) data type index, for type names (such as structures defined in header files).

'`vr`' (`@vindex`) variable index, for variable names (such as global variables of libraries).

Not every manual needs all of these, and most manuals use only two or three at most. The present manual, for example, 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).

You are not required to use the predefined indices strictly 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.

On the other hand, it is best not to stray too far from the meaning of the predefined indices. Otherwise, in the event that your text is combined with other text from other manuals, the index entries will not match up. Instead, define your own new index (see Section 13.5 [New Indices], page 111).

We recommend having a single index in the final document whenever possible, however many source indices you use, since then readers have only one place to look. Two or more source indices can be combined into one output index using the `@synindex` or `@syncodeindex` commands (see Section 13.4 [Combining Indices], page 110).

13.3 Defining the Entries of an Index

The data to make an index come from many individual indexing commands scattered throughout the Texinfo source file. Each command says to add one entry to a particular index; after formatting, the index 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, defining
@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, as listed in the previous section.

Concept index entries consist of text. The best way to write an index is to choose entries that are terse yet clear. If you can do this, the index often looks better if the entries are not capitalized, but written just as they would appear in the middle of a sentence. (Capitalize proper names and acronyms that always call for upper case letters.) This is the case convention we use in most GNU manuals' indices.

If you don't see how to make an entry terse yet clear, make it longer and clear—not terse and confusing. If many of the entries are several words long, the index may look better if you use a different convention: to capitalize the first word of each entry. But do not capitalize a case-sensitive name such as a C or Lisp function name or a shell command; that would be a spelling error.

Whichever case convention you use, please use it consistently!

Entries in indices other than the concept index are symbol names in programming languages, or program names; these names are usually case-sensitive, so use upper and lower case as required for them.

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 (see Chapter 9 [Marking Text], page 74), and `@r` for the normal roman font (see Section 9.2.3 [Fonts], page 84).

> **Caution:** Do not use a colon in an index entry. In Info, a colon separates the menu entry name from the node name, so a colon in the entry itself confuses Info. See Section 7.3 [Menu Parts], page 61, for more information about the structure of a menu entry.

13.4 Combining Indices

Sometimes you will want to combine two disparate indices such as functions and concepts, perhaps because you have few enough entries that a separate index 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 as part of 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 need to enclose every one of them between the braces of `@code`.

13.4.1 `@syncodeindex`

When you want to combine functions and concepts into one index, you should index the functions with `@findex` and index the concepts with `@cindex`, and use the `@syncodeindex` command to redirect the function index entries into the concept index.

The `@syncodeindex` command takes two arguments; they are the name of the index to redirect, and the name of the index to redirect it to. The template looks like this:

```
@syncodeindex from to
```

For this purpose, the indices are given two-letter names:

'`cp`' concept index

'`fn`' function index

'`vr`' variable index

'`ky`' key index

'`pg`' program index

'`tp`' 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 merge in with the concept index instead.

To merge both a variables index and a function index into a concept index, write the following:

```
@syncodeindex vr cp
@syncodeindex fn cp
```

The `@syncodeindex` command puts all the entries from the 'from' index (the redirected index) into the `@code` font, overriding whatever default font is used by the index to which the entries are now directed. This way, if you direct function names from a function index into a concept index, all the function names are printed in the `@code` font as you would expect.

13.4.2 `@synindex`

The `@synindex` command is nearly the same as the `@syncodeindex` command, except that it does not put the 'from' index entries into the `@code` font; rather it puts them in the roman font. Thus, you use `@synindex` when you merge a concept index into a function index.

See Section 4.1 [Printing Indices & Menus], page 45, for information about printing an index at the end of a book or creating an index menu in an Info file.

13.5 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 this 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 an abbreviation for "author".)

In general, Texinfo constructs the new indexing command by concatenating the name of the index with '`index`'; thus, defining an '`xy`' index leads to the automatic creation of an `@xyindex` command.

Use the `@printindex` command to print the index, as you do with the predefined indices. For example:

```
@node Author Index
@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 by default instead of a roman font.

You should define new indices before the end-of-header line of a Texinfo file, and (of course) before any `@synindex` or `@syncodeindex` commands (see Section 3.2 [Texinfo File Header], page 29).

14 Special Insertions

Texinfo provides several commands for inserting characters that have special meaning in Texinfo, such as braces, and for other graphic elements that do not correspond to simple characters you can type.

These are:

- '@' and braces and commas.
- Whitespace within and around a sentence.
- Accents.
- Dots and bullets.
- The TEX logo and the copyright symbol.
- The pounds currency symbol.
- The minus sign.
- Mathematical expressions.
- Glyphs for evaluation, macros, errors, etc.
- Footnotes.
- Images.

14.1 Inserting @ and {} and ,

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

The comma ',' is a special character only in one uncommon context: it separates arguments to commands that take multiple arguments.

14.1.1 Inserting '@' with @@

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

Do not put braces after an @@ command.

14.1.2 Inserting '{' and '}' with @{ 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.ppp

14.1.3 Inserting ',' with @comma{}

Ordinarily, a comma ',' is a normal character that can be simply typed in your input where you need it.

However, Texinfo uses the comma as a special character in one uncommon context: some commands, such as @acronym (see Section 9.1.15 [acronym], page 81) and @xref (see Chapter 8 [Cross References], page 64), as well as user-defined macros (see Section 19.1 [Defining Macros], page 150), can take more than one argument. In these cases, the comma character is used to separate arguments.

Since a comma chacter would confuse Texinfo's parsing for these commands, you must use the command '`@,`' instead if you want to have an actual comma in the output. Here are some examples:

```
@acronym{ABC, A Bizarre @comma{}}
@xref{Comma,, The @comma{} symbol}
@mymac{One argument@comma{} containing a comma}
```

Although @, can be used anywhere, there is no need for it anywhere except in this unusual case.

14.2 Inserting Space

The following sections describe commands that control spacing of various kinds within and after sentences.

14.2.1 Not Ending a Sentence

Depending on whether a period or exclamation point or question mark is inside 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 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 do not need 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.

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.

For example,

```
The s.o.p.@: has three parts ...
The s.o.p. has three parts ...
```

produces the following. If you look carefully at this printed output, you will see a little extraneous space after '`s.o.p.`' in the second line.

The s.o.p. has three parts ...
The s.o.p. has three parts ...

(Incidentally, '`s.o.p.`' is an abbreviation for "Standard Operating Procedure".)

`@:` has no effect on the Info and HTML output. In Docbook and XML, the previous punctuation character (.?!:) is output as an entity instead of as the normal character: '`. ? ! :`'. This gives further processors a chance to notice and not add the usual extra space.

Do not put braces after `@:` (or any non-alphabetic command).

14.2.2 Ending a Sentence

Use `@.` instead of a period, `@!` instead of an exclamation point, and `@?` instead of a question mark at the end of a sentence that ends with a 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 M.I.B. and to M.E.W@.  Also, give it to R.J.C@.
Give it to M.I.B. and to M.E.W.  Also, give it to R.J.C.
```

produces the following. If you look carefully at this printed output, you will see a little more whitespace after the 'W' in the first line.

Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.
Give it to M.I.B. and to M.E.W. Also, give it to R.J.C.

In the Info file output, `@.` is equivalent to a simple '.'; likewise for `@!` and `@?`.

The meanings of `@:` and `@.` in Texinfo are designed to work well with the Emacs sentence motion commands (see section "Sentences" in *The GNU Emacs Manual*).

Do not put braces after any of these commands.

14.2.3 Multiple Spaces

Ordinarily, TEX collapses multiple whitespace characters (space, tab, and newline) into a single space. Info output, on the other hand, preserves whitespace as you type it, except for changing a newline into a space; this is why it is important to put two spaces at the end of sentences in Texinfo documents.

Occasionally, you may want to actually insert several consecutive spaces, either for purposes of example (what your program does with multiple spaces as input), or merely for purposes of appearance in headings or lists. Texinfo supports three commands: *@SPACE*, *@TAB*, and *@NL*, all of which insert a single space into the output. (Here, *@SPACE* represents an '@' character followed by a space, i.e., '@ ', and *TAB* and *NL* represent the tab character and end-of-line, i.e., when '@' is the last character on a line.)

For example,

```
Spacey@ @ @ @
example.
```

produces

```
Spacey    example.
```

Other possible uses of *@SPACE* have been subsumed by `@multitable` (see Section 11.5 [Multi-column Tables], page 99).

Do not follow any of these commands with braces.

To produce a non-breakable space, see Section 15.5 [tie], page 125.

14.2.4 @dmn{*dimension*}: Format a Dimension

At times, you may want to write '12 pt' or '8.5 in' with little or no space between the number and the abbreviation for the dimension. You can use the `@dmn` command to do this. On seeing the command, TEX inserts just enough space for proper typesetting; the Info formatting commands insert no space at all, since the Info file does not require it.

To use the `@dmn` command, write the number and then follow it immediately, with no intervening space, by `@dmn`, and then by the dimension within braces. For example,

```
A4 paper is 8.27@dmn{in} wide.
```

produces

A4 paper is 8.27 in wide.

Not everyone uses this style. Some people prefer '`8.27 in.@:`' or '`8.27 inches`' to '`8.27@dmn{in}`' in the Texinfo file. In these cases, however, the formatters may insert a line break between the number and the dimension, so use `@w` (see Section 15.4 [w], page 124). Also, if you write a period after an abbreviation within a sentence, you should write '`@:`' after the period to prevent TEX from inserting extra whitespace, as shown here. See Section 14.2.1 [Not Ending a Sentence], page 114.

14.3 Inserting Accents

Here is a table with the commands Texinfo provides for inserting floating accents. The commands with non-alphabetic names do not take braces around their argument (which is taken to be the next character). (Exception: `@,` *does* take braces around its argument.) This is so as to make the source as convenient to type and read as possible, since accented characters are very common in some languages.

To get the true accented characters output in Info, and not just the ASCII transliterations, you can use the '`--enable-encoding`' option to `makeinfo` (see Section 21.1.3 [makeinfo options], page 168).

Command	Output	What
`@"o`	ö	umlaut accent
`@'o`	ó	acute accent
`@,{c}`	ç	cedilla accent
`@=o`	ō	macron/overbar accent
`@^o`	ô	circumflex accent
`@`o`	ò	grave accent
`@~o`	õ	tilde accent
`@dotaccent{o}`	ȯ	overdot accent
`@H{o}`	ő	long Hungarian umlaut
`@ringaccent{o}`	ů	ring accent
`@tieaccent{oo}`	o͡o	tie-after accent
`@u{o}`	ŏ	breve accent
`@ubaraccent{o}`	o̲	underbar accent
`@udotaccent{o}`	ọ	underdot accent
`@v{o}`	ǒ	hacek/check/caron accent

This table lists the Texinfo commands for inserting other characters commonly used in languages other than English.

`@exclamdown{}`	¡	upside-down !
`@questiondown{}`	¿	upside-down ?
`@aa{} @AA{}`	å Å	a,A with circle
`@ae{} @AE{}`	æ Æ	ae,AE ligatures
`@dotless{i}`	ı	dotless i
`@dotless{j}`	ȷ	dotless j
`@l{} @L{}`	ł Ł	suppressed-L,l
`@o{} @O{}`	ø Ø	O,o with slash
`@oe{} @OE{}`	œ Œ	oe,OE ligatures
`@ordf{} @ordm{}`	a̲ o̲	Spanish ordinals
`@ss{}`	ß	es-zet or sharp S

14.4 Inserting Ellipsis and Bullets

An *ellipsis* (a line of dots) is not typeset as 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. (You need to use braces with these commands because you can use them next to other text; without the braces, the formatters would be confused. See Section A.1 [@-Command Syntax], page 209, for further information.)

14.4.1 `@dots{}` (...) and `@enddots{}` (...)

Use the `@dots{}` command to generate an ellipsis, which is three dots in a row, appropriately spaced . . . like so. 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.

Similarly, the `@enddots{}` command generates an end-of-sentence ellipsis, which has different spacing afterwards, . . . Look closely to see the difference.

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.

14.4.2 `@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. (See Section 11.2 [@itemize], page 95.)

14.5 Inserting TeX and Legal Symbols: ©, ®

The logo 'TeX' is typeset in a special fashion and it needs an `@`-command. The copyright and registered symbols, '©' and '®', is also special. Each of these commands is followed by a pair of braces, '`{}`', without any whitespace between the name of the command and the braces.

14.5.1 `@TeX{}` (TeX) and `@LaTeX{}` (LaTeX)

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`'.

Similarly, use the `@LaTeX{}` command to generate 'LaTeX', which is even more special in printed manuals (and different from the incorrect `La@TeX{}`. In Info, the result is just '`LaTeX`'. (LaTeX is another macro package built on top of TeX, very loosely analogous to Texinfo in that it emphasizes logical structure, but much (much) larger.)

The spelling of these commands are unusual among Texinfo commands in that they use both uppercase and lowercase letters.

14.5.2 `@copyright{}` (©)

Use the `@copyright{}` command to generate the copyright symbol, '©'. Where possible, this is a '`c`' inside a circle; in Info, this is '`(C)`'.

14.5.3 `@registeredsymbol{}` (®)

Use the `@registeredsymbol{}` command to generate the registered symbol, '®'. Where possible, this is an '`R`' inside a circle; in Info, this is '`(R)`'.

14.6 `@euro{}` (€): Euro currency symbol

Use the `@euro{}` command to generate '€'. Where possible, this is the symbol for the Euro currency, invented as part of the European economic unification relatively recently. In plain Info, it is the word '`Euro `'. (The space is included in the text transliteration since typically there would be no space after the symbol, so it would be inappropriate to have a space in the source document.)

Texinfo cannot magically synthesize support for the Euro symbol where the underlying system (fonts, software, whatever) does not support it. Therefore, in many cases it is preferable to use the word "Euro". (In banking circles, the abbreviation for the Euro is EUR.)

In order to get the Euro symbol in encoded Info output, for example, it is necessary to specify `@documentencoding ISO-8859-15`. (See Section 18.2 [`@documentencoding`], page 148.) The Euro symbol is in ISO 8859-15 (aka Latin 9), and is *not* in the more widely-used and supported ISO 8859-1 (Latin 1).

14.7 `@pounds{}` (£): Pounds Sterling

Use the `@pounds{}` command to generate '£'. Where possible, this is the symbol for the currency pounds sterling. In Info, it is a '`#`'.

14.8 `@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, shorter than an em-dash:

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

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

'—' is an em-dash for text.

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

You should not use `@minus{}` inside `@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 (see Section 11.2 [`@itemize`], page 95).

14.9 `@math`: Inserting Mathematical Expressions

You can write a short mathematical expression with the `@math` command. Write the mathematical expression between braces, like this:

```
@math{(a + b)(a + b) = a^2 + 2ab + b^2}
```

This produces the following in TEX:

$$(a + b)(a + b) = a^2 + 2ab + b^2$$

and the following in other formats:

```
(a + b)(a + b) = a^2 + 2ab + b^2
```

Thus, the `@math` command has no effect on the Info and HTML output; `makeinfo` just reproduces the input, it does not try to interpret the mathematics in any way.

`@math` implies `@tex`. This not only makes it possible to write superscripts and subscripts (as in the above example), but also allows you to use any of the plain TEX math control sequences. It's conventional to use '\' instead of '@' for these commands. As in:

```
@math{\sin 2\pi \equiv \cos 3\pi}
```

which looks like this in TEX:

$$\sin 2\pi \equiv \cos 3\pi$$

and which looks like the input in Info and HTML:

```
\sin 2\pi \equiv \cos 3\pi
```

Since '\' is an escape character inside `@math`, you can use `@\` to get a literal backslash (`\\` will work in TEX, but you'll get the literal '\\' in Info). `@\` is not defined outside of `@math`, since a '\' ordinarily produces a literal '\'.

For displayed equations, you must at present use TEX directly (see Section 17.3 [Raw Formatter Commands], page 141).

14.10 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 commands to insert glyphs to indicate printed output, error messages, equivalence of expressions, and the location of point.

The glyph-insertion commands do not need to be used within an example, but most often they are. Every glyph-insertion command is followed by a pair of left- and right-hand braces.

14.10.1 Glyphs Summary

Here are the different glyph commands:

⇒	`@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.

14.10.2 `@result{}` (⇒): Indicating Evaluation

Use the `@result{}` command to indicate the result of evaluating an expression.

The `@result{}` command is displayed as '⇒' in the printed output and as '`=>`' in other formats.

Thus, the following,

```
(cdr '(1 2 3))
    ⇒ (2 3)
```

may be read as "`(cdr '(1 2 3))` evaluates to `(2 3)`".

14.10.3 `@expansion{}` (↦): 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 the printed output. and as '`==>`' in other formats.

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.

14.10.4 `@print{}` (⊣): 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 HTML and as '⊣' in the printed output.

In the following example, the printed text is indicated with '⊣', and 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
```

14.10.5 `@error{}` (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 HTML 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
```

' error ' itself is not part of the error message.

14.10.6 `@equiv{}` (≡): 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 HTML 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)`.

14.10.7 `@point{}` (⋆): 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 HTML 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
```

15 Forcing and Preventing Breaks

Usually, a Texinfo file is processed both by TEX and by one of the Info formatting commands. Line, paragraph, or page breaks sometimes 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 break, break prevention, or pagination commands.

15.1 Break Commands

The break commands create or allow line and paragraph breaks:

`@*` Force a line break.

`@sp` *n* Skip *n* blank lines.

`@-` Insert a discretionary hyphen.

`@hyphenation{`*hy-phen-a-ted words*`}`
Define hyphen points in *hy-phen-a-ted words*.

These commands hold text together on a single line:

`@w{`*text*`}` Prevent *text* from being split and hyphenated across two lines.

`@tie{}` Insert a normal interword space at which a line break may not occur.

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.

15.2 `@*` and `@/`: Generate and Allow Line Breaks

The `@*` command forces a line break in both the printed manual and in Info. The `@/` command allows a line break (printed manual only).

Here is an example with `@*`:

```
This line @* is broken @*in two places.
```

produces

```
This line
is broken
in two places.
```

The `@/` command can be useful within a url (see Section 8.9 [`@uref`], page 72), which tend to be long and are otherwise unbreakable. For example:

```
The official Texinfo home page is on the GNU web site:
@uref{http://www.gnu.org/@/software/@/gnu/@/texinfo}.
```

produces

> The official Texinfo home page is on the GNU web site:
> `http://www.gnu.org/software/gnu/texinfo`.

Without the `@/` commands, TEX would have nowhere to break the line. `@/` has no effect in the online output.

15.3 `@-` and `@hyphenation`: Helping TEX Hyphenate

Although TEX's hyphenation algorithm is generally pretty good, it does miss useful hyphenation points from time to time. (Or, far more rarely, insert an incorrect hyphenation.) So, for documents with an unusual vocabulary or when fine-tuning for a printed edition, you may wish to help TEX out. Texinfo supports two commands for this:

`@-` Insert a discretionary hyphen, i.e., a place where TEX can (but does not have to) hyphenate. This is especially useful when you notice an overfull hbox is due to TEX missing a hyphenation (see Section 20.10 [Overfull hboxes], page 163). TEX will not insert any hyphenation points itself into a word containing `@-`.

`@hyphenation{`*hy-phen-a-ted words*`}`
Tell TEX how to hyphenate *hy-phen-a-ted words*. As shown, you put a '`-`' at each hyphenation point. For example:

```
@hyphenation{man-u-script man-u-scripts}
```

TEX only uses the specified hyphenation points when the words match exactly, so give all necessary variants.

Info output is not hyphenated, so these commands have no effect there.

15.4 `@w{`*text*`}`: Prevent Line Breaks

`@w{`*text*`}` outputs *text* and prohibits line breaks within *text*, for both TEX and `makeinfo`.

Thus, you can use `@w` to produce a non-breakable space, fixed at the width of a normal interword space:

```
@w{ } @w{ } @w{ } indentation.
```

produces:

> indentation.

The space from `@w{ }`, as well as being non-breakable, also will not stretch or shrink. Sometimes that is what you want, for instance if you're doing indenting manual. However, usually you want a normal interword space that does stretch and shrink (in the printed output); see the `@tie` command in the next section.

You can also use the `@w` command to prevent TEX from automatically hyphenating a long name or phrase that happens to fall near the end of a line. `makeinfo` does not ever hyphenate words.

You can also use `@w` to avoid unwanted keyword expansion in source control systems. For example, to literally write `$Id$` in your document, use `@w{$}Id$`.

15.5 `@tie{}`: Inserting an Unbreakable Space

The `@tie{}` command produces a normal interword space at which a line break may not occur. Always write it with following (empty) braces, as usual for commands used within a paragraph. Here's an example:

```
@TeX{} was written by Donald E.@tie{}Knuth.
```

produces:

TEX was written by Donald E. Knuth.

There are two important differences between `@tie{}` and `@w{ }`:

- The space produced by `@tie{}` will stretch and shrink slightly along with the normal interword spaces in the paragraph; the space produced by `@w{ }` will not vary.
- `@tie{}` allows hyphenation of the surrounding words, while `@w{ }` inhibits hyphenation of those words (for TEXnical reasons, namely that it produces an '`\hbox`').

15.6 `@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.

15.7 `@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 `@titlepage` section of a Texinfo file to start the copyright page.

15.8 `@group`: Prevent Page Breaks

The `@group` command (on a line by itself) is used inside 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 `@display`, `@format`, `@flushleft` and `@flushright`. See Chapter 10 [Quotations and Examples], page 85. (What all these commands have in common is that each line of input produces a line of output.) In other contexts, `@group` can cause anomalous vertical spacing.

This formatting requirement means that you should write:

```
@example
@group
...
@end group
@end example
```

with the `@group` and `@end group` commands inside the `@example` and `@end example` commands.

The `@group` command is most often used to hold an example together on one page. In this Texinfo manual, more than 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.

15.9 `@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 800 mils (eight-tenths inch) remain on the page. It looks like this:

```
@need 800
This paragraph is preceded by ...
```

The `@need` command is useful for preventing orphans (single lines at the bottoms of printed pages).

16 Definition Commands

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 entity category—'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 (see Section 13.2 [Predefined Indices], page 108).

A manual need not and should not contain more than one definition for a given name. An appendix containing a summary should use `@table` rather than the definition commands.

16.1 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: a line with the `@def...` command and whatever arguments are appropriate for that command; the body of the definition; and a corresponding `@end` line.

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', enclose it in braces. For example:

```
@deffn {Interactive Command} isearch-forward
...
@end deffn
```

Otherwise, the second word will be mistaken for the name of the entity. As a general rule, when any of the arguments in the heading line *except* the last one are more than one word, you need to enclose them in braces.

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. Three predefined, specialized variations (`@defun`, `@defmac`, and `@defspec`) specify the category for you: "Function", "Macro", and "Special Form" respectively. (In Lisp, a special form is an entity much like a function.) Similarly, the general `@defvr` command is accompanied by several specialized variations for describing particular kinds of variables.

See Section 16.7 [Sample Function Definition], page 138, for a detailed example of a function definition, including the use of `@example` inside the definition.

Unfortunately, due to implementation difficulties, macros are not expanded in `@deffn` and all the other definition commands.

16.2 Definition Command Continuation Lines

The heading line of a definition command can get very long. Therefore, Texinfo has a special syntax allowing them to be continued over multiple lines of the source file: a lone '`@`' at the end of each line to be continued. Here's an example:

```
@defun fn-name @
  arg1 arg2 arg3
This is the basic continued defun.
@end defun
```

produces:

`fn-name` *arg1 arg2 arg3* [Function]

This is the basic continued defun.

As you can see, the continued lines are combined, as if they had been typed on one source line.

Although this example only shows a one-line continuation, continuations may extend over any number of lines; simply put an `@` at the end of each line to be continued.

The `@` character does not have to be the last character on the physical line: whitespace is allowed (and ignored) afterwards.

In general, any number of spaces or tabs around the `@` continuation character, both on the line with the `@` and on the continued line, are collapsed into a single space. There is one exception: the Texinfo processors will not fully collapse whitespace around a continuation inside braces. For example:

```
@deffn {Category @
  Name} ...
```

The output (not shown) has excess space between 'Category' and 'Name'. In this case, simply elide any unwanted whitespace in your input, or put the continuation `@` outside braces.

`@` does not (currently) function as a continuation character in *any* other context. Ordinarily, '`@`' followed by a whitespace character (space, tab, newline) produces a normal interword space (see Section 14.2.3 [Multiple Spaces], page 115).

16.3 Optional and Repeated Arguments

Some entities take optional or repeated arguments, which may be specified by a distinctive 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, [*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`'.

16.4 Two or More 'First' Lines

To create two or more 'first' or header lines for a definition, follow the first `@deffn` line by a line beginning with `@deffnx`. The `@deffnx` command works exactly like `@deffn` except that it does not generate extra vertical white space between it and the preceding line.

For example,

```
@deffn {Interactive Command} isearch-forward
@deffnx {Interactive Command} isearch-backward
These two search commands are similar except ...
@end deffn
```

produces

`isearch-forward` [Interactive Command]
`isearch-backward` [Interactive Command]

These two search commands are similar except ...

Each definition command has an 'x' form: `@defunx`, `@defvrx`, `@deftypefunx`, etc.

The 'x' forms work similarly to `@itemx` (see Section 11.4.3 [itemx], page 99).

16.5 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.

16.5.1 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, here is a definition:

```
@deffn Command forward-char nchars
Move point forward @var{nchars} characters.
@end deffn
```

This shows a rather terse definition for a "command" named `forward-char` with one argument, *nchars*.

`@deffn` and prints argument names such as *nchars* in slanted type in the printed output, because we think of these names as metasyntactic variables—they stand for the actual argument values. Within the text of the description, however, 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.

In the unusual case when an argument name contains '`--`', or another character sequence which is treated specially (see Section 1.7 [Conventions], page 9), use `@var` around the argument. This causes the name to be printed in slanted typewriter, instead of the regular slanted font, exactly as input.

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 ...`'. Terminate the definition with `@end defun` on a line of its own. Thus, the template is:

```
@defun function-name arguments...
body-of-definition
@end defun
```

`@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, see section "Special Forms" in *GNU Emacs Lisp Reference Manual*.) `@defspec` is equivalent to '`@deffn {Special Form} ...`' and works like `@defun`.

All these commands create entries in the index of functions.

16.5.2 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 follow 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", enclose it in braces. 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*, i.e., variables intended for users to change according to taste; Emacs has many such (see section "Variables" in *The GNU Emacs Manual*). `@defopt` is equivalent to '`@defvr {User Option} ...`' and works like `@defvar`. It creates an entry in the index of variables.

16.5.3 Functions in Typed Languages

The `@deftypefn` command and its variations are for describing functions in languages in which you must declare types of variables and functions, such as C and C++.

`@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`).

Since in typed languages, the actual names of the arguments are typically scattered among data type names and keywords, Texinfo cannot find them without help. You can either (a) write everything as straight text, and it will be printed in slanted type; (b) use `@var` for the variable names, which will uppercase the variable names in Info and use the slanted typewriter font in printed output; (c) use `@var` for the variable names and `@code` for the type names and keywords, which will be dutifully obeyed.

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 using continuations (see Section 16.2 [Def Cmd Continuation Lines], page 128), but could be on 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 ...`'. The template is:

```
@deftypefun type name arguments...
body-of-description
@end deftypefun
```

`@deftypefun` creates an entry in the index of functions for *name*.

16.5.4 Variables in Typed Languages

Variables in typed languages are handled in a manner similar to functions in typed languages. See Section 16.5.3 [Typed Functions], page 132. 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
```

`@deftypevar` *data-type name*

The `@deftypevar` command is the specialized definition command for variables in typed languages. `@deftypevar` is equivalent to '`@deftypevr Variable ...`'. The template is:

```
@deftypevar data-type name
body-of-description
@end deftypevar
```

These commands create entries in the index of variables.

16.5.5 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.

16.5.6 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 abstract 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.

16.5.6.1 Object-Oriented Variables

These commands allow you to define different sorts of variables in object-oriented programming languages.

`@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. For instance:

```
@defcv {Class Option} Window border-pattern
...
@end defcv
```

produces:

`border-pattern` [Class Option of `Window`]

...

`@defcv` creates an entry in the index of variables.

`@deftypecv` *category class data-type name*

The `@deftypecv` command is the definition command for typed class variables in object-oriented programming. It is analogous to `@defcv` with the addition of the *data-type* parameter to specify the type of the instance variable. Ordinarily, the data type is a programming language construct that should be marked with `@code`. For instance:

```
@deftypecv {Class Option} Window @code{int} border-pattern
...
@end deftypecv
```

produces:

`int border-pattern` [Class Option of `Window`]

...

`@deftypecv` 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} ...`'. For instance:

```
@defivar Window border-pattern
...
@end defivar
```

produces:

`border-pattern` [Instance Variable of `Window`]
...

`@defivar` creates an entry in the index of variables.

`@deftypeivar` *class data-type name*

The `@deftypeivar` command is the definition command for typed instance variables in object-oriented programming. It is analogous to `@defivar` with the addition of the *data-type* parameter to specify the type of the instance variable. Ordinarily, the data type is a programming language construct that should be marked with `@code`. For instance:

```
@deftypeivar Window @code{int} border-pattern
...
@end deftypeivar
```

produces:

`int border-pattern` [Instance Variable of `Window`]
...

`@deftypeivar` creates an entry in the index of variables.

16.5.6.2 Object-Oriented Methods

These commands allow you to define different sorts of function-like entities resembling methods in object-oriented programming languages. These entities take arguments, as functions do, but are associated with particular classes of objects.

`@defop` *category class name arguments...*

The `@defop` command is the general definition command for these method-like entities.

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.

`@deftypeop` *category class data-type name arguments...*

The `@deftypeop` command is the definition command for typed operations in object-oriented programming. It is similar to `@defop` with the addition of the *data-type* parameter to specify the return type of the method. `@deftypeop` creates an entry 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.

`@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.

`@defmethod` creates an entry in the index of functions.

`@deftypemethod` *class data-type name arguments...*

The `@deftypemethod` command is the definition command for methods in object-oriented typed languages, such as C++ and Java. It is similar to the `@defmethod` command with the addition of the *data-type* parameter to specify the return type of the method. `@deftypemethod` creates an entry in the index of functions.

16.6 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.

16.7 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 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}.
@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.

17 Conditionally Visible Text

The *conditional commands* allow you to use different text for different output formats, or for general conditions that you define. For example, you can use them to specify different text for the printed manual and the Info output.

The conditional commands comprise the following categories.

- Commands specific to an output format (Info, TEX, HTML, ...).
- Commands specific to any output format *other* than a given one (not Info, not TEX, ...).
- 'Raw' formatter text for any output format, passed straight through with no interpretation of `@`-commands.
- Format-independent variable substitutions, and testing if a variable is set or clear.

17.1 Conditional Commands

Texinfo has an `@if`*`format`* environment for each output format, to allow conditional inclusion of text for a particular output format.

`@ifinfo` begins segments of text that should be ignored by TEX when it typesets the printed manual, and by `makeinfo` when not producing Info output. The segment of text appears only in the Info file and, for historical compatibility, the plain text output.

The environments for the other formats are analogous:

`@ifdocbook ... @end ifdocbook`
: Text to appear only in the Docbook output.

`@ifhtml ... @end ifhtml`
: Text to appear only in the HTML output.

`@ifplaintext ... @end ifplaintext`
: Text to appear only in the plain text output.

`@iftex ... @end iftex`
: Text to appear only in the printed manual.

`@ifxml ... @end ifxml`
: Text to appear only in the XML output.

The `@if...` and `@end if...` commands must appear on lines by themselves in your source file.

Here is an example showing all these conditionals:

```
@iftex
This text will appear only in the printed manual.
@end iftex
@ifinfo
However, this text will appear only in Info and plain text.
@end ifinfo
@ifhtml
And this text will only appear in HTML.
@end ifhtml
```

```
@ifplaintext
Whereas this text will only appear in plain text.
@end ifplaintext
@ifxml
Notwithstanding that this will only appear in XML.
@end ifxml
@ifdocbook
Nevertheless, this will only appear in Docbook.
@end ifdocbook
```

The preceding example produces the following line:

This text will appear only in the printed manual.

Notice that you only see one of the input lines, depending on which version of the manual you are reading.

17.2 Conditional Not Commands

You can specify text to be included in any output format *other* than a given one with the `@ifnot...` environments:

```
@ifnotdocbook ... @end ifnotdocbook
@ifnothtml ... @end ifnothtml
@ifnotinfo ... @end ifnotinfo
@ifnotplaintext ... @end ifnotplaintext
@ifnottex ... @end ifnottex
@ifnotxml ... @end ifnotxml
```

The `@ifnot...` command and the `@end` command must appear on lines by themselves in your actual source file.

If the output file is being made in the given format, the region is *ignored*. Otherwise, it is included.

There is one exception (for historical compatibility): `@ifnotinfo` text is omitted for both Info and plain text output, not just Info. To specify text which appears only in Info and not in plain text, use `@ifnotplaintext`, like this:

```
@ifinfo
@ifnotplaintext
This will be in Info, but not plain text.
@end ifnotplaintext
@end ifinfo
```

The regions delimited by these commands are ordinary Texinfo source as with `@iftex`, not raw formatter source as with `@tex` (see Section 17.3 [Raw Formatter Commands], page 141).

17.3 Raw Formatter Commands

Inside a region delineated by `@iftex` and `@end iftex`, you can embed some raw TEX commands. The Texinfo processors will ignore such a region unless TEX output is being produced. You can write the TEX commands as you would write them in a normal TEX file, except that you must replace the '\' used by TEX with an '`@`'. For example, in the

`@titlepage` section of a Texinfo file, you can use the TEX command `@vskip` to format the copyright page. (The `@titlepage` command causes Info to ignore the region automatically, as it does with the `@iftex` command.)

However, most features of plain TEX will not work within `@iftex`, as they are overridden by Texinfo features. The purpose of `@iftex` is to provide conditional processing for the Texinfo source, not provide access to underlying formatting features.

You can enter plain TEX completely, and use '\' in the TEX commands, by delineating a region with the `@tex` and `@end tex` commands. All plain TEX commands and category codes are restored within an `@tex` region. The sole exception is that the `@` character still introduces a command, so that `@end tex` can be recognized properly. As with `@iftex`, Texinfo processors will ignore such a region unless TEX output is being produced.

In complex cases, you may wish to define new TEX macros within `@tex`. You must use `\gdef` to do this, not `\def`, because `@tex` regions are processed in a TEX group.

As an example, here is a mathematical expression written in plain TEX:

```
@tex
$$ \chi^2 = \sum_{i=1}^N
          \left (y_i - (a + b x_i)
          \over \sigma_i\right)^2 $$
@end tex
```

The output of this example will appear only in a printed manual. If you are reading this in Info, you will not see the equation that appears in the printed manual. In a printed manual, the above expression looks like this:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - (a + bx_i)}{\sigma_i} \right)^2$$

Analogously, you can use `@ifhtml ... @end ifhtml` to delimit a region to be included in HTML output only, and `@html ... @end html` for a region of raw HTML.

Likewise, you can use `@ifxml ... @end ifxml` to delimit a region to be included in XML output only, and `@xml ... @end xml` for a region of raw XML.

Again likewise, you can use `@ifdocbook ... @end ifdocbook` to delimit a region to be included in Docbook output only, and `@docbook ... @end docbook` for a region of raw Docbook.

In all cases, the exception to the raw processing is that `@` is still an escape character, so the `@end` command can be recognized.

17.4 `@set`, `@clear`, and `@value`

You can direct the Texinfo formatting commands to format or ignore parts of a Texinfo file with the `@set`, `@clear`, `@ifset`, and `@ifclear` commands.

Here are brief descriptions of these commands, see the following sections for more details:

`@set` *flag* [*value*]
: Set the variable *flag*, to the optional *value* if specifed.

`@clear` *flag*
: Undefine the variable *flag*, whether or not it was previously defined.

`@ifset` *flag*
: If *flag* is set, text through the next `@end ifset` command is formatted. If *flag* is clear, text through the following `@end ifset` command is ignored.

`@ifclear` *flag*
: If *flag* is set, text through the next `@end ifclear` command is ignored. If *flag* is clear, text through the following `@end ifclear` command is formatted.

17.4.1 `@set` and `@value`

You use the `@set` command to specify a value for a flag, which is later expanded by the `@value` command.

A *flag* (aka *variable*) is an identifier. It is best to use only letters and numerals in a flag name, not '`-`' or '`_`'—they will work in some contexts, but not all, due to limitations in TEX.

The value is the remainder of the input line, and can contain anything.

Write the `@set` command like this:

```
@set foo This is a string.
```

This sets the value of the flag `foo` to "This is a string.".

The Texinfo formatters then replace an `@value{`*flag*`}` command with the string to which *flag* is set. Thus, when `foo` is set as shown above, the Texinfo formatters convert this:

```
@value{foo}
```

to this:

```
This is a string.
```

You can write an `@value` command within a paragraph; but you must write an `@set` command on a line of its own.

If you write the `@set` command like this:

```
@set foo
```

without specifying a string, the value of `foo` is the empty string.

If you clear a previously set flag with `@clear` *flag*, a subsequent `@value{flag}` command will report an error.

For example, if you set `foo` as follows:

```
@set howmuch very, very, very
```

then the formatters transform

```
It is a @value{howmuch} wet day.
```

into

```
It is a very, very, very wet day.
```

If you write

```
@clear howmuch
```

then the formatters transform

```
It is a @value{howmuch} wet day.
```

into

```
It is a {No value for "howmuch"} wet day.
```

17.4.2 `@ifset` and `@ifclear`

When a *flag* is set, the Texinfo formatting commands format text between subsequent pairs of `@ifset` *`flag`* and `@end ifset` commands. When the *flag* is cleared, the Texinfo formatting commands do *not* format the text. `@ifclear` operates analogously.

Write the conditionally formatted text between `@ifset` *`flag`* and `@end ifset` commands, like this:

```
@ifset flag
conditional-text
@end ifset
```

For example, you can create one document that has two variants, such as a manual for a 'large' and 'small' model:

```
You can use this machine to dig up shrubs
without hurting them.

@set large

@ifset large
It can also dig up fully grown trees.
@end ifset

Remember to replant promptly ...
```

In the example, the formatting commands will format the text between `@ifset large` and `@end ifset` because the `large` flag is set.

When *flag* is cleared, the Texinfo formatting commands do *not* format the text between `@ifset` *`flag`* and `@end ifset`; that text is ignored and does not appear in either printed or Info output.

For example, if you clear the flag of the preceding example by writing an `@clear large` command after the `@set large` command (but before the conditional text), then the Texinfo formatting commands ignore the text between the `@ifset large` and `@end ifset` commands. In the formatted output, that text does not appear; in both printed and Info output, you see only the lines that say, "You can use this machine to dig up shrubs without hurting them. Remember to replant promptly ...".

If a flag is cleared with an `@clear` *`flag`* command, then the formatting commands format text between subsequent pairs of `@ifclear` and `@end ifclear` commands. But if the flag is set with `@set` *`flag`*, then the formatting commands do *not* format text between an `@ifclear` and an `@end ifclear` command; rather, they ignore that text. An `@ifclear` command looks like this:

```
@ifclear flag
```

17.4.3 `@value` Example

You can use the `@value` command to minimize the number of places you need to change when you record an update to a manual. See Section C.2 [GNU Sample Texts], page 217, for the full text of an example of using this to work with Automake distributions.

This example is adapted from section "Overview" in *The GNU Make Manual*.

1. Set the flags:

```
@set EDITION 0.35 Beta
@set VERSION 3.63 Beta
@set UPDATED 14 August 1992
@set UPDATE-MONTH August 1992
```

2. Write text for the `@copying` section (see Section 3.3.1 [copying], page 32):

```
@copying
This is Edition @value{EDITION},
last updated @value{UPDATED},
of @cite{The GNU Make Manual},
for @code{make}, version @value{VERSION}.

Copyright ...

Permission is granted ...
@end copying
```

3. Write text for the title page, for people reading the printed manual:

```
@titlepage
@title GNU Make
@subtitle A Program for Directing Recompilation
@subtitle Edition @value{EDITION}, ...
@subtitle @value{UPDATE-MONTH}
@page
@insertcopying
...
@end titlepage
```

   (On a printed cover, a date listing the month and the year looks less fussy than a date listing the day as well as the month and year.)

4. Write text for the Top node, for people reading the Info file:

```
@ifnottex
@node Top
@top Make

@insertcopying
...
@end ifnottex
```

   After you format the manual, the `@value` constructs have been expanded, so the output contains text like this:

```
This is Edition 0.35 Beta, last updated 14 August 1992,
of ‘The GNU Make Manual’, for ‘make’, Version 3.63 Beta.
```

When you update the manual, you change only the values of the flags; you do not need to edit the three sections.

17.5 Conditional Nesting

Conditionals can be nested; however, the details are a little tricky. The difficulty comes with failing conditionals, such as `@ifhtml` when HTML is not being produced, where the included text is to be ignored. However, it is not to be *completely* ignored, since it is useful to have one `@ifset` inside another, for example—that is a way to include text only if two conditions are met. Here's an example:

```
@ifset somevar
@ifset anothervar
Both somevar and anothervar are set.
@end ifset
@ifclear anothervar
Somevar is set, anothervar is not.
@end ifclear
@end ifset
```

Technically, Texinfo requires that for a failing conditional, the ignored text must be properly nested with respect to that failing conditional. Unfortunately, it's not always feasible to check that *all* conditionals are properly nested, because then the processors could have to fully interpret the ignored text, which defeats the purpose of the command. Here's an example illustrating these rules:

```
@ifset a
@ifset b
@ifclear ok  - ok, ignored
@end junky   - ok, ignored
@end ifset
@c WRONG - missing @end ifset.
```

Finally, as mentioned above, all conditional commands must be on lines by themselves, with no text (even spaces) before or after. Otherwise, the processors cannot reliably determine which commands to consider for nesting purposes.

18 Internationalization

Texinfo has some support for writing in languages other than English, although this area still needs considerable work.

For a list of the various accented and special characters Texinfo supports, see Section 14.3 [Inserting Accents], page 116.

18.1 `@documentlanguage` *cc*: Set the Document Language

The `@documentlanguage` command declares the current document language. Write it on a line by itself, with a two-letter ISO-639 language code following (list is included below). If you have a multilingual document, the intent is to be able to use this command multiple times, to declare each language change. If the command is not used at all, the default is `en` for English.

At present, this command is ignored in Info and HTML output. For TEX, it causes the file '`txi-cc.tex`' to be read (if it exists). Such a file appropriately redefines the various English words used in TEX output, such as 'Chapter', 'See', and so on.

It would be good if this command also changed TEX's ideas of the current hyphenation patterns (via the TEX primitive `\language`), but this is unfortunately not currently implemented.

Hereare the valid language codes, from ISO-639.

`aa`	Afar	`ab`	Abkhazian	`af`	Afrikaans
`am`	Amharic	`ar`	Arabic	`as`	Assamese
`ay`	Aymara	`az`	Azerbaijani	`ba`	Bashkir
`be`	Byelorussian	`bg`	Bulgarian	`bh`	Bihari
`bi`	Bislama	`bn`	Bengali; Bangla	`bo`	Tibetan
`br`	Breton	`ca`	Catalan	`co`	Corsican
`cs`	Czech	`cy`	Welsh	`da`	Danish
`de`	German	`dz`	Bhutani	`el`	Greek
`en`	English	`eo`	Esperanto	`es`	Spanish
`et`	Estonian	`eu`	Basque	`fa`	Persian
`fi`	Finnish	`fj`	Fiji	`fo`	Faroese
`fr`	French	`fy`	Frisian	`ga`	Irish
`gd`	Scots Gaelic	`gl`	Galician	`gn`	Guarani
`gu`	Gujarati	`ha`	Hausa	`he`	Hebrew
`hi`	Hindi	`hr`	Croatian	`hu`	Hungarian
`hy`	Armenian	`ia`	Interlingua	`id`	Indonesian
`ie`	Interlingue	`ik`	Inupiak	`is`	Icelandic
`it`	Italian	`iu`	Inuktitut	`ja`	Japanese
`jw`	Javanese	`ka`	Georgian	`kk`	Kazakh
`kl`	Greenlandic	`km`	Cambodian	`kn`	Kannada
`ks`	Kashmiri	`ko`	Korean	`ku`	Kurdish
`ky`	Kirghiz	`la`	Latin	`ln`	Lingala
`lt`	Lithuanian	`lo`	Laothian	`lv`	Latvian, Lettish
`mg`	Malagasy	`mi`	Maori	`mk`	Macedonian
`ml`	Malayalam	`mn`	Mongolian	`mo`	Moldavian

mr	Marathi	ms	Malay	mt	Maltese
my	Burmese	na	Nauru	ne	Nepali
nl	Dutch	no	Norwegian	oc	Occitan
om	(Afan) Oromo	or	Oriya	pa	Punjabi
pl	Polish	ps	Pashto, Pushto	pt	Portuguese
qu	Quechua	rm	Rhaeto-Romance	rn	Kirundi
ro	Romanian	ru	Russian	rw	Kinyarwanda
sa	Sanskrit	sd	Sindhi	sg	Sangro
sh	Serbo-Croatian	si	Sinhalese	sk	Slovak
sl	Slovenian	sm	Samoan	sn	Shona
so	Somali	sq	Albanian	sr	Serbian
ss	Siswati	st	Sesotho	su	Sundanese
sv	Swedish	sw	Swahili	ta	Tamil
te	Telugu	tg	Tajik	th	Thai
ti	Tigrinya	tk	Turkmen	tl	Tagalog
tn	Setswana	to	Tonga	tr	Turkish
ts	Tsonga	tt	Tatar	tw	Twi
ug	Uighur	uk	Ukrainian	ur	Urdu
uz	Uzbek	vi	Vietnamese	vo	Volapuk
wo	Wolof	xh	Xhosa	yi	Yiddish
yo	Yoruba	za	Zhuang	zh	Chinese
zu	Zulu				

18.2 `@documentencoding` *enc*: Set Input Encoding

The `@documentencoding` command declares the input document encoding. Write it on a line by itself, with a valid encoding specification following.

At present, Texinfo supports only these encodings:

`US-ASCII` This has no particular effect, but it's included for completeness.

`ISO-8859-1`
`ISO-8859-15`
`ISO-8859-2`

> These specify the standard encodings for Western European (the first two) and Eastern European languages (the third), respectively. ISO 8859-15 replaces some little-used characters from 8859-1 (e.g., precomposed fractions) with more commonly needed ones, such as the Euro symbol.
>
> A full description of the encodings is beyond our scope here; one useful reference is `http://czyborra.com/charsets/iso8859.html`.

Specifying an encoding *enc* has the following effects:

In Info output, if the option '`--enable-encoding`' is given to `makeinfo`, a so-called 'Local Variables' section (see section "File Variables" in *The GNU Emacs Manual*) is output including *enc*. This allows Info readers to set the encoding appropriately.

```
Local Variables:
coding: enc
End:
```

In HTML output, a '`<meta>`' tag is output, in the '`<head>`' section of the HTML, that specifies *enc*. Web servers and browsers cooperate to use this information so the correct encoding is used to display the page, if supported by the system.

```
<meta http-equiv="Content-Type" content="text/html;
     charset=enc">
```

In all other cases, it is recognized but ignored.

19 Defining New Texinfo Commands

Texinfo provides several ways to define new commands:

- A Texinfo *macro* allows you to define a new Texinfo command as any sequence of text and/or existing commands (including other macros). The macro can have any number of *parameters*—text you supply each time you use the macro.

  Incidentally, these macros have nothing to do with the `@defmac` command, which is for documenting macros in the subject of the manual (see Section 16.1 [Def Cmd Template], page 127).
- '`@alias`' is a convenient way to define a new name for an existing command.
- '`@definfoenclose`' allows you to define new commands with customized output in the Info file.

19.1 Defining Macros

You use the Texinfo `@macro` command to define a macro, like this:

```
@macro macroname{param1, param2, ...}
text ... \param1\ ...
@end macro
```

The *parameters* *param1*, *param2*, ... correspond to arguments supplied when the macro is subsequently used in the document (described in the next section).

For a macro to work consistently with TEX, *macroname* must consist entirely of letters: no digits, hyphens, underscores, or other special characters. So, we recommend using only letters. However, `makeinfo` will accept anything except '`{}_^=`'; '`_`' and '`^`' are excluded so that macros can be called in `@math` mode without a following space (see Section 14.9 [`@math`], page 118).

If a macro needs no parameters, you can define it either with an empty list ('`@macro foo {}`') or with no braces at all ('`@macro foo`').

The definition or *body* of the macro can contain most Texinfo commands, including previously-defined macros. Not-yet-defined macro invocations are not allowed; thus, it is not possible to have mutually recursive Texinfo macros. Also, a macro definition that defines another macro does not work in TEX due to limitations in the design of `@macro`.

In the macro body, instances of a parameter name surrounded by backslashes, as in '`\param1\`' in the example above, are replaced by the corresponding argument from the macro invocation. You can use parameter names any number of times in the body, including zero.

To get a single '`\`' in the macro expansion, use '`\\`'. Any other use of '`\`' in the body yields a warning.

The newlines after the `@macro` line and before the `@end macro` line are ignored, that is, not included in the macro body. All other whitespace is treated according to the usual Texinfo rules.

To allow a macro to be used recursively, that is, in an argument to a call to itself, you must define it with '`@rmacro`', like this:

```
@rmacro rmac {arg}
a\arg\b
@end rmacro
...
@rmac{1@rmac{text}2}
```

This produces the output 'a1atextb2b'. With '`@macro`' instead of '`@rmacro`', an error message is given.

You can undefine a macro *foo* with `@unmacro` *`foo`*. It is not an error to undefine a macro that is already undefined. For example:

```
@unmacro foo
```

19.2 Invoking Macros

After a macro is defined (see the previous section), you can use (*invoke*) it in your document like this:

```
@macroname {arg1, arg2, ...}
```

and the result will be just as if you typed the body of *macroname* at that spot. For example:

```
@macro foo {p, q}
Together: \p\ & \q\.
@end macro
@foo{a, b}
```

produces:

Together: a & b.

Thus, the arguments and parameters are separated by commas and delimited by braces; any whitespace after (but not before) a comma is ignored. The braces are required in the invocation (but not the definition), even when the macro takes no arguments, consistent with all other Texinfo commands. For example:

```
@macro argless {}
No arguments here.
@end macro
@argless{}
```

produces:

No arguments here.

To insert a comma, brace, or backslash in an argument, prepend a backslash, as in

```
@macname {\\\{\}\,}
```

which will pass the (almost certainly error-producing) argument '`\{},`' to *macname*. However, commas in parameters, even if escaped by a backslash, might cause trouble in TEX.

If the macro is defined to take a single argument, and is invoked without any braces, the entire rest of the line after the macro name is supplied as the argument. For example:

```
@macro bar {p}
Twice: \p\ & \p\.
@end macro
@bar aah
```

produces:

Twice: aah & aah.

If the macro is defined to take a single argument, and is invoked with braces, the braced text is passed as the argument, regardless of commas. For example:

```
@macro bar {p}
Twice: \p\ & \p\.
@end macro
@bar{a,b}
```

produces:

Twice: a,b & a,b.

19.3 Macro Details and Caveats

Due to unavoidable limitations, certain macro-related constructs cause problems with TEX. If you get macro-related errors when producing the printed version of a manual, try expanding the macros with `makeinfo` by invoking `texi2dvi` with the '`-E`' option (see Section 20.3 [Format with texi2dvi], page 157).

- As mentioned earlier, macro names must consist entirely of letters.
- It is not advisable to redefine any TEX primitive, plain, or Texinfo command name as a macro. Unfortunately this is a very large set of names, and the possible resulting errors are unpredictable.
- All macros are expanded inside at least one TEX group. This means that `@set` and other such commands have no effect inside a macro.
- Commas in macro arguments, even if escaped by a backslash, don't always work.
- Macro arguments cannot cross lines.
- It is (usually) best to avoid comments inside macro definitions, but see the next item.
- Macros containing a command which must be on a line by itself, such as a conditional, cannot be invoked in the middle of a line. In general, the interaction of newlines in the macro definitions and invocations depends on the precise commands and context. You may be able to work around some problems with judicious use of `@c`. Suppose you define a macro that is always intended to be used on a line by itself:

  ```
  @macro linemac
  @cindex whatever
  @c
  @end macro
  ...
  foo
  @linemac
  bar
  ```

  Without the `@c`, there will be an unwanted blank line between the '`@cindex whatever`' and the '`bar`' (one newline comes from the macro definition, one from after the invocation), causing a paragraph break.

  On the other hand, you wouldn't want the `@c` if the macro was sometimes invoked in the middle of a line (the text after the invocation would be treated as a comment).

- In general, you can't arbitrarily substitute a macro call for Texinfo command arguments, even when the text is the same. It might work with some commands, it fails with others. Best not to do it at all. For instance, this fails:

```
@macro offmacro
off
@end macro
@headings @offmacro
```

  You would expect this to be equivalent to `@headings off`, but for TEXnical reasons, it fails with a mysterious error message (`Paragraph ended before @headings was complete`).

- Macros cannot define macros in the natural way. To do this, you must use conditionals and raw TEX. For example:

```
@ifnottex
@macro ctor {name, arg}
@macro \name\
something involving \arg\ somehow
@end macro
@end macro
@end ifnottex
@tex
\gdef\ctor#1{\ctorx#1,}
\gdef\ctorx#1,#2,{\def#1{something involving #2 somehow}}
@end tex
```

  The `makeinfo` implementation also has limitations:

- `@verbatim` and macros do not mix; for instance, you can't start a verbatim block inside a macro and end it outside. (See Section 10.4 [verbatim], page 87.) Starting any environment inside a macro and ending it outside may or may not work, for that matter.
- Macros that completely define macros are ok, but it's not possible to have incorrectly nested macro definitions. That is, `@macro` and `@end macro` (likewise for `@rmacro`) must be correctly paired. For example, you cannot start a macro definition within a macro, and then end the nested definition outside the macro.
- `@rmacro` is a kludge.

One more limitation is common to both implementations: white space is ignored at the beginnings of lines.

Future major revisions of Texinfo may ease some of these limitations (by introducing a new macro syntax).

19.4 '`@alias` *new=existing*'

The '`@alias`' command defines a new command to be just like an existing one. This is useful for defining additional markup names, thus preserving semantic information in the input even though the output result may be the same.

Write the '`@alias`' command on a line by itself, followed by the new command name, an equals sign, and the existing command name. Whitespace around the equals sign is ignored. Thus:

```
@alias new = existing
```

For example, if your document contains citations for both books and some other media (movies, for example), you might like to define a macro `@moviecite{}` that does the same thing as an ordinary `@cite{}` but conveys the extra semantic information as well. You'd do this as follows:

```
@alias moviecite = cite
```

Macros do not always have the same effect as aliases, due to vagaries of argument parsing. Also, aliases are much simpler to define than macros. So the command is not redundant. (It was also heavily used in the Jargon File!)

Aliases must not be recursive, directly or indirectly.

It is not advisable to redefine any TEX primitive, plain, or Texinfo command name as an alias. Unfortunately this is a very large set of names, and the possible resulting errors are completely random.

19.5 '`definfoenclose`': Customized Highlighting

A `@definfoenclose` command may be used to define a highlighting command for Info, but not for TEX. A command defined using `@definfoenclose` marks text by enclosing it in strings that precede and follow the text. You can use this to get closer control of your Info output.

Presumably, if you define a command with `@definfoenclose` for Info, you will create a corresponding command for TEX, either in '`texinfo.tex`', '`texinfo.cnf`', or within an '`@iftex`' in your document.

Write a `@definfoenclose` command on a line and follow it with three arguments separated by commas. The first argument to `@definfoenclose` is the `@`-command name (without the `@`); the second argument is the Info start delimiter string; and the third argument is the Info end delimiter string. The latter two arguments enclose the highlighted text in the Info file. A delimiter string may contain spaces. Neither the start nor end delimiter is required. If you do not want a start delimiter but do want an end delimiter, you must follow the command name with two commas in a row; otherwise, the Info formatting commands will naturally misinterpret the end delimiter string you intended as the start delimiter string.

If you do a `@definfoenclose` on the name of a pre-defined macro (such as `@emph`, `@strong`, `@t`, or `@i`), the enclosure definition will override the built-in definition.

An enclosure command defined this way takes one argument in braces; this is intended for new markup commands (see Chapter 9 [Marking Text], page 74).

For example, you can write:

```
@definfoenclose phoo,//,\\
```

near the beginning of a Texinfo file to define `@phoo` as an Info formatting command that inserts '//' before and '\\' after the argument to `@phoo`. You can then write `@phoo{bar}` wherever you want '//bar\\' highlighted in Info.

Also, for TEX formatting, you could write

```
@iftex
@global@let@phoo=@i
@end iftex
```

to define `@phoo` as a command that causes TEX to typeset the argument to `@phoo` in italics.

Each definition applies to its own formatter: one for TEX, the other for `texinfo-format-buffer` or `texinfo-format-region`. The `@definfoenclose` command need not be within '`@ifinfo`', but the raw TEX commands do need to be in '`@iftex`'.

Here is another example: write

```
@definfoenclose headword, , :
```

near the beginning of the file, to define `@headword` as an Info formatting command that inserts nothing before and a colon after the argument to `@headword`.

'`@definfoenclose`' definitions must not be recursive, directly or indirectly.

20 Formatting and Printing Hardcopy

There are three major shell commands for making a printed manual from a Texinfo file: one for converting the Texinfo file into a file that will be printed, a second for sorting indices, and a third 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 GNU Emacs.

If you are using GNU Emacs, you can use commands provided by Texinfo mode instead of shell commands. In addition to the three commands 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.

20.1 Use TEX

The typesetting program called TEX is used for formatting a Texinfo file. TEX is a very powerful typesetting program and, if used correctly, does an exceptionally good job. (See Section 20.16 [How to Obtain TEX], page 166, for information on how to obtain TEX.)

The standalone `makeinfo` program and Emacs functions `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 Section 21.1 [Creating an Info File], page 168).

20.2 Format with `tex` and `texindex`

You can format the Texinfo file with the shell command `tex` followed by the name of the Texinfo file. For example:

```
tex foo.texi
```

TEX will produce a *DVI file* as well as several auxiliary files containing information for indices, cross references, etc. The DVI file (for *DeVice Independent* file) can be printed on virtually any device (see the following sections).

The `tex` formatting command itself does not sort the indices; it writes an output file of unsorted index data. To generate a printed index after running the `tex` command, you first need a sorted index to work from. The `texindex` command sorts indices. (The source file '`texindex.c`' comes as part of the standard Texinfo distribution, among other places.) (`texi2dvi` runs `tex` and `texindex` as necessary.)

The `tex` formatting command outputs unsorted index files under names that obey a standard convention: the name of your main input file with any '`.tex`' (or similar, see section "tex invocation" in *Web2c*) extension removed, followed by the two letter names of indices. 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`.

Instead of specifying all the unsorted index file names explicitly, you can use '`??`' as shell wildcards 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 looks for a file with that name (see Section 4.1 [Printing Indices & Menus], page 45). `texindex` does not alter the raw index output file.

After you have sorted the indices, you need to rerun `tex` on the Texinfo file. This regenerates the DVI file, this time with up-to-date index entries.

Finally, you may need to run `tex` one more time, to get the page numbers in the cross-references correct.

To summarize, this is a five step process:

1. Run `tex` on your Texinfo file. This generates a DVI file (with undefined cross-references and no indices), and the raw index files (with two letter extensions).
2. Run `texindex` on the raw index files. This creates the corresponding sorted index files (with three letter extensions).
3. Run `tex` again on your Texinfo file. This regenerates the DVI file, this time with indices and defined cross-references, but with page numbers for the cross-references from last time, generally incorrect.
4. Sort the indices again, with `texindex`.
5. Run `tex` one last time. This time the correct page numbers are written for the cross-references.

Alternatively, it's a one-step process: run `texi2dvi` (see Section 20.3 [Format with texi2dvi], page 157).

You need not run `texindex` each time after you run `tex`. If you do not, 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.

Sometimes you may wish to print a document while you know it is incomplete, or to print just one chapter of a document. In that case, the usual auxiliary files that TEX creates and warnings TEX gives when cross-references are not satisfied are just nuisances. You can avoid them with the `@novalidate` command, which you must give *before* the `@setfilename` command (see Section 3.2.3 [`@setfilename`], page 30). Thus, the beginning of your file would look approximately like this:

```
\input texinfo
@novalidate
@setfilename myfile.info
...
```

`@novalidate` also turns off validation in `makeinfo`, just like its `--no-validate` option (see Section 21.1.4 [Pointer Validation], page 172).

20.3 Format with `texi2dvi`

The `texi2dvi` command automatically runs both TEX and `texindex` as many times as necessary to produce a DVI file with sorted indices and all cross-references resolved. It

is therefore simpler than manually executing the `tex`—`texindex`—`tex`—`tex` sequence described in the previous section.

To run `texi2dvi` on an input file '`foo.texi`', do this (where '`prompt$` ' is your shell prompt):

```
prompt$ texi2dvi foo.texi
```

As shown in this example, the input filenames to `texi2dvi` must include any extension ('`.texi`', '`.texinfo`', etc.). Under MS-DOS and perhaps in other circumstances, you may need to run '`sh texi2dvi foo.texi`' instead of relying on the operating system to invoke the shell on the '`texi2dvi`' script.

Perhaps the most useful option to `texi2dvi` is '`--command=`*cmd*'. This inserts *cmd* on a line by itself after the `@setfilename` in a temporary copy of the input file before running TEX. With this, you can specify different printing formats, such as `@smallbook` (see Section 20.11 [smallbook], page 164), `@afourpaper` (see Section 20.12 [A4 Paper], page 164), or `@pagesizes` (see Section 20.13 [pagesizes], page 165), without actually changing the document source. (You can also do this on a site-wide basis with '`texinfo.cnf`'; see Section 20.9 [Preparing for TEX], page 162).

With the '`--pdf`' option, `texi2dvi` produces PDF output instead of DVI (see Section 20.15 [PDF Output], page 166), by running `pdftex` instead of `tex`. Alternatively, the command `texi2pdf` is an abbreviation for running '`texi2dvi --pdf`'.

`texi2dvi` can also be used to process LATEX files; simply run '`texi2dvi filename.ext`'.

`texi2dvi` will use `etex` (or `pdfetex`) if they are available; these extended versions of TEX are not required, and the DVI (or PDF) output is identical, but they simplify the TEX programming in some cases, and provide additional tracing information when debugging '`texinfo.tex`'.

For a list of other options, run '`texi2dvi --help`'.

20.4 Shell Print Using `lpr -d`

The precise command to print a DVI file depends on your system installation. Two common ones are '`dvips foo.dvi -o`' and '`lpr -d foo.dvi`'.

For example, the following commands will (perhaps) suffice to sort the indices, format, and print the *Bison Manual*:

```
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.)

Using the `texi2dvi` shell script (see the previous section):

```
texi2dvi bison.texinfo
lpr -d bison.dvi
# or perhaps dvips bison.dvi -o
```

`lpr` is a standard program on Unix systems, but it is usually absent on MS-DOS/MS-Windows. Some network packages come with a program named `lpr`, but these are usually

limited to sending files to a print server over the network, and generally don't support the '`-d`' option. If you are unfortunate enough to work on one of these systems, you have several alternative ways of printing DVI files:

- Find and install a Unix-like `lpr` program, or its clone. If you can do that, you will be able to print DVI files just like described above.
- Send the DVI files to a network printer queue for DVI files. Some network printers have special queues for printing DVI files. You should be able to set up your network software to send files to that queue. In some cases, the version of `lpr` which comes with your network software will have a special option to send a file to specific queues, like this:

  ```
  lpr -Qdvi -hprint.server.domain bison.dvi
  ```

- Convert the DVI file to a Postscript or PCL file and send it to your local printer. See section "Invoking Dvips" in *Dvips*, and the man pages for `dvilj`, for detailed description of these tools. Once the DVI file is converted to the format your local printer understands directly, just send it to the appropriate port, usually '`PRN`'.

20.5 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 Chapter 20 [Format and Print Hardcopy], page 156, 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. For example, here is how to use `texi2dvi` to format and print *Using and Porting GNU CC* from a shell within Emacs:

```
texi2dvi gcc.texinfo
lpr -d gcc.dvi
```

See the next section for more information about formatting and printing in Texinfo mode.

20.6 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 the display of the buffer in which the operations occur.

C-c C-t C-b
M-x texinfo-tex-buffer
: Run `texi2dvi` on the current buffer.

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

C-c C-t C-i
M-x texinfo-texindex
: Sort the indices of a Texinfo file formatted with `texinfo-tex-region`.

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 tex-show-print-queue
: Show the print queue.

C-c C-t C-d
M-x texinfo-delete-from-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 tex-kill-job
: Kill the currently running TEX job started by either `texinfo-tex-region` or `texinfo-tex-buffer`, or any other process running in the Texinfo shell buffer.

C-c C-t C-x
M-x texinfo-quit-job
: Quit a TEX formatting job that has stopped because of an error by sending an ⟨x⟩ to it. When you do this, TEX preserves a record of what it did in a '`.log`' file.

C-c C-t C-l
M-x tex-recenter-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 `texi2dvi` on the buffer.
`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 '`*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 '`*tex-shell*`' buffer, and you can switch to and from and use the '`*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-texi2dvi-command`	`"texi2dvi"`
`texinfo-tex-command`	`"tex"`
`texinfo-texindex-command`	`"texindex"`
`texinfo-delete-from-print-queue-command`	`"lprm"`
`texinfo-tex-trailer`	`"@bye"`
`tex-start-of-header`	`"%**start"`
`tex-end-of-header`	`"%**end"`
`tex-dvi-print-command`	`"lpr -d"`
`tex-show-queue-command`	`"lpq"`

You can change the values of these variables 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*).

Beginning with version 20, GNU Emacs offers a user-friendly interface, called *Customize*, for changing values of user-definable variables. See section "Easy Customization Interface" in *The GNU Emacs Manual*, for more details about this. The Texinfo variables can be found in the '`Development/Docs/Texinfo`' group, once you invoke the *M-x customize* command.

20.7 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` or `texi2dvi` commands as a `compile-command` and have Emacs run it 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 could put the following:

```
Local Variables:
compile-command: "texi2dvi gdb.texinfo"
End:
```

This technique is most often used by programmers who also compile programs this way; see section "Compilation" in *The GNU Emacs Manual*.

20.8 TEX Formatting Requirements Summary

Every Texinfo file that is to be input to TEX must begin with a `\input` command and must contain an `@setfilename` command:

```
\input texinfo
@setfilename arg-not-used-by-TeX
```

The first command instructs TEX to load the macros it needs to process a Texinfo file and the second command opens auxiliary files.

Every Texinfo file must end with a line that terminates TEX's processing and forces out unfinished pages:

```
@bye
```

Strictly speaking, these lines are all a Texinfo file needs to be processed successfully by TEX.

Usually, however, the beginning includes an `@settitle` command to define the title of the printed manual, 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. (And of course most manuals contain a body of text as well.)

For more information, see:

- Section 3.2.4 [`@settitle`], page 31.
- Section 3.7.2 [`@setchapternewpage`], page 41.
- Appendix E [Page Headings], page 226.
- Section 3.4 [Titlepage & Copyright Page], page 33.
- Section 4.1 [Printing Indices & Menus], page 45.
- Section 3.5 [Contents], page 38.

20.9 Preparing for TEX

TEX needs to know where to find the '`texinfo.tex`' file that the '`\input texinfo`' command on the first line reads. The '`texinfo.tex`' file tells TEX how to handle `@`-commands; it is included in all standard GNU distributions. The latest version is always available from the Texinfo source repository:

`http://savannah.gnu.org/cgi-bin/viewcvs/texinfo/texinfo/doc/texinfo.tex?rev=HEAD`

Usually, the installer has put the '`texinfo.tex`' file in the default directory that contains TEX macros when GNU Texinfo, Emacs or other GNU software is installed. In this case, TEX will find the file and you do not need to do anything special. If this has not been done, you can put '`texinfo.tex`' in the current directory when you run TEX, and TEX will find it there.

Also, you should install '`epsf.tex`', if it is not already installed from another distribution. More details are at the end of the description of the `@image` command (see Section 12.2 [Images], page 104).

Likewise for '`pdfcolor.tex`', if it is not already installed and you use pdftex.

Optionally, you may create an additional '`texinfo.cnf`', and install it as well. This file is read by TEX when the `@setfilename` command is executed (see Section 3.2.3 [`@setfilename`], page 30). You can put any commands you like there, according to local site-wide conventions. They will be read by TEX when processing any Texinfo document. For example, if '`texinfo.cnf`' contains the line '`@afourpaper`' (see Section 20.12 [A4 Paper], page 164), then all Texinfo documents will be processed with that page size in effect. If you have nothing to put in '`texinfo.cnf`', you do not need to create it.

If neither of the above locations for these system files suffice for you, you can specify the directories explicitly. For '`texinfo.tex`', you can do this by writing the complete path for the file after the `\input` command. Another way, that works for both '`texinfo.tex`' and '`texinfo.cnf`' (and any other file TEX might read), is to set the `TEXINPUTS` environment variable in your '`.cshrc`' or '`.profile`' file.

Which you use of '`.cshrc`' or '`.profile`' depends on whether you use a Bourne shell-compatible (`sh`, `bash`, `ksh`, ...) or C shell-compatible (`csh`, `tcsh`) command interpreter. The latter read the '`.cshrc`' file for initialization information, and the former read '`.profile`'.

In a '`.cshrc`' file, you could use the following `csh` command sequence:

```
setenv TEXINPUTS .:/home/me/mylib:
```

In a '`.profile`' file, you could use the following `sh` command sequence:

```
TEXINPUTS=.:/home/me/mylib:
export TEXINPUTS
```

On MS-DOS/MS-Windows, you would say it like this[1]:

```
set TEXINPUTS=.;d:/home/me/mylib;c:
```

It is customary for DOS/Windows users to put such commands in the '`autoexec.bat`' file, or in the Windows Registry.

These settings would cause TEX to look for '`\input`' file first in the current directory, indicated by the '`.`', then in a hypothetical user '`me`''s '`mylib`' directory, and finally in the system directories. (A leading, trailing, or doubled '`:`' indicates searching the system directories at that point.)

Finally, you may wish to dump a '`.fmt`' file (see section "Memory dumps" in *Web2c*) so that TEX can load Texinfo faster. (The disadvantage is that then updating '`texinfo.tex`' requires redumping.) You can do this by running this command, assuming '`epsf.tex`' is findable by TEX:

```
initex texinfo @dump
```

(`dump` is a TEX primitive.) Then, move '`texinfo.fmt`' to wherever your `.fmt` files are found; typically, this will be in the subdirectory '`web2c`' of your TEX installation.

20.10 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". '`@hbox`' is a TEX primitive not needed in the Texinfo language.)

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 considered hyphenation. See Section F.3 [Catching Errors with TEX Formatting], page 231, for more information about typesetting errors.

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.

If you have many overfull boxes and/or an antipathy to rewriting, you can coerce TEX into greatly increasing the allowable interword spacing, thus (if you're lucky) avoiding many of the bad line breaks, like this:

[1] Note the use of the '`;`' character, instead of '`:`', as directory separator on these systems.

```
@tex
\global\emergencystretch = .9\hsize
@end tex
```

(You should adjust the fraction as needed.) This huge value for `\emergencystretch` cannot be the default, since then the typeset output would generally be of noticeably lower quality; the default is '`.15\hsize`'. `\hsize` is the TEX dimension containing the current line width.

For what overfull boxes you have, however, TEX will print a large, ugly, black rectangle beside the line that contains the overfull hbox unless told otherwise. 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, write the following in the beginning of the Texinfo file on a line of its own, before the `@titlepage` command:

```
@finalout
```

20.11 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 title page:

```
@smallbook
```

(Since many books are 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.)

If you write the `@smallbook` command between the start-of-header and end-of-header lines, the Texinfo mode TEX region formatting command, `texinfo-tex-region`, will format the region in "small" book size (see Section 3.2.2 [Start of Header], page 30).

See Section 10.7 [small], page 89, for information about commands that make it easier to produce examples for a smaller manual.

See Section 20.3 [Format with texi2dvi], page 157, and Section 20.9 [Preparing for TEX], page 162, for other ways to format with `@smallbook` that do not require changing the source file.

20.12 Printing on A4 Paper

You can tell TEX to format a document for printing on European size A4 paper (or A5) with the `@afourpaper` (or `@afivepaper`) command. Write the command on a line by itself near the beginning of the Texinfo file, before the title page. For example, this is how you would write the header for this manual:

```
\input texinfo    @c -*-texinfo-*-
@c %**start of header
@setfilename texinfo
@settitle Texinfo
@afourpaper
@c %**end of header
```

See Section 20.3 [Format with texi2dvi], page 157, and Section 20.9 [Preparing for TEX], page 162, for other ways to format for different paper sizes that do not require changing the source file.

You may or may not prefer the formatting that results from the command `@afourlatex`. There's also `@afourwide` for A4 paper in wide format.

20.13 `@pagesizes` [*width*][, *height*]: Custom Page Sizes

You can explicitly specify the height and (optionally) width of the main text area on the page with the `@pagesizes` command. Write this on a line by itself near the beginning of the Texinfo file, before the title page. The height comes first, then the width if desired, separated by a comma. Examples:

```
@pagesizes 200mm,150mm
```

and

```
@pagesizes 11.5in
```

This would be reasonable for printing on B5-size paper. To emphasize, this command specifies the size of the *text area*, not the size of the paper (which is 250 mm by 177 mm for B5, 14 in by 8.5 in for legal).

To make more elaborate changes, such as changing any of the page margins, you must define a new command in '`texinfo.tex`' (or '`texinfo.cnf`', see Section 20.9 [Preparing for TEX], page 162).

See Section 20.3 [Format with texi2dvi], page 157, and Section 20.9 [Preparing for TEX], page 162, for other ways to specify `@pagesizes` that do not require changing the source file.

`@pagesizes` is ignored by `makeinfo`.

20.14 Cropmarks and Magnification

You can (attempt to) direct TEX to print cropmarks at the corners of pages with the `@cropmarks` command. Write the `@cropmarks` command on a line by itself between `@iftex` and `@end iftex` lines near the beginning of the Texinfo file, before the title page, like this:

```
@iftex
@cropmarks
@end iftex
```

This command is mainly for printers that typeset several pages on one sheet of film; but you can attempt to use it to mark the corners of a book set to 7 by 9.25 inches with the `@smallbook` command. (Printers will not produce cropmarks for regular sized output that is printed on regular sized paper.) Since different printing machines work in different ways, you should explore the use of this command with a spirit of adventure. You may have to redefine the command in '`texinfo.tex`'.

You can attempt to direct TEX to typeset pages larger or smaller than usual with the `\mag` TEX command. Everything that is typeset is scaled proportionally larger or smaller. (`\mag` stands for "magnification".) This is *not* a Texinfo `@`-command, but is a plain TEX command that is prefixed with a backslash. You have to write this command between `@tex` and `@end tex` (see Section 17.3 [Raw Formatter Commands], page 141).

Follow the `\mag` command with an '`=`' and then a number that is 1000 times the magnification you desire. For example, to print pages at 1.2 normal size, write the following near the beginning of the Texinfo file, before the title page:

```
@tex
\mag=1200
@end tex
```

With some printing technologies, you can print normal-sized copies that look better than usual by giving a larger-than-normal master to your print shop. They do the reduction, thus effectively increasing the resolution.

Depending on your system, DVI files prepared with a nonstandard-`\mag` may not print or may print only with certain magnifications. Be prepared to experiment.

20.15 PDF Output

The simplest way to generate PDF output from Texinfo source is to run the convenience script `texi2pdf`; this simply executes the `texi2dvi` script with the '`--pdf`' option (see Section 20.3 [Format with texi2dvi], page 157). If for some reason you want to process by hand, simply run the `pdftex` program instead of plain `tex`. That is, run '`pdftex foo.texi`' instead of '`tex foo.texi`'.

PDF stands for 'Portable Document Format'. It was invented by Adobe Systems some years ago for document interchange, based on their PostScript language. Related links:

- GNU GV, a Ghostscript-based PDF reader (`http://www.foolabs.com/xpdf/`). (It can also preview PostScript documents.)
- A freely available standalone PDF reader (`http://www.foolabs.com/xpdf/`) for the X window system.
- PDF definition (`http://partners.adobe.com/asn/acrobat/sdk/public/docs/`).

At present, Texinfo does not provide '`@ifpdf`' or '`@pdf`' commands as for the other output formats, since PDF documents contain many internal links that would be hard or impossible to get right at the Texinfo source level.

PDF files require special software to be displayed, unlike the plain ASCII formats (Info, HTML) that Texinfo supports. They also tend to be much larger than the DVI files output by TEX by default. Nevertheless, a PDF file does define an actual typeset document in a self-contained file, so it has its place.

20.16 How to Obtain TEX

TEX is freely redistributable. You can obtain TEX for Unix systems via anonymous ftp or on physical media. The core material consists of the Web2c TEX distribution (`http://tug.org/web2c`).

Instructions for retrieval by anonymous ftp and information on other available distributions: `http://tug.org/unixtex.ftp`.

The Free Software Foundation provides a core distribution on its Source Code CD-ROM suitable for printing Texinfo manuals. To order it, contact:

Free Software Foundation, Inc.
59 Temple Place Suite 330
Boston, MA 02111-1307
USA
Telephone: +1-617-542-5942
Fax: (including Japan) +1-617-542-2652
Free Dial Fax (in Japan):
0031-13-2473 (KDD)
0066-3382-0158 (IDC)
Electronic mail: `gnu@gnu.org`

Many other TEX distributions are available; see `http://tug.org/`.

21 Creating and Installing Info Files

This chapter describes how to create and install Info files. See Section 1.4 [Info Files], page 5, for general information about the file format itself.

21.1 Creating an Info File

`makeinfo` is a program that converts a Texinfo file into an Info file, HTML file, or plain text. `texinfo-format-region` and `texinfo-format-buffer` are GNU Emacs functions that convert Texinfo to Info.

For information on installing the Info file in the Info system, see Section 21.2 [Installing an Info File], page 176.

21.1.1 `makeinfo` Preferred

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. We recommend it. `makeinfo` is a C program that is independent of Emacs. You do not need to run Emacs to use `makeinfo`, which means you can use `makeinfo` on machines that are too small to run Emacs. You can run `makeinfo` in any one of three ways: from an operating system shell, from a shell inside Emacs, or by typing the *C-c C-m C-r* or the *C-c C-m C-b* command in Texinfo mode in Emacs.

The `texinfo-format-region` and the `texinfo-format-buffer` commands are useful if you cannot run `makeinfo`. Also, in some circumstances, they format short regions or buffers more quickly than `makeinfo`.

21.1.2 Running `makeinfo` from a Shell

To create an Info file from a Texinfo file, invoke `makeinfo` followed by the name of the Texinfo file. Thus, to create the Info file for Bison, type the following to the shell:

```
makeinfo bison.texinfo
```

(You can run a shell inside Emacs by typing *M-x shell*.)

`makeinfo` has many options to control its actions and output; see the next section.

21.1.3 Options for `makeinfo`

The `makeinfo` program accepts many options. Perhaps the most commonly needed are those that change the output format. By default, `makeinfo` outputs Info files.

Each command line option is a word preceded by '`--`' or a letter preceded by '`-`'. You can use abbreviations for the long option names as long as they are unique.

For example, you could use the following shell command to create an Info file for '`bison.texinfo`' in which each line is filled to only 68 columns:

```
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 and would also set the fill column to 70.

The options are:

`-D var` Cause the variable *var* to be defined. This is equivalent to `@set var` in the Texinfo file (see Section 17.4 [set clear value], page 142).

`--commands-in-node-names`
: Allow `@`-commands in node names. This is not recommended, as it can probably never be implemented in TEX. It also makes `makeinfo` much slower. Also, this option is ignored when '`--no-validate`' is used. See Section 21.1.4 [Pointer Validation], page 172, for more details.

`--css-include=file`
: Include the contents of *file*, which should contain cascading style sheets specifications, in the '`<style>`' block of the HTML output. See Section 22.3 [HTML CSS], page 183. If *file* is '`-`', read standard input.

`--docbook`
: Generate Docbook output rather than Info.

`--enable-encoding`
: Output accented and special characters in Info or plain text output based on '`@documentencoding`'. See Section 18.2 [`documentencoding`], page 148, and Section 14.3 [Inserting Accents], page 116.

`--error-limit=limit`
`-e limit` Set the maximum number of errors that `makeinfo` will report before exiting (on the assumption that continuing would be useless); default 100.

`--fill-column=width`
`-f 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.) The default value is 72. Ignored with '`--html`'.

`--footnote-style=style`
`-s style` Set the footnote style to *style*, either '`end`' for the end node style (the default) 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 (see Section 12.3 [Footnotes], page 106). 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. Ignored with '`--html`'.

`--force`
`-F` Ordinarily, if the input file has errors, the output files are not created. With this option, they are preserved.

`--help`
`-h` Print a usage message listing all available options, then exit successfully.

`--html` Generate HTML output rather than Info. See Chapter 22 [Generating HTML], page 182. By default, the HTML output is split into one output file per Texinfo source node, and the split output is written into a subdirectory with the name of the top-level info file.

`-I` *dir*
: Append *dir* to the directory search list for finding files that are included using the `@include` command. By default, `makeinfo` searches only the current directory. If *dir* is not given, the current directory '`.`' is appended. Note that *dir* can actually be a list of several directories separated by the usual path separator character ('`:`' on Unix, '`;`' on MS-DOS/MS-Windows).

`--ifdocbook`
`--ifhtml`
`--ifinfo`
`--ifplaintext`
`--iftex`
`--ifxml`
: For the specified format, process '`@if`*format*' and '`@`*format*' commands even if not generating the given output format. For instance, if '`--iftex`' is specified, then '`@iftex`' and '`@tex`' blocks will be read.

`--macro-expand=`*file*
`-E` *file*
: Output the Texinfo source with all the macros expanded to the named file. Normally, the results of macro expansion are used internally by `makeinfo` and then discarded. This option is used by `texi2dvi`.

`--no-headers`
`--plaintext`
: Do not include menus or node separator lines in the output, and implicitly '`--enable-encoding`' (see above). This results in a simple plain text file that you can (for example) send in email without complications, or include in a distribution (as in an '`INSTALL`' file).

  For HTML output, likewise omit menus. And if '`--no-split`' is also specified, do not include a navigation links at the top of each node (these are never included in the default case of split output). See Chapter 22 [Generating HTML], page 182.

  In both cases, ignore `@setfilename` and write to standard output by default—can be overridden with '`-o`'.

`--no-ifdocbook`
`--no-ifhtml`
`--no-ifinfo`
`--no-ifplaintext`
`--no-iftex`
`--no-ifxml`
: Do not process '`@if`*format*' and '`@`*format*' commands, and do process '`@ifnot`*format*', even if generating the given format. For instance, if '`--no-ifhtml`' is specified, then '`@ifhtml`' and '`@html`' blocks will not be read, and '`@ifnothtml`' blocks will be.

`--no-number-footnotes`
: Suppress automatic footnote numbering. By default, `makeinfo` numbers each footnote sequentially in a single node, resetting the current footnote number to 1 at the start of each node.

`--no-number-sections`
: Do not output chapter, section, and appendix numbers. You need to specify this if your manual is not hierarchically-structured.

`--no-split`
: Suppress the splitting stage of `makeinfo`. By default, large output files (where the size is greater than 70k bytes) are split into smaller subfiles. For Info output, each one is approximately 50k bytes. For HTML output, each file contains one node (see Chapter 22 [Generating HTML], page 182).

`--no-pointer-validate`
`--no-validate`
: Suppress the pointer-validation phase of `makeinfo`—a dangerous thing to do. This can also be done with the `@novalidate` command (see Section 20.1 [Use TEX], page 156). Normally, after a Texinfo file is processed, some consistency checks are made to ensure that cross references can be resolved, etc. See Section 21.1.4 [Pointer Validation], page 172.

`--no-warn`
: Suppress warning messages (but *not* error messages).

`--number-sections`
: Output chapter, section, and appendix numbers as in printed manuals. This is the default. It works only with hierarchically-structured manuals.

`--output=`*file*
`-o` *file*
: Specify that the output should be directed to *file* and not to the file name specified in the `@setfilename` command found in the Texinfo source (see Section 3.2.3 [setfilename], page 30). If *file* is '`-`', output goes to standard output and '`--no-split`' is implied. For split HTML output, *file* is the name for the directory into which all HTML nodes are written (see Chapter 22 [Generating HTML], page 182).

`-P` *dir*
: Prepend *dir* to the directory search list for `@include`. If *dir* is not given, the current directory '`.`' is prepended. See '`-I`' for more details.

`--paragraph-indent=`*indent*
`-p` *indent*
: Set the paragraph indentation style to *indent*. The value set by this option overrides the value set in a Texinfo file by an `@paragraphindent` command (see Section 3.7.3 [paragraphindent], page 42). The value of *indent* is interpreted as follows:

  '`asis`'
  : Preserve any existing indentation at the starts of paragraphs.

  '`0`' or '`none`'
  : Delete any existing indentation.

  num
  : Indent each paragraph by *num* spaces.

`--reference-limit=`*limit*
`-r` *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. The default is 1000.

`--split-size=`*num*
: Keep Info files to at most *num* characters; default is 300,000.

`-U` *var*
: Cause *var* to be undefined. This is equivalent to `@clear` *var* in the Texinfo file (see Section 17.4 [set clear value], page 142).

`--verbose`
: Cause `makeinfo` to display messages saying what it is doing. Normally, `makeinfo` only outputs messages if there are errors or warnings.

`--version`
`-V`
: Print the version number, then exit successfully.

`--xml`
: Generate XML output rather than Info.

`makeinfo` also reads the environment variable `TEXINFO_OUTPUT_FORMAT` to determine the output format, if not overridden by a command line option. The possible values are:

```
docbook  html  info  plaintext  xml
```

If not set, Info output is the default.

21.1.4 Pointer Validation

If you do not suppress pointer validation with the '`--no-validate`' option or the `@novalidate` command in the source file (see Section 20.1 [Use TEX], page 156), `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 node pointed to by the 'Previous' field must have a 'Next' field which points back to this node.
3. Every node except the 'Top' node must have an 'Up' pointer.
4. The node referenced by an 'Up' pointer must itself reference the current node through a menu item, unless the node referenced by 'Up' has the form '(*file*)'.
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.
6. Every node except 'Top' should be referenced by at least one other node, either via the 'Previous' or 'Next' links, or via a menu or a cross-reference.

Some Texinfo documents might fail during the validation phase because they use commands like `@value` and `@definfoenclose` in node definitions and cross-references inconsistently. (Your best bet is to avoid using `@`-commands in node names.) Consider the following example:

```
@set nodename Node 1

@node @value{nodename}, Node 2, Top, Top

This is node 1.

@node Node 2, , Node 1, Top

This is node 2.
```

Here, the node "Node 1" was referenced both verbatim and through `@value`.

By default, `makeinfo` fails such cases, because node names are not fully expanded until they are written to the output file. You should always try to reference nodes consistently; e.g., in the above example, the second `@node` line should have also used `@value`. However, if, for some reason, you *must* reference node names inconsistently, and `makeinfo` fails to validate the file, you can use the '`--commands-in-node-names`' option to force `makeinfo` to perform the expensive expansion of all node names it finds in the document. This might considerably slow down the program, though; twofold increase in conversion time was measured for large documents such as the Jargon file.

The support for `@`-commands in `@node` directives is not general enough to be freely used. For example, if the example above redefined `nodename` somewhere in the document, `makeinfo` will fail to convert it, even if invoked with the '`--commands-in-node-names`' option.

'`--commands-in-node-names`' has no effect if the '`--no-validate`' option is given.

21.1.5 Running `makeinfo` Within Emacs

You can run `makeinfo` in GNU Emacs Texinfo mode 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 `makeinfo-region` the output goes to a temporary buffer. When you invoke `makeinfo-buffer` output goes to the file set with `@setfilename` (see Section 3.2.3 [setfilename], page 30).

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 (from `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 20.6 [Texinfo Mode Printing], page 159.)

You can specify options for `makeinfo` by setting the `makeinfo-options` variable with either the *M-x customize* or the *M-x set-variable* command, or by setting the variable in your '`.emacs`' initialization file.

For example, you could write the following in your '`.emacs`' file:

```
(setq makeinfo-options
     "--paragraph-indent=0 --no-split
      --fill-column=70 --verbose")
```

For more information, see Section 21.1.3 [Options for `makeinfo`], page 168, as well as "Easy Customization Interface," "Examining and Setting Variables," and "Init File" in *The GNU Emacs Manual*.

21.1.6 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 can provide you with further help in finding formatting errors. These procedures are described in an appendix; see Appendix F [Catching Mistakes], page 230. However, the `makeinfo` program is often faster and provides better error checking (see Section 21.1.5 [makeinfo in Emacs], page 173).

21.1.7 Batch Formatting

You can format Texinfo files for Info using `batch-texinfo-format` and Emacs Batch mode. You can run Emacs in Batch mode from any shell, including a shell inside of Emacs. (See section "Command Arguments" in *The GNU Emacs Manual*.)

Here is a shell command to format all the files that end in '`.texinfo`' in the current directory:

```
emacs -batch -funcall batch-texinfo-format *.texinfo
```

Emacs processes all the files listed on the command line, even if an error occurs while attempting to format some of them.

Run `batch-texinfo-format` only with Emacs in Batch mode as shown; it is not interactive. It kills the Batch mode Emacs on completion.

`batch-texinfo-format` is convenient if you lack `makeinfo` and want to format several Texinfo files at once. When you use Batch mode, you create a new Emacs process. This frees your current Emacs, so you can continue working in it. (When you run `texinfo-format-region` or `texinfo-format-buffer`, you cannot use that Emacs for anything else until the command finishes.)

21.1.8 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 300,000 bytes, `texinfo-format-buffer` and `makeinfo` split the large Info file into shorter *indirect* subfiles of about 300,000 bytes each. Big files are split into smaller files so that Emacs does not need 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 D [Include Files], page 222, 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 name 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 the file '`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:`’.

In the list of indirect files, the number following the file name records the cumulative number of bytes in the preceding indirect files, not counting the file list itself, the tag table, or the permissions text in each file. In the tag table, the number following the node name records the location of the beginning of the node, in bytes from the beginning of the (unsplit) output.

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 F.6.1 [Using Info-validate], page 235.

21.2 Installing an Info File

Info files are usually kept in the ‘`info`’ directory. You can read Info files using the standalone Info program or the Info reader built into Emacs. (See Info file ‘`info`’, node ‘`Top`’, for an introduction to Info.)

21.2.1 The Directory File ‘`dir`’

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`’. (You can find the location of this file within Emacs by typing *`C-h i`* to enter Info and then typing *`C-x C-f`* to see the pathname to the ‘`info`’ directory.)

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 need to specify the 'Top' node, since Info goes to the 'Top' node if no node name is mentioned. See Section 7.6 [Nodes in Other Info Files], page 62.)

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 line for the 'Top' node of the Emacs manual looks like this in Info:

```
File: emacs  Node: Top, Up: (DIR), Next: Distrib
```

In this case, the '`dir`' file name is written in upper case letters—it can be written in either upper or lower case. This is not true in general, it is a special case for '`dir`'.

21.2.2 Listing a New Info File

To add a new Info file to your system, you must write a menu entry to add to the menu in the '`dir`' file in the '`info`' directory. For example, if you were adding documentation for GDB, you would write the following new entry:

```
* GDB: (gdb).           The source-level C debugger.
```

The first part of the menu entry is the menu entry name, followed by a colon. The second part is the name of the Info file, in parentheses, followed by a period. The third part is the description.

The name of an Info file often has a '`.info`' extension. Thus, the Info file for GDB might be called either '`gdb`' or '`gdb.info`'. The Info reader programs automatically try the file name both with and without '`.info`'[1]; so it is better to avoid clutter and not to write '`.info`' explicitly in the menu entry. For example, the GDB menu entry should use just '`gdb`' for the file name, not '`gdb.info`'.

21.2.3 Info Files in Other Directories

If an Info file is not in the '`info`' directory, there are three ways to specify its location:

1. Write the pathname in the '`dir`' file as the second part of the menu.
2. If you are using Emacs, list the name of the file in a second '`dir`' file, in its directory; and then add the name of that directory to the `Info-directory-list` variable in your personal or site initialization file.

   This variable tells Emacs where to look for '`dir`' files (the files must be named '`dir`'). Emacs merges the files named '`dir`' from each of the listed directories. (In Emacs version 18, you can set the `Info-directory` variable to the name of only one directory.)

[1] On MS-DOS/MS-Windows systems, Info will try the '`.inf`' extension as well.

3. Specify the Info directory name in the `INFOPATH` environment variable in your '`.profile`' or '`.cshrc`' initialization file. (Only you and others who set this environment variable will be able to find Info files whose location is specified this way.)

For example, to reach a test file in the '`/home/bob/info`' directory, you could add an entry like this to the menu in the standard '`dir`' file:

```
* Test: (/home/bob/info/info-test).  Bob's own test file.
```

In this case, the absolute file name of the '`info-test`' file is written as the second part of the menu entry.

Alternatively, you could write the following in your '`.emacs`' file:

```
(require 'info)
(setq Info-directory-list
 (cons (expand-file-name "/home/bob/info")
       Info-directory-list))
```

This tells Emacs to merge the system '`dir`' file with the '`dir`' file in '`/home/bob/info`'. Thus, Info will list the '`/home/bob/info/info-test`' file as a menu entry in the '`/home/bob/info/dir`' file. Emacs does the merging only when *`M-x info`* is first run, so if you want to set `Info-directory-list` in an Emacs session where you've already run `info`, you must `(setq Info-dir-contents nil)` to force Emacs to recompose the '`dir`' file.

Finally, you can tell Info where to look by setting the `INFOPATH` environment variable in your shell startup file, such as '`.cshrc`', '`.profile`' or '`autoexec.bat`'. If you use a Bourne-compatible shell such as `sh` or `bash` for your shell command interpreter, you set the `INFOPATH` environment variable in the '`.profile`' initialization file; but if you use `csh` or `tcsh`, you set the variable in the '`.cshrc`' initialization file. On MS-DOS/MS-Windows systems, you must set `INFOPATH` in your '`autoexec.bat`' file or in the Registry. Each type of shell uses a different syntax.

- In a '`.cshrc`' file, you could set the `INFOPATH` variable as follows:

  ```
  setenv INFOPATH .:~/info:/usr/local/emacs/info
  ```

- In a '`.profile`' file, you would achieve the same effect by writing:

  ```
  INFOPATH=.:$HOME/info:/usr/local/emacs/info
  export INFOPATH
  ```

- In a '`autoexec.bat`' file, you write this command[2]:

  ```
  set INFOPATH=.;%HOME%/info;c:/usr/local/emacs/info
  ```

The '`.`' indicates the current directory as usual. Emacs uses the `INFOPATH` environment variable to initialize the value of Emacs's own `Info-directory-list` variable. The stand-alone Info reader merges any files named '`dir`' in any directory listed in the `INFOPATH` variable into a single menu presented to you in the node called '`(dir)Top`'.

However you set `INFOPATH`, if its last character is a colon[3], this is replaced by the default (compiled-in) path. This gives you a way to augment the default path with new directories without having to list all the standard places. For example (using `sh` syntax):

---

[2] Note the use of '`;`' as the directory separator, and a different syntax for using values of other environment variables.

[3] On MS-DOS/MS-Windows systems, use semi-colon instead.

```
INFOPATH=/local/info:
export INFOPATH
```

will search '`/local/info`' first, then the standard directories. Leading or doubled colons are not treated specially.

When you create your own '`dir`' file for use with `Info-directory-list` or `INFOPATH`, it's easiest to start by copying an existing '`dir`' file and replace all the text after the '`* Menu:`' with your desired entries. That way, the punctuation and special CTRL-_ characters that Info needs will be present.

21.2.4 Installing Info Directory Files

When you install an Info file onto your system, you can use the program `install-info` to update the Info directory file '`dir`'. Normally the makefile for the package runs `install-info`, just after copying the Info file into its proper installed location.

In order for the Info file to work with `install-info`, you include the commands `@dircategory` and `@direntry`...`@end direntry` in the Texinfo source file. Use `@direntry` to specify the menu entries to add to the Info directory file, and use `@dircategory` to specify which part of the Info directory to put it in. Here is how these commands are used in this manual:

```
@dircategory Texinfo documentation system
@direntry
* Texinfo: (texinfo).           The GNU documentation format.
* install-info: (texinfo)Invoking install-info. ...
...
@end direntry
```

Here's what this produces in the Info file:

```
INFO-DIR-SECTION Texinfo documentation system
START-INFO-DIR-ENTRY
* Texinfo: (texinfo).           The GNU documentation format.
* install-info: (texinfo)Invoking install-info. ...
...
END-INFO-DIR-ENTRY
```

The `install-info` program sees these lines in the Info file, and that is how it knows what to do.

Always use the `@direntry` and `@dircategory` commands near the beginning of the Texinfo input, before the first `@node` command. If you use them later on in the input, `install-info` will not notice them.

If you use `@dircategory` more than once in the Texinfo source, each usage specifies the 'current' category; any subsequent `@direntry` commands will add to that category.

When choosing a category name for the `@dircategory` command, we recommend consulting the Free Software Directory (`http://www.gnu.org/directory`). If your program is not listed there, or listed incorrectly or incompletely, please report the situation to the directory maintainers (`bug-directory@gnu.org`) so that the category names can be kept in sync.

Here are a few examples (see the '`util/dir-example`' file in the Texinfo distribution for large sample `dir` file):

```
Emacs
```

Localization
Printing
Software development
Software libraries
Text creation and manipulation

Each 'Invoking' node for every program installed should have a corresponding `@direntry`. This lets users easily find the documentation for the different programs they can run, as with the traditional `man` system.

21.2.5 Invoking `install-info`

`install-info` inserts menu entries from an Info file into the top-level '`dir`' file in the Info system (see the previous sections for an explanation of how the '`dir`' file works). It's most often run as part of software installation, or when constructing a '`dir`' file for all manuals on a system. Synopsis:

```
install-info [option]... [info-file [dir-file]]
```

If *info-file* or *dir-file* are not specified, the options (described below) that define them must be. There are no compile-time defaults, and standard input is never used. `install-info` can read only one Info file and write only one '`dir`' file per invocation.

If *dir-file* (however specified) does not exist, `install-info` creates it if possible (with no entries).

If any input file is compressed with `gzip` (see section "Invoking gzip" in *Gzip*), `install-info` automatically uncompresses it for reading. And if *dir-file* is compressed, `install-info` also automatically leaves it compressed after writing any changes. If *dir-file* itself does not exist, `install-info` tries to open '*dir-file*`.gz`'.

Options:

`--delete` Delete the entries in *info-file* from *dir-file*. The file name in the entry in *dir-file* must be *info-file* (except for an optional '`.info`' in either one). Don't insert any new entries.

`--dir-file=`*name*
`-d` *name* Specify file name of the Info directory file. This is equivalent to using the *dir-file* argument.

`--entry=`*text*
`-e` *text* Insert *text* as an Info directory entry; *text* should have the form of an Info menu item line plus zero or more extra lines starting with whitespace. If you specify more than one entry, they are all added. If you don't specify any entries, they are determined from information in the Info file itself.

`--help`
`-h` Display a usage message listing basic usage and all available options, then exit successfully.

`--info-file=`*file*
`-i` *file* Specify Info file to install in the directory. Equivalent to using the *info-file* argument.

`--info-dir=`*dir*
`-D` *dir* Specify the directory where the directory file '`dir`' resides. Equivalent to '`--dir-file=`*dir*`/dir`'.

`--item=`*text*
Same as '`--entry=`*text*'. An Info directory entry is actually a menu item.

`--quiet` Suppress warnings.

`--remove`
`-r` Same as '`--delete`'.

`--section=`*sec*
`-s` *sec* Put this file's entries in section *sec* of the directory. If you specify more than one section, all the entries are added in each of the sections. If you don't specify any sections, they are determined from information in the Info file itself.

`--version`
`-V` Display version information and exit successfully.

22 Generating HTML

`makeinfo` generates Info output by default, but given the '`--html`' option, it will generate HTML, for web browsers and other programs. This chapter gives some details on such HTML output.

`makeinfo` can also write in XML and Docbook format, but we do not as yet describe these further. See Section 1.3 [Output Formats], page 4, for a brief overview of all the output formats.

22.1 HTML Translation

`makeinfo` will include segments of Texinfo source between `@ifhtml` and `@end ifhtml` in the HTML output (but not any of the other conditionals, by default). Source between `@html` and `@end html` is passed without change to the output (i.e., suppressing the normal escaping of input '`<`', '`>`' and '`&`' characters which have special significance in HTML). See Section 17.1 [Conditional Commands], page 140.

The '`--footnote-style`' option is currently ignored for HTML output; footnotes are always linked to the end of the output file.

By default, a navigation bar is inserted at the start of each node, analogous to Info output. The '`--no-headers`' option suppresses this if used with '`--no-split`'. Header `<link>` elements in split output can support info-like navigation with browsers like Lynx and Emacs W3 which implement this HTML 1.0 feature.

The HTML generated is mostly standard (i.e., HTML 2.0, RFC-1866). One exception is that HTML 3.2 tables are generated from the `@multitable` command, but tagged to degrade as well as possible in browsers without table support. The HTML 4 '`lang`' attribute on the '`<html>`' attribute is also used. (Please report output from an error-free run of `makeinfo` which has browser portability problems as a bug.)

22.2 HTML Splitting

When splitting output (which is the default), `makeinfo` writes HTML output into (generally) one output file per Texinfo source `@node`.

The output file name is the node name with special characters replaced by '`-`'s, so it can work as a filename. In the unusual case of two different nodes having the same name after this treatment, they are written consecutively to the same file, with HTML anchors so each can be referred to separately. If `makeinfo` is run on a system which does not distinguish case in filenames, nodes which are the same except for case will also be folded into the same output file.

When splitting, the HTML output files are written into a subdirectory, with the name chosen as follows:

1. `makeinfo` first tries the subdirectory with the base name from `@setfilename` (that is, any extension is removed). For example, HTML output for `@setfilename gcc.info` would be written into a subdirectory named '`gcc`'.
2. If that directory cannot be created for any reason, then `makeinfo` tries appending '`.html`' to the directory name. For example, output for `@setfilename texinfo` would be written to '`texinfo.html`'.

3. If the '*name*`.html`' directory can't be created either, `makeinfo` gives up.

In any case, the top-level output file within the directory is always named '`index.html`'.

Monolithic output (`--no-split`) is named according to `@setfilename` (with any '`.info`' extension is replaced with '`.html`') or `--output` (the argument is used literally).

22.3 HTML CSS

Cascading Style Sheets (CSS for short) is an Internet standard for influencing the display of HTML documents: see `http://www.w3.org/Style/CSS/`.

By default, `makeinfo` includes a few simple CSS commands to better implement the appearance of some of the environments. Here are two of them, as an example:

```
pre.display { font-family:inherit }
pre.smalldisplay { font-family:inherit; font-size:smaller }
```

A full explanation of CSS is (far) beyond this manual; please see the reference above. In brief, however, this specification tells the web browser to use a 'smaller' font size for `@smalldisplay` text, and to use the 'inherited' font (generally a regular roman typeface) for both `@smalldisplay` and `@display`. By default, the HTML '`<pre>`' command uses a monospaced font.

You can influence the CSS in the HTML output with the '`--css-include=`*file*' option to `makeinfo`. This includes the contents *file* in the HTML output, as you might expect. However, the details are somewhat tricky, as described in the following, to provide maximum flexibility.

The CSS file may begin with so-called '`@import`' directives, which link to external CSS specifications for browsers to use when interpreting the document. Again, a full description is beyond our scope here, but we'll describe how they work syntactically, so we can explain how `makeinfo` handles them.

There can be more than one '`@import`', but they have to come first in the file, with only whitespace and comments interspersed, no normal definitions. (Technical exception: an '`@charset`' directive may precede the '`@import`''s. This does not alter `makeinfo`'s behavior, it just copies the '`@charset`' if present.) Comments in CSS files are delimited by '`/* ... */`', as in C. An '`@import`' directive must be in one of these two forms:

```
@import url(http://example.org/foo.css);
@import "http://example.net/bar.css";
```

As far as `makeinfo` is concerned, the crucial characters are the '`@`' at the beginning and the semicolon terminating the directive. When reading the CSS file, it simply copies any such '`@`'-directive into the output, as follows:

- If *file* contains only normal CSS declarations, it is included after `makeinfo`'s default CSS, thus overriding it.
- If *file* begins with '`@import`' specifications (see below), then the '`import`''s are included first (they have to come first, according to the standard), and then `makeinfo`'s default CSS is included. If you need to override `makeinfo`'s defaults from an '`@import`', you can do so with the '`! important`' CSS construct, as in:

```
pre.smallexample { font-size: inherit ! important }
```

- If *file* contains both '`@import`' and inline CSS specifications, the '`@import`''s are included first, then `makeinfo`'s defaults, and lastly the inline CSS from *file*.
- Any `@`-directive other than '`@import`' and '`@charset`' is treated as a CSS declaration, meaning `makeinfo` includes its default CSS and then the rest of the file.

If the CSS file is malformed or erroneous, `makeinfo`'s output is unspecified. `makeinfo` does not try to interpret the meaning of the CSS file in any way; it just looks for the special '`@`' and '`;`' characters and blindly copies the text into the output. Comments in the CSS file may or may not be included in the output.

22.4 HTML Cross-references

Cross-references between Texinfo manuals in HTML format amount, in the end, to a standard HTML `<a>` link, but the details are unfortunately complex. This section describes the algorithm used in detail, so that Texinfo can cooperate with other programs, such as `texi2html`, by writing mutually compatible HTML files.

This algorithm may or may not be used for links *within* HTML output for a Texinfo file. Since no issues of compatibility arise in such cases, we do not need to specify this.

We try to support references to such "external" manuals in both monolithic and split forms. A *monolithic* (mono) manual is entirely contained in one file, and a *split* manual has a file for each node. (See Section 22.2 [HTML Splitting], page 182.)

Acknowledgement: this algorithm was primarily devised by Patrice Dumas in 2003–04.

22.4.1 HTML Cross-reference Link Basics

For our purposes, an HTML link consists of four components: a host name, a directory part, a file part, and a target part. We always assume the `http` protocol. For example:

`http://`*`host/dir/file`*`.html#`*`target`*

The information to construct a link comes from the node name and manual name in the cross-reference command in the Texinfo source (see Chapter 8 [Cross References], page 64), and from *external information*, which is currently simply hardwired. In the future, it may come from an external data file.

We now consider each part in turn.

The *host* is hardwired to be the local host. This could either be the literal string '`localhost`', or, according to the rules for HTML links, the '`http://localhost/`' could be omitted entirely.

The *dir* and *file* parts are more complicated, and depend on the relative split/mono nature of both the manual being processed and the manual that the cross-reference refers to. The underlying idea is that there is one directory for Texinfo manuals in HTML, and each manual is either available as a monolithic file '`manual.html`', or a split subdirectory '`manual/*.html`'. Here are the cases:

- If the present manual is split, and the referent manual is also split, the directory is '`../`*`referent`*`/`' and the file is the expanded node name (described later).
- If the present manual is split, and the referent manual is mono, the directory is '`../`' and the file is '*`referent`*`.html`'.

- If the present manual is mono, and the referent manual is split, the directory is '*referent*`/`' and the file is the expanded node name.
- If the present manual is mono, and the referent manual is also mono, the directory is '`./`' (or just the empty string), and the file is '*referent*`.html`'.

One exception: the algorithm for node name expansion prefixes the string '`g_t`' when the node name begins with a non-letter. This kludge (due to XHTML rules) is not necessary for filenames, and is therefore omitted.

Any directory part in the filename argument of the source cross-reference command is ignored. Thus, `@xref{,,,../foo}` and `@xref{,,,foo}` both use '`foo`' as the manual name. This is because any such attempted hardwiring of the directory is very unlikely to be useful for both Info and HTML output.

Finally, the *target* part is always the expanded node name.

Whether the present manual is split or mono is determined by user option; `makeinfo` defaults to split, with the '`--no-split`' option overriding this.

Whether the referent manual is split or mono is another bit of the external information. For now, `makeinfo` simply assumes the referent manual is the same as the present manual.

There can be a mismatch between the format of the referent manual that the generating software assumes, and the format it's actually present in. See Section 22.4.5 [HTML Xref Mismatch], page 188.

22.4.2 HTML Cross-reference Node Name Expansion

As mentioned in the previous section, the key part of the HTML cross-reference algorithm is the conversion of node names in the Texinfo source into strings suitable for XHTML identifiers and filenames. The restrictions are similar for each: plain ASCII letters, numbers, and the '`-`' and '`_`' characters are all that can be used. (Although HTML anchors can contain most characters, XHTML is more restrictive.)

Cross-references in Texinfo can actually refer either to nodes or anchors (see Section 6.5 [anchor], page 59), but anchors are treated identically to nodes in this context, so we'll continue to say "node" names for simplicity.

(`@`-commands and 8-bit characters are not presently handled by `makeinfo` for HTML cross-references. See the next section.)

A special exception: the Top node (see Section 3.6 [The Top Node], page 39) is always mapped to the file '`index.html`', to match web server software. However, the HTML *target* is '`Top`'. Thus (in the split case):

```
@xref{Top, Introduction,, emacs, The GNU Emacs Manual}.
⇒ <a href="emacs/index.html#Top">
```

1. The standard ASCII letters (a-z and A-Z) are not modified. All other characters are changed as specified below.
2. The standard ASCII numbers (0-9) are not modified except when a number is the first character of the node name. In that case, see below.
3. Multiple consecutive space, tab and newline characters are transformed into just one space. (It's not possible to have newlines in node names with the current implementation, but we specify it anyway, just in case.)

4. Leading and trailing spaces are removed.
5. After the above has been applied, each remaining space character is converted into a '`-`' character.
6. Other ASCII 7-bit characters are transformed into '`_00xx`', where *xx* is the ASCII character code in (lowercase) hexadecimal. This includes '`_`', which is mapped to '`_005f`'.
7. If the node name does not begin with a letter, the literal string '`g_t`' is prefixed to the result. (Due to the rules above, that string can never occur otherwise; it is an arbitrary choice, standing for "GNU Texinfo".) This is necessary because XHTML requires that identifiers begin with a letter.

   For example:

   ```
   @node A  node --- with _'%
   ⇒ A-node-_002d_002d_002d-with-_005f_0027_0025
   ```

   Notice in particular:

- '`_`' ⇒ '`_005f`'
- '`-`' ⇒ '`_002d`'
- '`A node`' ⇒ '`A-node`'

On case-folding computer systems, nodes differing only by case will be mapped to the same file.

In particular, as mentioned above, Top always maps to the file '`index.html`'. Thus, on a case-folding system, Top and a node named 'Index' will both be written to '`index.html`'.

Fortunately, the targets serve to distinguish these cases, since HTML target names are always case-sensitive, independent of operating system.

22.4.3 HTML Cross-reference Command Expansion

In standard Texinfo, node names may not contain `@`-commands. `makeinfo` has an option '`--commands-in-node-names`' which partially supports it (see Section 21.1.2 [Invoking makeinfo], page 168), but it is not robust and not recommended.

Thus, `makeinfo` also does not fully implement this part of the HTML cross-reference algorithm, but it is documented here for the sake of completeness.

First, comments are removed.

Next, any `@value` commands (see Section 17.4.1 [set value], page 143) and macro invocations (see Section 19.2 [Invoking Macros], page 151) are fully expanded.

Then, for the following commands, the command name and braces are removed, the text of the argument is recursively transformed:

```
@asis @b @cite @code @command @dfn @dmn @dotless
@emph @env @file @indicateurl @kbd @key
@samp @sc @slanted @strong @t @var @w
```

For `@sc`, any letters are capitalized.

The following commands are replaced by constant text, as shown. If any of these commands have non-empty arguments, as in `@TeX{bad}`, it is an error, and the result is unspecified. '(space)' means a space character, '(nothing)' means the empty string, etc.

The notation 'U+*xxxx*' means Unicode code point *xxxx*. There are further transformations of many of these expansions for the final file or target name, such as space characters to '`-`', etc., according to the other rules.

`@(newline)`	(space)
`@(space)`	(space)
`@(tab)`	(space)
`@!`	'`!`'
`@*`	(space)
`@-`	(nothing)
`@.`	'`.`'
`@:`	(nothing)
`@?`	'`?`'
`@@`	'`@`'
`@{`	'`{`'
`@}`	'`}`'
`@LaTeX`	'`LaTeX`'
`@TeX`	'`TeX`'
`@bullet`	U+2022
`@comma`	'`,`'
`@copyright`	U+00A9
`@dots`	U+2026
`@enddots`	'`...`'
`@equiv`	U+2261
`@error`	'`error-->`'
`@euro`	U+20AC
`@exclamdown`	U+00A1
`@expansion`	U+2192
`@minus`	U+2212
`@ordf`	U+00AA
`@ordm`	U+00BA
`@point`	U+2217
`@pounds`	U+00A3
`@print`	'`-|`'
`@questiondown`	U+00BF
`@registeredsymbol`	U+00AE
`@result`	U+21D2
`@tie`	(space)

An `@acronym` or `@abbr` command is replaced by the first argument, followed by the second argument in parentheses, if present. See Section 9.1.15 [acronym], page 81.

An `@email` command is replaced by the *text* argument if present, else the address. See Section 9.1.17 [email], page 82.

An `@image` command is replaced by the filename (first) argument. See Section 12.2 [Images], page 104.

A `@verb` command is replaced by its transformed argument. See Section 9.1.6 [verb], page 78.

Any other command is an error, and the result is unspecified.

22.4.4 HTML Cross-reference 8-bit Character Expansion

Usually, characters other than plain 7-bit ASCII are transformed into the corresponding Unicode code point(s) in Normalization Form C, which uses precomposed characters where available. (This is the normalization form recommended by the W3C and other bodies.) This holds when that code point is 0xffff or less, as it almost always is.

These will then be further transformed by the rules above into the string '*_xxxx*', where *xxxx* is the code point in hex.

For example, combining this rule and the previous section:

```
@node @b{A} @TeX{} @u{B} @point{}@enddots{}
⇒ A-TeX-B_0306-_2605_002e_002e_002e
```

Notice: 1) `@enddots` expands to three periods which in turn expands to three '`_002e`''s; 2) `@u{B}` is a 'B' with a breve accent, which does not exist as a pre-accented Unicode character, therefore expands to '`B_0306`' (B with combining breve).

When the Unicode code point is above 0xffff, the transformation is '*__xxxxxx*', that is, two leading underscores followed by six hex digits. Since Unicode has declared that their highest code point is 0x10ffff, this is sufficient. (We felt it was better to define this extra escape than to always use six hex digits, since the first two would nearly always be zeros.)

For the definition of Unicode Normalization Form C, see Unicode report UAX#15, `http://www.unicode.org/reports/tr15/`. Many related documents and implementations are available elsewhere on the web.

22.4.5 HTML Cross-reference Mismatch

As mentioned earlier (see Section 22.4.1 [HTML Xref Link Basics], page 184), the generating software has to guess whether a given manual being cross-referenced is available in split or monolithic form—and, inevitably, it might guess wrong. However, it is possible when the referent manual itself is generated, it is possible to handle at least some mismatches.

In the case where we assume the referent is split, but it is actually available in mono, the only recourse would be to generate a '`manual/`' subdirectory full of HTML files which redirect back to the monolithic '`manual.html`'. Since this is essentially the same as a split manual in the first place, it's not very appealing.

On the other hand, in the case where we assume the referent is mono, but it is actually available in split, it is possible to use JavaScript to redirect from the putatively monolithic '`manual.html`' to the different '`manual/node.html`' files. Here's an example:

```
function redirect() {
  switch (location.hash) {
    case "#Node1":
      location.replace("manual/Node1.html#Node1"); break;
    case "#Node2" :
      location.replace("manual/Node2.html#Node2"); break;
    ...
    default:;
  }
}
```

Then, in the `<body>` tag of '`manual.html`':

```
<body onLoad="redirect();">
```

Once again, this is something the software which generated the *referent* manual has to do in advance, it's not something the software generating the actual cross-reference in the present manual can control.

Ultimately, we hope to allow for an external configuration file to control which manuals are available from where, and how.

Appendix A @-Command List

Here is an alphabetical list of the @-commands in Texinfo. Square brackets, [ ], indicate optional arguments; an ellipsis, '...', indicates repeated text.

More specifics on the general syntax of different @-commands are given in the section below.

`@`*whitespace*
: An @ followed by a space, tab, or newline produces a normal, stretchable, interword space. See Section 14.2.3 [Multiple Spaces], page 115.

`@!`
: Generate an exclamation point that really does end a sentence (usually after an end-of-sentence capital letter). See Section 14.2.2 [Ending a Sentence], page 114.

`@"`
`@'`
: Generate an umlaut or acute accent, respectively, over the next character, as in ö and ó. See Section 14.3 [Inserting Accents], page 116.

`@*`
: Force a line break. See Section 15.2 [Line Breaks], page 123.

`@,{`*c*`}`
: Generate a cedilla accent under *c*, as in ç. See Section 14.3 [Inserting Accents], page 116.

`@-`
: Insert a discretionary hyphenation point. See Section 15.3 [- and hyphenation], page 124.

`@.`
: Produce a period that really does end a sentence (usually after an end-of-sentence capital letter). See Section 14.2.2 [Ending a Sentence], page 114.

`@/`
: Produces no output, but allows a line break. See Section 15.2 [Line Breaks], page 123.

`@:`
: 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 14.2.1 [Not Ending a Sentence], page 114.

`@=`
: Generate a macron (bar) accent over the next character, as in ō. See Section 14.3 [Inserting Accents], page 116.

`@?`
: Generate a question mark that really does end a sentence (usually after an end-of-sentence capital letter). See Section 14.2.2 [Ending a Sentence], page 114.

`@@`
: Stands for an at sign, '@'. See Section 14.1 [Inserting @ and {} and ,], page 113.

`@\`
: Stands for a backslash ('\') inside `@math`. See Section 14.9 [`math`], page 118.

`@^`
`@``
: Generate a circumflex (hat) or grave accent, respectively, over the next character, as in ô and è. See Section 14.3 [Inserting Accents], page 116.

`@{`
: Stands for a left brace, '{'. See Section 14.1 [Inserting @ and {} and ,], page 113.

`@}`
: Stands for a right-hand brace, '}'.
See Section 14.1 [Inserting @ and {} and ,], page 113.

`@~` Generate a tilde accent over the next character, as in Ñ. See Section 14.3 [Inserting Accents], page 116.

`@AA{}`
`@aa{}` Generate the uppercase and lowercase Scandinavian A-ring letters, respectively: Å, å. See Section 14.3 [Inserting Accents], page 116.

`@abbr{`*abbreviation*`}`
Tag *abbreviation* as an abbreviation, such as 'Comput.'. See Section 9.1.14 [`abbr`], page 81.

`@acronym{`*acronym*`}`
Tag *acronym* as an acronym, such as 'NASA'. See Section 9.1.15 [`acronym`], page 81.

`@AE{}`
`@ae{}` Generate the uppercase and lowercase AE ligatures, respectively: Æ, æ. See Section 14.3 [Inserting Accents], page 116.

`@afivepaper`
Change page dimensions for the A5 paper size. See Section 20.12 [A4 Paper], page 164.

`@afourlatex`
`@afourpaper`
`@afourwide`
Change page dimensions for the A4 paper size. See Section 20.12 [A4 Paper], page 164.

`@alias` *new=existing*
Make the command '`@`*new*' an alias for the existing command '`@`*existing*'. See Section 19.4 [alias], page 153.

`@anchor{`*name*`}`
Define *name* as the current location for use as a cross-reference target. See Section 6.5 [`@anchor`], page 59.

`@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 49.

`@appendixsec` *title*
`@appendixsection` *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. `@appendixsection` is a longer spelling of the `@appendixsec` command. See Section 5.8 [Section Commands], page 50.

`@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 50.

`@appendixsubsubsec` *title*
: Begin an appendix subsubsection within an appendix subsection. 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 51.

`@asis` Used following `@table`, `@ftable`, and `@vtable` to print the table's first column without highlighting ("as is"). See Section 11.4 [Making a Two-column Table], page 97.

`@author` *author*
: Typeset *author* flushleft and underline it. See Section 3.4.3 [The `@title` and `@author` Commands], page 35.

`@b{`*text*`}` Print *text* in **bold** font. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@bullet{}`
: Generate a large round dot, or the closest possible thing to one. See Section 14.4.2 [`@bullet`], page 117.

`@bye` Stop formatting a file. The formatters do not see the contents of a file following an `@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 1.8 [Comments], page 10.

`@caption` Define the full caption for a `@float`. See Section 12.1.2 [caption shortcaption], page 103.

`@cartouche`
: Highlight an example or quotation by drawing a box with rounded corners around it. Pair with `@end cartouche`. No effect in Info. See Section 10.14 [Drawing Cartouches Around Examples], page 92.)

`@center` *line-of-text*
: Center the line of text following the command. See Section 3.4.2 [`@center`], page 34.

`@centerchap` *line-of-text*
: Like `@chapter`, but centers the chapter title. See Section 5.4 [`@chapter`], page 48.

`@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 49.

`@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 48.

`@cindex` *entry*
: Add *entry* to the index of concepts. See Section 13.1 [Defining the Entries of an Index], page 108.

`@cite{`*reference*`}`
: Highlight the name of a book or other reference that lacks a companion Info file. See Section 9.1.13 [`@cite`], page 81.

`@clear` *flag*
: Unset *flag*, preventing the Texinfo formatting commands from formatting text between subsequent pairs of `@ifset` *flag* and `@end ifset` commands, and preventing `@value{`*flag*`}` from expanding to the value to which *flag* is set. See Section 17.4 [`@set @clear @value`], page 142.

`@code{`*sample-code*`}`
: Highlight text that is an expression, a syntactically complete token of a program, or a program name. See Section 9.1.2 [`@code`], page 75.

`@comma{}`
: Insert a comma ',' character; only needed when a literal comma would be taken as an argument separator. See Section 14.1.3 [Inserting a Comma], page 113.

`@command{`*command-name*`}`
: Indicate a command name, such as `ls`. See Section 9.1.10 [`@command`], page 80.

`@comment` *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 `@c`. See Section 1.8 [Comments], page 10.

`@contents`
: Print a complete table of contents. Has no effect in Info, which uses menus instead. See Section 3.5 [Generating a Table of Contents], page 38.

`@copyright{}`
: Generate the copyright symbol ©. See Section 14.5.2 [`@copyright{}`], page 117.

`@defcodeindex` *index-name*
: Define a new index and its indexing command. Print entries in an `@code` font. See Section 13.5 [Defining New Indices], page 111.

`@defcv` *category class name*
`@defcvx` *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 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deffn` *category name arguments ...*
`@deffnx` *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 16 [Definition Commands], page 127.

`@defindex` *index-name*
: Define a new index and its indexing command. Print entries in a roman font. See Section 13.5 [Defining New Indices], page 111.

`@definfoenclose` *newcmd, before, after,*
: Create new `@`-command *newcmd* for Info that marks text by enclosing it in strings that precede and follow the text. See Section 19.5 [definfoenclose], page 154.

`@defivar` *class instance-variable-name*
`@defivarx` *class instance-variable-name*
: This command formats a description for an instance variable in object-oriented programming. The command is equivalent to '`@defcv {Instance Variable} ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defmac` *macroname arguments ...*
`@defmacx` *macroname arguments ...*
: Format a description for a macro. The command is equivalent to '`@deffn Macro ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defmethod` *class method-name arguments ...*
`@defmethodx` *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 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defop` *category class name arguments ...*
`@defopx` *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 16 [Definition Commands], page 127, and Section 16.5.6 [Abstract Objects], page 135.

`@defopt` *option-name*
`@defoptx` *option-name*
: Format a description for a user option. The command is equivalent to '`@defvr {User Option} ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defspec` *special-form-name arguments ...*
`@defspecx` *special-form-name arguments ...*
: Format a description for a special form. The command is equivalent to '`@deffn {Special Form} ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deftp` *category name-of-type attributes...*
`@deftpx` *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 16 [Definition Commands], page 127, and Section 16.5.5 [Data Types], page 134.

`@deftypecv` *category class data-type name*
`@deftypecvx` *category class data-type name*

Format a description for a typed class variable in object-oriented programming. See Chapter 16 [Definition Commands], page 127, and Section 16.5.6 [Abstract Objects], page 135.

`@deftypefn` *classification data-type name arguments...*
`@deftypefnx` *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 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deftypefun` *data-type function-name arguments...*
`@deftypefunx` *data-type function-name arguments...*

Format a description for a function in a typed language. The command is equivalent to '`@deftypefn Function ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deftypeivar` *class data-type variable-name*
`@deftypeivarx` *class data-type variable-name*

Format a description for a typed instance variable in object-oriented programming. See Chapter 16 [Definition Commands], page 127, and Section 16.5.6 [Abstract Objects], page 135.

`@deftypemethod` *class data-type method-name arguments...*
`@deftypemethodx` *class data-type method-name arguments...*

Format a description for a typed method in object-oriented programming. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deftypeop` *category class data-type name arguments...*
`@deftypeopx` *category class data-type name arguments...*

Format a description for a typed operation in object-oriented programming. See Chapter 16 [Definition Commands], page 127, and Section 16.5.6 [Abstract Objects], page 135.

`@deftypevar` *data-type variable-name*
`@deftypevarx` *data-type variable-name*

Format a description for a variable in a typed language. The command is equivalent to '`@deftypevr Variable ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@deftypevr` *classification data-type name*
`@deftypevrx` *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 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defun` *function-name arguments...*
`@defunx` *function-name arguments...*
: Format a description for functions. The command is equivalent to '`@deffn Function ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defvar` *variable-name*
`@defvarx` *variable-name*
: Format a description for variables. The command is equivalent to '`@defvr Variable ...`'. See Chapter 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@defvr` *category name*
`@defvrx` *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 16 [Definition Commands], page 127, and Section 16.4 [Def Cmds in Detail], page 129.

`@detailmenu`
: Avoid `makeinfo` confusion stemming from the detailed node listing in a master menu. See Section 3.6.2 [Master Menu Parts], page 40.

`@dfn{`*term*`}`
: Highlight the introductory or defining use of a term. See Section 9.1.12 [`@dfn`], page 80.

`@dircategory` *dirpart*
: Specify a part of the Info directory menu where this file's entry should go. See Section 21.2.4 [Installing Dir Entries], page 179.

`@direntry`
: Begin the Info directory menu entry for this file. Pair with `@end direntry`. See Section 21.2.4 [Installing Dir Entries], page 179.

`@display` Begin a kind of example. Like `@example` (indent text, do not fill), but do not select a new font. Pair with `@end display`. See Section 10.8 [`@display`], page 89.

`@dmn{`*dimension*`}`
: Format a unit of measure, as in 12 pt. Causes TEX to insert a thin space before *dimension*. No effect in Info. See Section 14.2.4 [`@dmn`], page 115.

`@docbook` Enter Docbook completely. Pair with `@end docbook`. See Section 17.3 [Raw Formatter Commands], page 141.

`@documentdescription`
: Set the document description text, included in the HTML output. Pair with `@end documentdescription`. See Section 3.7.1 [`@documentdescription`], page 41.

`@documentencoding` *enc*
: Declare the input encoding to be *enc*. See Section 18.2 [`@documentencoding`], page 148.

`@documentlanguage` *CC*
: Declare the document language as the two-character ISO-639 abbreviation *CC*. See Section 18.1 [`@documentlanguage`], page 147.

`@dotaccent{`*c*`}`
: Generate a dot accent over the character *c*, as in ȯ. See Section 14.3 [Inserting Accents], page 116.

`@dots{}` Insert an ellipsis: '`...`'. See Section 14.4.1 [`@dots`], page 117.

`@email{`*address*`[, `*displayed-text*`]}`
: Indicate an electronic mail address. See Section 9.1.17 [`@email`], page 82.

`@emph{`*text*`}`
: Highlight *text*; text is displayed in *italics* in printed output, and surrounded by asterisks in Info. See Section 9.2 [Emphasizing Text], page 83.

`@end` *environment*
: Ends *environment*, as in '`@end example`'. See Section 1.6 [@-commands], page 8.

`@env{`*environment-variable*`}`
: Indicate an environment variable name, such as `PATH`. See Section 9.1.8 [`@env`], page 79.

`@enddots{}`
: Generate an end-of-sentence of ellipsis, like this .... See Section 14.4.1 [`@dots{}`], page 117.

`@enumerate [`*number-or-letter*`]`
: Begin a numbered list, using `@item` for each entry. Optionally, start list with *number-or-letter*. Pair with `@end enumerate`. See Section 11.3 [`@enumerate`], page 96.

`@equiv{}` Indicate to the reader the exact equivalence of two forms with a glyph: '≡'. See Section 14.10.6 [Equivalence], page 121.

`@euro{}` Generate the Euro currency sign. See Section 14.6 [`@euro{}`], page 118.

`@error{}` Indicate to the reader with a glyph that the following text is an error message: 'error '. See Section 14.10.5 [Error Glyph], page 121.

`@evenfooting [`*left*`] @| [`*center*`] @| [`*right*`]`
`@evenheading [`*left*`] @| [`*center*`] @| [`*right*`]`
: Specify page footings resp. headings for even-numbered (left-hand) pages. See Section E.4 [How to Make Your Own Headings], page 228.

`@everyfooting [left] @| [center] @| [right]`
`@everyheading [left] @| [center] @| [right]`
: Specify page footings resp. headings for every page. Not relevant to Info. See Section E.4 [How to Make Your Own Headings], page 228.

`@example` Begin an example. Indent text, do not fill, and select fixed-width font. Pair with `@end example`. See Section 10.3 [`@example`], page 86.

`@exampleindent` *indent*
: Indent example-like environments by *indent* number of spaces (perhaps 0). See Section 3.7.5 [Paragraph Indenting], page 43.

`@exclamdown{}`
: Produce an upside-down exclamation point. See Section 14.3 [Inserting Accents], page 116.

`@exdent` *line-of-text*
: Remove any indentation a line might have. See Section 10.10 [Undoing the Indentation of a Line], page 90.

`@expansion{}`
: Indicate the result of a macro expansion to the reader with a special glyph: '↦'. See Section 14.10.3 [↦ Indicating an Expansion], page 120.

`@file{`*filename*`}`
: Highlight the name of a file, buffer, node, or directory. See Section 9.1.9 [`@file`], page 80.

`@finalout`
: Prevent TEX from printing large black warning rectangles beside over-wide lines. See Section 20.10 [Overfull hboxes], page 163.

`@findex` *entry*
: Add *entry* to the index of functions. See Section 13.1 [Defining the Entries of an Index], page 108.

`@float` Environment to define floating material. Pair with `@end float`. See Section 12.1 [Floats], page 102.

`@flushleft`
`@flushright`
: Left justify every line but leave the right end ragged. Leave font as is. Pair with `@end flushleft`. `@flushright` analogous. See Section 10.11 [`@flushleft` and `@flushright`], page 90.

`@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 'Separate' node style. See Section 12.3 [Footnotes], page 106.

`@footnotestyle` *style*
: Specify an Info file's footnote style, either '`end`' for the end node style or '`separate`' for the separate node style. See Section 12.3 [Footnotes], page 106.

`@format` Begin a kind of example. Like `@display`, but do not narrow the margins. Pair with `@end format`. See Section 10.3 [`@example`], page 86.

`@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 11.4.2 [`@ftable` and `@vtable`], page 99.

`@group` Hold text together that must appear on one printed page. Pair with `@end group`. Not relevant to Info. See Section 15.8 [`@group`], page 125.

`@H{`*c*`}` Generate the long Hungarian umlaut accent over *c*, as in ő.

`@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 50.

`@headings` *on-off-single-double*
: Turn page headings on or off, and/or specify single-sided or double-sided page headings for printing. See Section 3.4.6 [The `@headings` Command], page 37.

`@html` Enter HTML completely. Pair with `@end html`. See Section 17.3 [Raw Formatter Commands], page 141.

`@hyphenation{`*hy-phen-a-ted words*`}`
: Explicitly define hyphenation points. See Section 15.3 [`@-` and `@hyphenation`], page 124.

`@i{`*text*`}` Print *text* in *italic* font. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@ifclear` *flag*
: If *flag* is cleared, the Texinfo formatting commands format text between `@ifclear` *flag* and the following `@end ifclear` command. See Section 17.4 [`@set @clear @value`], page 142.

`@ifdocbook`
`@ifhtml`
`@ifinfo` Begin text that will appear only in the given output format. `@ifinfo` output appears in both Info and (for historical compatibility) plain text output. Pair with `@end ifdocbook` resp. `@end ifhtml` resp. `@end ifinfo`. See Chapter 17 [Conditionals], page 140.

`@ifnotdocbook`
`@ifnothtml`
`@ifnotinfo`
`@ifnotplaintext`
`@ifnottex`
`@ifnotxml`
: Begin a stretch of text that will be ignored in one output format but not the others. The text appears in the formats not specified: `@ifnothtml` text is omitted from html output, etc. The exception is `@ifnotinfo` text, which is omitted

from plain text output as well as Info output. Pair with the corresponding `@end ifnot`*`format`*. See Chapter 17 [Conditionals], page 140.

`@ifplaintext`
: Begin text that will appear only in the plain text output. Pair with `@end ifplaintext`. See Chapter 17 [Conditionals], page 140.

`@ifset` *`flag`*
: If *flag* is set, the Texinfo formatting commands format text between `@ifset` *`flag`* and the following `@end ifset` command. See Section 17.4 [`@set @clear @value`], page 142.

`@iftex`
: Begin text that will not appear in the Info file or other output, but will be processed only by TEX. Pair with `@end iftex`. See Chapter 17 [Conditionally Visible Text], page 140.

`@ifxml`
: Begin text that will appear only in the XML output. Pair with `@end ifxml`. See Chapter 17 [Conditionals], page 140.

`@ignore`
: Begin text that will not appear in any output. Pair with `@end ignore`. See Section 1.8 [Comments and Ignored Text], page 10.

`@image{`*`filename`*`, [`*`width`*`], [`*`height`*`], [`*`alt`*`], [`*`ext`*`]}`
: Include graphics image in external *filename* scaled to the given *width* and/or *height*, using *alt* text and looking for '*`filename.ext`*' in HTML. See Section 12.2 [Images], page 104.

`@include` *`filename`*
: Incorporate the contents of the file *filename* into the Info file or printed document. See Appendix D [Include Files], page 222.

`@indicateurl{`*`indicateurl`*`}`
: Indicate text that is a uniform resource locator for the World Wide Web. See Section 9.1.16 [`@indicateurl`], page 82.

`@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 8.8 [Cross references using `@inforef`], page 72.

`\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 recognize `@` until after it has read the definitions file. See Section 3.2 [Texinfo File Header], page 29.

`@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`, `@ftable`, and `@vtable`. See Chapter 11 [Lists and Tables], page 94.

`@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 11.2 [`@itemize`], page 95.

`@itemx` Like `@item` but do not generate extra vertical space above the item text. See Section 11.4.3 [`@itemx`], page 99.

`@kbd{`*`keyboard-characters`*`}`
Indicate text that is characters of input to be typed by users. See Section 9.1.3 [`@kbd`], page 76.

`@kbdinputstyle` *`style`*
Specify when `@kbd` should use a font distinct from `@code`. See Section 9.1.3 [`@kbd`], page 76.

`@key{`*`key-name`*`}`
Indicate a name for a key on a keyboard. See Section 9.1.4 [`@key`], page 77.

`@kindex` *`entry`*
Add *entry* to the index of keys. See Section 13.1 [Defining the Entries of an Index], page 108.

`@L{}`
`@l{}` Generate the uppercase and lowercase Polish suppressed-L letters, respectively: Ł, ł.

`@LaTeX{}` Insert the logo LaTeX. See Section 14.5.1 [TeX and LaTeX], page 117.

`@lisp` Begin an example of Lisp code. Indent text, do not fill, and select fixed-width font. Pair with `@end lisp`. See Section 10.6 [`@lisp`], page 88.

`@listoffloats`
Produce a table-of-contents-like listing of `@float`s. See Section 12.1.3 [listoffloats], page 103.

`@lowersections`
Change subsequent chapters to sections, sections to subsections, and so on. See Section 5.12 [`@raisesections` and `@lowersections`], page 51.

`@macro` *`macroname {params}`*
Define a new Texinfo command *`@macroname{params}`*. Only supported by `makeinfo` and `texi2dvi`. See Section 19.1 [Defining Macros], page 150.

`@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. See Section 5.6 [`@majorheading` and `@chapheading`], page 49.

`@math{`*`mathematical-expression`*`}`
Format a mathematical expression. See Section 14.9 [`@math`: Inserting Mathematical Expressions], page 118.

`@menu` Mark the beginning of a menu of nodes in Info. No effect in a printed manual. Pair with `@end menu`. See Chapter 7 [Menus], page 60.

`@minus{}` Generate a minus sign, '−'. See Section 14.8 [`@minus`], page 118.

`@multitable` *`column-width-spec`*
: Begin a multi-column table. Pair with `@end multitable`. See Section 11.5.1 [Multitable Column Widths], page 100.

`@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 15.9 [`@need`], page 126.

`@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 6.3 [`@node`], page 55.

`@noindent`
: Prevent text from being indented as if it were a new paragraph. See Section 10.12 [`@noindent`], page 91.

`@novalidate`
: Suppress validation of node references, omit creation of auxiliary files with TEX. Use before `@setfilename`. See Section 21.1.4 [Pointer Validation], page 172.

`@O{}`
`@o{}`
: Generate the uppercase and lowercase O-with-slash letters, respectively: Ø, ø.

`@oddfooting [`*`left`*`] @| [`*`center`*`] @| [`*`right`*`]`
`@oddheading [`*`left`*`] @| [`*`center`*`] @| [`*`right`*`]`
: Specify page footings resp. headings for odd-numbered (right-hand) pages. See Section E.4 [How to Make Your Own Headings], page 228.

`@OE{}`
`@oe{}`
: Generate the uppercase and lowercase OE ligatures, respectively: Œ, œ. See Section 14.3 [Inserting Accents], page 116.

`@option{`*`option-name`*`}`
: Indicate a command-line option, such as '`-l`' or '`--help`'. See Section 9.1.11 [`@option`], page 80.

`@page`
: Start a new page in a printed manual. No effect in Info. See Section 15.7 [`@page`], page 125.

`@pagesizes [`*`width`*`][, `*`height`*`]`
: Change page dimensions. See Section 20.13 [pagesizes], page 165.

`@paragraphindent` *`indent`*
: Indent paragraphs by *indent* number of spaces (perhaps 0); preserve source file indentation if *indent* is `asis`. See Section 3.7.3 [Paragraph Indenting], page 42.

`@pindex` *`entry`*
: Add *entry* to the index of programs. See Section 13.1 [Defining the Entries of an Index], page 108.

`@point{}`
: Indicate the position of point in a buffer to the reader with a glyph: '⋆'. See Section 14.10.7 [Indicating Point in a Buffer], page 121.

`@pounds{}`
: Generate the pounds sterling currency sign. See Section 14.7 [`@pounds{}`], page 118.

`@print{}` Indicate printed output to the reader with a glyph: ‘ ⊣ ’. See Section 14.10.4 [Print Glyph], page 120.

`@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-or-title*`], [`*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. Only the first argument is mandatory. See Section 8.7 [`@pxref`], page 71.

`@questiondown{}`
: Generate an upside-down question mark. See Section 14.3 [Inserting Accents], page 116.

`@quotation`
: Narrow the margins to indicate text that is quoted from another real or imaginary work. Takes optional argument of prefix text. Pair with `@end quotation`. See Section 10.2 [`@quotation`], page 86.

`@r{`*text*`}` Print *text* in roman font. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@raisesections`
: Change subsequent sections to chapters, subsections to sections, and so on. See Section 5.12 [`@raisesections` and `@lowersections`], page 51.

`@ref{`*node-name*`, [`*entry*`], [`*topic-or-title*`], [`*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 8.6 [`@ref`], page 70.

`@refill` This command used to refill and indent the paragraph after all the other processing has been done. It is no longer needed, since all formatters now automatically refill as needed, but you may still see it in the source to some manuals, as it does no harm.

`@registeredsymbol{}`
: Generate the legal symbol ®. See Section 14.5.3 [`@registeredsymbol{}`], page 118.

`@result{}`
: Indicate the result of an expression to the reader with a special glyph: ‘⇒’. See Section 14.10.2 [`@result`], page 120.

`@ringaccent{`*c*`}`
: Generate a ring accent over the next character, as in ů. See Section 14.3 [Inserting Accents], page 116.

`@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 9.1.5 [@samp], page 78.

`@sansserif{`*text*`}`
: Print *text* in a sans serif font if possible. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@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 9.2.2 [Smallcaps], page 83.

`@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 49.

`@set` *flag* `[`*string*`]`
: Make *flag* active, causing the Texinfo formatting commands to format text between subsequent pairs of `@ifset` *flag* and `@end ifset` commands. Optionally, set value of *flag* to *string*. See Section 17.4 [@set @clear @value], page 142.

`@setchapternewpage` *on-off-odd*
: Specify whether chapters start on new pages, and if so, whether on odd-numbered (right-hand) new pages. See Section 3.7.2 [@setchapternewpage], page 41.

`@setcontentsaftertitlepage`
: Put the table of contents after the '`@end titlepage`' even if the `@contents` command is not there. See Section 3.5 [Contents], page 38.

`@setfilename` *info-file-name*
: Provide a name to be used by the Info file. This command is essential for TEX formatting as well, even though it produces no output. See Section 3.2.3 [@setfilename], page 30.

`@setshortcontentsaftertitlepage`
: Place the short table of contents after the '`@end titlepage`' command even if the `@shortcontents` command is not there. See Section 3.5 [Contents], page 38.

`@settitle` *title*
: Provide a title for page headers in a printed manual, and the default document description for HTML '`<head>`'. See Section 3.2.4 [@settitle], page 31.

`@shortcaption`
: Define the short caption for a `@float`. See Section 12.1.2 [caption shortcaption], page 103.

`@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 3.5 [Generating a Table of Contents], page 38.

`@shorttitlepage` *title*
: Generate a minimal title page. See Section 3.4.1 [`@titlepage`], page 34.

`@slanted{`*text*`}`
: Print *text* in a *slanted* font if possible. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@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 20.11 [Printing Small Books], page 164. Also, see Section 10.7 [small], page 89.

`@smalldisplay`
: Begin a kind of example. Like `@smallexample` (narrow margins, no filling), but do not select the fixed-width font. Pair with `@end smalldisplay`. See Section 10.7 [small], page 89.

`@smallexample`
: Indent text to indicate an example. Do not fill, select fixed-width font, narrow the margins. In printed manuals, print text in a smaller font than with `@example`. Pair with `@end smallexample`. See Section 10.7 [small], page 89.

`@smallformat`
: Begin a kind of example. Like `@smalldisplay`, but do not narrow the margins. Pair with `@end smallformat`. See Section 10.7 [small], page 89.

`@smalllisp`
: Begin an example of Lisp code. Same as `@smallexample`. Pair with `@end smalllisp`. See Section 10.7 [small], page 89.

`@sp` *n*
: Skip *n* blank lines. See Section 15.6 [`@sp`], page 125.

`@ss{}`
: Generate the German sharp-S es-zet letter, ß. See Section 14.3 [Inserting Accents], page 116.

`@strong {`*text*`}`
: Emphasize *text* by typesetting it in a **bold** font for the printed manual and by surrounding it with asterisks for Info. See Section 9.2.1 [Emphasizing Text], page 83.

`@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 50.

`@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 50.

`@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 51.

`@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 51.

`@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 35.

`@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 3.5 [Generating a Table of Contents], page 38.

`@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 13.4 [Combining Indices], page 110.

`@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 13.4 [Combining Indices], page 110.

`@t{`*text*`}` Print *text* in a `fixed-width`, typewriter-like font. No effect in Info. See Section 9.2.3 [Fonts], page 84.

`@tab` Separate columns in a multitable. See Section 11.5.2 [Multitable Rows], page 100.

`@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*. Pair with `@end table`. See Section 11.4 [Making a Two-column Table], page 97. Also see Section 11.4.2 [`@ftable` and `@vtable`], page 99, and Section 11.4.3 [`@itemx`], page 99.

`@TeX{}` Insert the logo TEX. See Section 14.5.1 [TEX and LATEX], page 117.

`@tex` Enter TEX completely. Pair with `@end tex`. See Section 17.3 [Raw Formatter Commands], page 141.

`@thischapter`
`@thischaptername`
`@thisfile`
`@thispage`
`@thistitle`
: Only allowed in a heading or footing. Stands for the number and name of the current chapter (in the format 'Chapter 1: Title'), the chapter name only, the filename, the current page number, and the title of the document, respectively. See Section E.4 [How to Make Your Own Headings], page 228.

`@tie{}` Generate a normal interword space at which a line break is not allowed. See Section 15.5 [`@tie{}`], page 125.

`@tieaccent{`*cc*`}`
Generate a tie-after accent over the next two characters *cc*, as in 'o͡o'. See Section 14.3 [Inserting Accents], page 116.

`@tindex` *entry*
Add *entry* to the index of data types. See Section 13.1 [Defining the Entries of an Index], page 108.

`@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 35.

`@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 34.

`@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 E.4 [How to Make Your Own Headings], page 228.

`@top` *title*
In a Texinfo file to be formatted with `makeinfo`, identify the topmost `@node` 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 `@ifnottex` and `@end ifnottex`. In TEX and `texinfo-format-buffer`, the `@top` command is merely a synonym for `@unnumbered`. See Section 6.4 [Creating Pointers with `makeinfo`], page 58.

`@u{`*c*`}`
`@ubaraccent{`*c*`}`
`@udotaccent{`*c*`}`
Generate a breve, underbar, or underdot accent, respectively, over or under the character *c*, as in o̍, o̲, ọ. See Section 14.3 [Inserting Accents], page 116.

`@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 49.

`@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 50.

`@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 50.

`@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 51.

`@uref{`*url*`[, `*displayed-text*`][, `*replacement*`}`
`@url{`*url*`[, `*displayed-text*`][, `*replacement*`}`
: Define a cross reference to an external uniform resource locator for the World Wide Web. See Section 8.9 [`@uref`], page 72.

`@v{`*c*`}`
: Generate check accent over the character *c*, as in ŏ. See Section 14.3 [Inserting Accents], page 116.

`@value{`*flag*`}`
: Replace *flag* with the value to which it is set by `@set` *flag*. See Section 17.4 [`@set @clear @value`], page 142.

`@var{`*metasyntactic-variable*`}`
: Highlight a metasyntactic variable, which is something that stands for another piece of text. See Section 9.1.7 [Indicating Metasyntactic Variables], page 79.

`@verb{`*delim literal delim*`}`
: Output *literal*, delimited by the single character *delim*, exactly as is (in the fixed-width font), including any whitespace or Texinfo special characters. See Section 9.1.6 [`verb`], page 78.

`@verbatim`
: Output the text of the environment exactly as is (in the fixed-width font). Pair with `@end verbatim`. See Section 10.4 [`verbatim`], page 87.

`@verbatiminclude` *filename*
: Output the contents of *filename* exactly as is (in the fixed-width font). See Section 10.5 [`verbatiminclude`], page 88.

`@vindex` *entry*
: Add *entry* to the index of variables. See Section 13.1 [Defining the Entries of an Index], page 108.

`@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 [Copyright], page 36.

`@vtable` *`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 variables. Pair with `@end vtable`. The same as `@table`, except for indexing. See Section 11.4.2 [`@ftable` and `@vtable`], page 99.

`@w{`*`text`*`}` Prevent *text* from being split across two lines. See Section 15.4 [`@w`], page 124.

`@xml` Enter XML completely. Pair with `@end xml`. See Section 17.3 [Raw Formatter Commands], page 141.

`@xref{`*`node-name`*`, [`*`entry`*`], [`*`topic-or-title`*`], [`*`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 8.4 [`@xref`], page 66.

A.1 @-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: `@.`, `@:`, `@*`, `@`*`SPACE`*, `@`*`TAB`*, `@`*`NL`*, `@@`, `@{`, 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 require arguments within braces.
   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); for example, `@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 classes that have different argument syntaxes. 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 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. It is also no longer needed, since all formatters now refill paragraphs automatically.

Appendix B Tips and Hints

Here are some tips for writing Texinfo documentation:

- Write in the present tense, not in the past or the future.
- Write actively! For example, write "We recommend that ..." rather than "It is recommended that ...".
- Use 70 or 72 as your fill column. Longer lines are hard to read.
- Include a copyright notice and copying permissions.

Index, Index, Index!

Write many index entries, in different ways. Readers like indices; they are helpful and convenient.

Although it is easiest to write index entries as you write the body of the text, some people prefer to write entries afterwards. In either case, write an entry before the paragraph to which it applies. This way, an index entry points to the first page of a paragraph that is split across pages.

Here are more hints we have found valuable:

- Write each index entry differently, so each entry refers to a different place in the document.
- Write index entries only where a topic is discussed significantly. For example, it is not useful to index "debugging information" in a chapter on reporting bugs. Someone who wants to know about debugging information will certainly not find it in that chapter.
- Consistently capitalize the first word of every concept index entry, or else consistently use lower case. Terse entries often call for lower case; longer entries for capitalization. Whichever case convention you use, please use one or the other consistently! Mixing the two styles looks bad.
- Always capitalize or use upper case for those words in an index for which this is proper, such as names of countries or acronyms. Always use the appropriate case for case-sensitive names, such as those in C or Lisp.
- Write the indexing commands that refer to a whole section immediately after the section command, and write the indexing commands that refer to a paragraph before that paragraph.

  In the example that follows, a blank line comes after the index entry for "Leaping":

```
@section The Dog and the Fox
@cindex Jumping, in general
@cindex Leaping

@cindex Dog, lazy, jumped over
@cindex Lazy dog jumped over
@cindex Fox, jumps over dog
@cindex Quick fox jumps over dog
The quick brown fox jumps over the lazy dog.
```

  (Note that the example shows entries for the same concept that are written in different ways—'`Lazy dog`', and '`Dog, lazy`'—so readers can look up the concept in different ways.)

Blank Lines

- Insert a blank line between a sectioning command and the first following sentence or paragraph, or between the indexing commands associated with the sectioning command and the first following sentence or paragraph, as shown in the tip on indexing. Otherwise, a formatter may fold title and paragraph together.
- Always insert a blank line before an `@table` command and after an `@end table` command; but never insert a blank line after an `@table` command or before an `@end table` command.

  For example,

```
Types of fox:

@table @samp
@item Quick
Jump over lazy dogs.

@item Brown
Also jump over lazy dogs.
@end table

@noindent
On the other hand, ...
```

  Insert blank lines before and after `@itemize ... @end itemize` and `@enumerate ... @end enumerate` in the same way.

Complete Phrases

Complete phrases are easier to read than ...

- Write entries in an itemized list as complete sentences; or at least, as complete phrases. Incomplete expressions ... awkward ... like this.
- Write the prefatory sentence or phrase for a multi-item list or table as a complete expression. Do not write "You can set:"; instead, write "You can set these variables:". The former expression sounds cut off.

Editions, Dates and Versions

Include edition numbers, version numbers, and dates in the `@copying` text (for people reading the Texinfo file, and for the legal copyright in the output files). Then use `@insertcopying` in the `@titlepage` section (for people reading the printed output) and the Top node (for people reading the online output).

It is easiest to do this using `@set` and `@value`. See Section 17.4.3 [`@value` Example], page 145, and Section C.2 [GNU Sample Texts], page 217.

Definition Commands

Definition commands are `@deffn`, `@defun`, `@defmac`, and the like, and enable you to write descriptions in a uniform format.

- Write just one definition command for each entity you define with a definition command. The automatic indexing feature creates an index entry that leads the reader to the definition.
- Use `@table ... @end table` in an appendix that contains a summary of functions, not `@deffn` or other definition commands.

Capitalization

- Capitalize "Texinfo"; it is a name. Do not write the '`x`' or '`i`' in upper case.
- Capitalize "Info"; it is a name.
- Write TEX using the `@TeX{}` command. Note the uppercase '`T`' and '`X`'. This command causes the formatters to typeset the name according to the wishes of Donald Knuth, who wrote TEX.

Spaces

Do not use spaces to format a Texinfo file, except inside of `@example ... @end example` and other literal environments and commands.

For example, TEX fills the following:

```
@kbd{C-x v}
@kbd{M-x vc-next-action}
   Perform the next logical operation
   on the version-controlled file
   corresponding to the current buffer.
```

so it looks like this:

> *C-x v M-x vc-next-action* Perform the next logical operation on the version-controlled file corresponding to the current buffer.

In this case, the text should be formatted with `@table`, `@item`, and `@itemx`, to create a table.

@code, @samp, @var, and '---'

- Use `@code` around Lisp symbols, including command names. For example,

  ```
  The main function is @code{vc-next-action}, ...
  ```

- Avoid putting letters such as '`s`' immediately after an '`@code`'. Such letters look bad.
- Use `@var` around meta-variables. Do not write angle brackets around them.
- Use three hyphens in a row, '`---`', to indicate a long dash. TEX typesets these as a long dash and the Info formatters reduce three hyphens to two.

Periods Outside of Quotes

Place periods and other punctuation marks *outside* of quotations, unless the punctuation is part of the quotation. This practice goes against publishing conventions in the United States, but enables the reader to distinguish between the contents of the quotation and the whole passage.

For example, you should write the following sentence with the period outside the end quotation marks:

```
Evidently, ‘au’ is an abbreviation for ‘‘author’’.
```

since ‘`au`’ does *not* serve as an abbreviation for ‘`author.`’ (with a period following the word).

Introducing New Terms

- Introduce new terms so that a reader who does not know them can understand them from context; or write a definition for the term.

  For example, in the following, the terms “check in”, “register” and “delta” are all appearing for the first time; the example sentence should be rewritten so they are understandable.

  > The major function assists you in checking in a file to your version control system and registering successive sets of changes to it as deltas.

- Use the `@dfn` command around a word being introduced, to indicate that the reader should not expect to know the meaning already, and should expect to learn the meaning from this passage.

`@pxref`

Absolutely never use `@pxref` except in the special context for which it is designed: inside parentheses, with the closing parenthesis following immediately after the closing brace. One formatter automatically inserts closing punctuation and the other does not. This means that the output looks right both in printed output and in an Info file, but only when the command is used inside parentheses.

Invoking from a Shell

You can invoke programs such as Emacs, GCC, and `gawk` from a shell. The documentation for each program should contain a section that describes this. Unfortunately, if the node names and titles for these sections are all different, they are difficult for users to find.

So, there is a convention to name such sections with a phrase beginning with the word ‘Invoking’, as in ‘Invoking Emacs’; this way, users can find the section easily.

ANSI C Syntax

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.

Bad Examples

Here are several examples of bad writing to avoid:

In this example, say, " . . . you must `@dfn{`check in`}` the new version." That flows better.

> When you are done editing the file, you must perform a `@dfn{`check in`}`.

In the following example, say, ". . . makes a unified interface such as VC mode possible."

> SCCS, RCS and other version-control systems all perform similar functions in broadly similar ways (it is this resemblance which makes a unified control mode like this possible).

And in this example, you should specify what 'it' refers to:

> If you are working with other people, it assists in coordinating everyone's changes so they do not step on each other.

And Finally . . .

- Pronounce TEX as if the '`X`' were a Greek 'chi', as the last sound in the name 'Bach'. But pronounce Texinfo as in 'speck': "teckinfo".
- Write notes for yourself at the very end of a Texinfo file after the `@bye`. None of the formatters process text after the `@bye`; it is as if the text were within `@ignore` . . . `@end ignore`.

Appendix C Sample Texinfo Files

The first example is from the first chapter (see Section 1.11 [Short Sample], page 12), given here in its entirety, without commentary. The second includes the full texts to be used in GNU manuals.

C.1 Short Sample

Here is a complete, short sample Texinfo file, without any commentary. You can see this file, with comments, in the first chapter. See Section 1.11 [Short Sample], page 12.

In a nutshell: The `makeinfo` program transforms a Texinfo source file such as this into an Info file or HTML; and TEX typesets it for a printed manual.

```
\input texinfo   @c -*-texinfo-*-
@c %**start of header
@setfilename sample.info
@settitle Sample Manual 1.0
@c %**end of header

@copying
This is a short example of a complete Texinfo file.

Copyright (C) 2004 Free Software Foundation, Inc.
@end copying

@titlepage
@title Sample Title
@page
@vskip 0pt plus 1filll
@insertcopying
@end titlepage

@c Output the table of the contents at the beginning.
@contents

@ifnottex
@node Top
@top GNU Sample

@insertcopying
@end ifnottex

@menu
* First Chapter::    The first chapter is the
                      only chapter in this sample.
* Index::            Complete index.
@end menu
```

```
@node First Chapter
@chapter First Chapter

@cindex chapter, first

This is the first chapter.
@cindex index entry, another

Here is a numbered list.

@enumerate
@item
This is the first item.

@item
This is the second item.
@end enumerate


@node Index
@unnumbered Index

@printindex cp

@bye
```

C.2 GNU Sample Texts

Following is a sample Texinfo document with the full texts that should be used in GNU manuals.

As well as the legal texts, it also serves as a practical example of how many elements in a GNU system can affect the manual. If you're not familiar with all these different elements, don't worry. They're not required and a perfectly good manual can be written without them. They're included here nonetheless because many manuals do (or could) benefit from them.

See Section 1.11 [Short Sample], page 12, for a minimal example of a Texinfo file. See Chapter 3 [Beginning a File], page 28, for a full explanation of that minimal example.

Here are some notes on the example:

- The '`$Id:`' comment is for the CVS (see section "Overview" in *Concurrent Versions System*) or RCS (see rcsintro(1)) version control systems, which expand it into a string such as:

```
$Id: texinfo.txi,v 1.128 2004/12/29 15:06:41 karl Exp $
```

(This is useful in all sources that use version control, not just manuals.) You may wish to include the '`$Id:`' comment in the `@copying` text, if you want a completely unambiguous reference to the documentation version.

If you want to literally write `$Id$`, use `@w`: `@w{$}Id$`.

- The '`version.texi`' in the `@include` command is maintained automatically by Automake (see section "Introduction" in *GNU Automake*). It sets the '`VERSION`' and '`UPDATED`' values used elsewhere. If your distribution doesn't use Automake, but you do use Emacs, you may find the time-stamp.el package helpful (see section "Time Stamps" in *The GNU Emacs Manual*).
- The `@syncodeindex` command reflects the recommendation to use only one index where possible, to make it easier for readers to look up index entries.
- The `@dircategory` is for constructing the Info directory. See Section 21.2.4 [Installing Dir Entries], page 179, which includes a variety of recommended category names.
- The 'Invoking' node is a GNU standard to help users find the basic information about command-line usage of a given program. See section "Manual Structure Details" in *GNU Coding Standards*.
- It is best to include the entire GNU Free Documentation License in a GNU manual, unless the manual is only a few pages long. Of course this sample is even shorter than that, but it includes the FDL anyway in order to show one conventional way to do so. The '`fdl.texi`' file is available on the GNU machines and in the Texinfo and other GNU source distributions.

  The FDL provides for omitting itself under certain conditions, but in that case the sample texts given here have to be modified. See Section G.1 [GNU Free Documentation License], page 237.
- If your manual has invariant sections (again, see the license itself for details), then don't forget to change the text here accordingly.
- For documents that express your personal views, feelings or experiences, it is more appropriate to use a license permitting only verbatim copying, rather than the FDL. See Section C.3 [Verbatim Copying License], page 220.

Here is the sample document:

```
\input texinfo   @c -*-texinfo-*-
@comment $Id: texinfo.txi,v 1.128 2004/12/29 15:06:41 karl Exp $
@comment %**start of header
@setfilename sample.info

@finalout
@include version.texi
@settitle GNU Sample @value{VERSION}
@syncodeindex pg cp
@comment %**end of header
@copying
This manual is for GNU Sample
(version @value{VERSION}, @value{UPDATED}),
which is an example in the Texinfo documentation.
```

```
Copyright @copyright{} 2004 Free Software Foundation, Inc.

@quotation
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1 or
any later version published by the Free Software Foundation; with no
Invariant Sections, with the Front-Cover Texts being ``A GNU Manual,''
and with the Back-Cover Texts as in (a) below.  A copy of the
license is included in the section entitled ``GNU Free Documentation
License.''

(a) The FSF's Back-Cover Text is: ``You have freedom to copy and modify
this GNU Manual, like GNU software.  Copies published by the Free
Software Foundation raise funds for GNU development.''
@end quotation
@end copying

@dircategory Texinfo documentation system
@direntry
* sample: (sample)Invoking sample.
@end direntry

@titlepage
@title GNU Sample
@subtitle for version @value{VERSION}, @value{UPDATED}
@author A.U. Thor (@email{bug-texinfo@@gnu.org})
@page
@vskip 0pt plus 1filll
@insertcopying
@end titlepage

@contents

@ifnottex
@node Top
@top GNU Sample

@insertcopying
@end ifnottex

@menu
* Invoking sample::
* Copying This Manual::
* Index::
@end menu
```

```
@node Invoking sample
@chapter Invoking sample

@pindex sample
@cindex invoking @command{sample}

This is a sample manual.  There is no sample program to
invoke, but if there was, you could see its basic usage
and command line options here.


@node Copying This Manual
@appendix Copying This Manual

@menu
* GNU Free Documentation License::  License for copying this manual.
@end menu

@include fdl.texi


@node Index
@unnumbered Index

@printindex cp

@bye
```

C.3 Verbatim Copying License

For software manuals and other documentation, it is important to use a license permitting free redistribution and updating, so that when a free program is changed, the documentation can be updated as well.

On the other hand, for documents that express your personal views, feelings or experiences, it is more appropriate to use a license permitting only verbatim copying.

Here is sample text for such a license permitting verbatim copying only. This is just the license text itself. For a complete sample document, see the previous sections.

```
@copying
This document is a sample for allowing verbatim copying only.

Copyright @copyright{} 2004 Free Software Foundation, Inc.

@quotation
Permission is granted to make and distribute verbatim copies
of this entire document without royalty provided the
copyright notice and this permission notice are preserved.
```

```
@end quotation
@end copying
```

C.4 All-permissive Copying License

For software manuals and other documentation, it is important to use a license permitting free redistribution and updating, so that when a free program is changed, the documentation can be updated as well.

On the other hand, for small supporting files, short manuals (under 300 lines long) and rough documentation (README files, INSTALL files, etc.), the full FDL would be overkill. They can use a simple all-permissive license.

Here is sample text for such an all-permissive license. This is just the license text itself. For a complete sample document, see the previous sections.

```
Copyright © 2004 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
```

Appendix D 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.

Include files let you keep a single large document as a collection of conveniently small parts.

D.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 buffers.texi
```

The name of the file is taken literally, with a single exception: `@value{`*var*`}` references are expanded. This makes it possible to reliably include files in other directories in a distribution. See Section 10.5 [`@verbatiminclude`], page 88, for an example.

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; nothing after `@bye` is formatted.

In the past, you were required 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 GNU Emacs Texinfo mode command, `texinfo-multiple-files-update`, that is designed for `@include` files.

When an included file does not have any node lines in it, the multiple files update command does not try to create a menu entry for it. Consequently, you can include any file, such as a version or an update file without node lines, not just files that are chapters. Small includable files like this are created by Automake (see Section C.2 [GNU Sample Texts], page 217).

D.2 `texinfo-multiple-files-update`

GNU Emacs Texinfo mode provides the `texinfo-multiple-files-update` command. This command creates or updates 'Next', 'Previous', and 'Up' pointers of included files as well as those in the outer or overall Texinfo file, and it creates or updates a main menu in the

outer file. Depending whether you call it with optional arguments, the command 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 *C-u* as a prefix argument:
  - Create or update pointers in the first `@node` line in each included file.
  - Create or update the 'Top' level node pointers of the outer file.
  - Create and insert a master menu in 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, such as *C-u 8*:
  - 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.
  - Create or update the 'Top' level node pointers of the outer or overall file.
  - 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.

D.3 Include Files Requirements

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, `@chapter` 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 contain only *one* node, the 'Top' node. It should *not* contain any nodes besides the single 'Top' node. The `texinfo-multiple-files-update` command will not process them.

D.4 Sample File with `@include`

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

... See Appendix C [Sample Texinfo Files], page 216, for
examples of the rest of the frontmatter ...

@ifnottex
@node Top
@top Include Example
@end ifnottex

@include foo.texinfo
@include bar.texinfo
@include concept-index.texinfo
@bye
```

An included file, such as '`foo.texinfo`', might look like this:

```
@node First
@chapter First Chapter

Contents of first chapter ...
```

The full contents of '`concept-index.texinfo`' might be as simple as this:

```
@node Concept Index
@unnumbered Concept Index

@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 entries and a list of 41 `@include` files.

D.5 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 7.6 [Referring to Other Info Files], page 62. Also, see Section 8.4.5 [`@xref` with Four and Five Arguments], page 68.)

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 into smaller files if necessary. This means that you can write menus and cross references without naming the different Texinfo files.

Appendix E 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.)

E.1 Headings Introduced

Texinfo provides standard page 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. Typically, you will use 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, 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.

E.2 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 footing 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.

A single-sided page looks like this:

```
 ------------------------
|                        |
| chapter   page number  |
|                        |
| Start of text ...      |
| ...                    |
|                        |
```

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.

Two pages, side by side as in an open book, look like this:

```
 ------------------------     ------------------------
|                        |   |                        |
| page number     title  |   | chapter   page number  |
|                        |   |                        |
| Start of text ...      |   | More  text ...         |
| ...                    |   | ...                    |
|                        |   |                        |
```

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.

E.3 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 a new chapter on the same page as the last page of the preceding chapter, after skipping some vertical whitespace. Also cause TEX to typeset for single-sided printing. (You can override the headers format with the `@headings double` command; see Section 3.4.6 [The `@headings` Command], page 37.)

`@setchapternewpage odd`
: Specify the double-sided heading format, with chapters on new pages.

Texinfo lacks an `@setchapternewpage even` command.

E.4 How to Make Your Own Headings

You can use the standard headings provided with Texinfo or specify your own. By default, Texinfo has no footers, so if you specify them, the available page size for the main text will be slightly reduced.

Texinfo provides six commands for specifying headings and footings:

- `@everyheading` `@everyfooting` generate page headers and footers that are the same for both even- and odd-numbered pages.
- `@evenheading` and `@evenfooting` command generate headers and footers for even-numbered (left-hand) pages.
- `@oddheading` and `@oddfooting` generate headers and footers for odd-numbered (right-hand) pages.

Write custom heading specifications in the Texinfo file immediately after the `@end titlepage` command. 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:

```
@headings off
@everyheading @thischapter @| @thispage @| @today{}
```

You need to divide the left part from the central part and the central part from the right 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 `@`-commands. The text is printed as if the part were within an ordinary paragraph in the body of the page. The `@`-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...`' series of `@`-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: Title'.

`@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 D [Include Files], page 222, 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:

```
@headings off
@everyheading @emph{Draft!} @| @thispage @| @thischapter
@everyfooting @| @| Version: 0.27: @today{}
```

Beware of overlong titles: they may overlap another part of the header or footer and blot it out.

Appendix F 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.

F.1 `makeinfo` Find Errors

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.

F.2 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, the following cross reference fails to format:

```
(@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.)
```

Incidentally, 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`).)

F.3 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 F.2 [Debugging with Info], page 230, 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: *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: *tex-shell* ----------
```

In this case, TEX produced an accurate and understandable error message:

```
Paragraph ended before @xref was complete.
```

'`@par`' is an internal TEX command of no relevance to Texinfo. '`l.27`' means that TEX detected the problem on line 27 of the Texinfo file. The '`?`' is the prompt TEX uses in this circumstance.

Unfortunately, TEX is not always so helpful, and sometimes you must truly be a Sherlock Holmes to discover what went wrong.

In any case, if you run into a problem like this, you can do one of three things.

1. You can tell TEX to continue running and ignore just this error by typing ⟨RET⟩ at the '`?`' prompt.
2. You can tell TEX to continue running and to ignore all 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-c` (or `C-c C-c`, if you are running a shell inside Emacs).
3. You can tell TEX to stop this run by typing `x` ⟨RET⟩ at the '`?`' prompt.

If you are running TEX inside Emacs, you need to switch to the shell buffer and line at which TEX offers 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 the Texinfo file lacks header lines, 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 3.14159 (Web2c 7.0)
(test.texinfo [1])
*
```

In this case, simply type `\end` ⟨RET⟩ after the asterisk. Then write the header lines in the Texinfo file and run the TEX command again. (Note the use of the backslash, '\'. TEX uses '\' instead of '`@`'; and in this circumstance, you are working directly with TEX, not with Texinfo.)

F.4 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 (*C-u* as prefix argument, 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, indented by hierarchical level. For example, here is a part of what was produced by running `texinfo-show-structure` on this manual:

```
Lines matching "^@\\(chapter \\|sect\\|subs\\|subh\\|
unnum\\|major\\|chapheading \\|heading \\|appendix\\)"
in buffer texinfo.texi.
...
4177:@chapter Nodes
4198:    @heading Two Paths
4231:    @section Node and Menu Illustration
4337:    @section The @code{@@node} Command
4393:        @subheading Choosing Node and Pointer Names
4417:        @subsection How to Write an @code{@@node} Line
4469:        @subsection @code{@@node} Line Tips
...
```

This says that lines 4337, 4393, and 4417 of '`texinfo.texi`' begin with the `@section`, `@subheading`, and `@subsection` 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 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 n 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.

F.5 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.

F.6 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.

F.6.1 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 F.6.3 [Tagifying], page 236.

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 do not 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 entries and cross references point to actual nodes.

`Info-validate` requires a tag table and does not work with files that have been split. (The `texinfo-format-buffer` command automatically splits large files.) 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.

F.6.2 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 300,000 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.

F.6.3 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.

F.6.4 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 Section 21.1 [Creating an Info File], page 168.)

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 F.6.1 [Using Info-validate], page 235, 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 Info 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 G Copying This Manual

G.1 GNU Free Documentation License

Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

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

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

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

1. APPLICABILITY AND DEFINITIONS

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

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

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

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

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

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

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

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

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

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

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

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

3. COPYING IN QUANTITY

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

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

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

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

4. MODIFICATIONS

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

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

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

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

D. Preserve all the copyright notices of the Document.

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

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

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

H. Include an unaltered copy of this License.

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

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

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

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

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

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

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

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

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

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

5. COMBINING DOCUMENTS

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

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

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

6. COLLECTIONS OF DOCUMENTS

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

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

7. AGGREGATION WITH INDEPENDENT WORKS

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

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

8. TRANSLATION

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

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

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

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

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

G.1.1 ADDENDUM: How to use this License for your documents

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

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

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

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

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

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

Command and Variable Index

This is an alphabetical list of all the `@`-commands, assorted Emacs Lisp functions, and several variables. To make the list easier to use, the commands are listed without their preceding '`@`'.

!

`!` (end of sentence) 114

"

`"` (umlaut accent) 116

’

`’` (umlaut accent) 116

*

`*` (force line break) 123

,

`,` (cedilla accent) 116

-

`-` (discretionary hyphen) 124

.

`.` (end of sentence) 114

/

`/` (allow line break) 123

<

`<colon>` (suppress end-of-sentence space) 114
`<newline>` 115
`<space>` 115
`<tab>` 115

=

`=` (macron accent) 116

?

`?` (end of sentence) 114

@

`@` (literal '`@`') 113

^

`^` (circumflex accent) 116

‘

`‘` (grave accent) 116

{

`{` (literal '`{`') 113

}

`}` (literal '`}`') 113

\

`\` (literal `\` in `@math`) 119
`\emergencystretch` 163
`\gdef` within `@tex` 142
`\input` (raw TEX startup) 10
`\mag` (raw TEX magnification) 165

~

`~` (tilde accent) 116

A

`aa` 116
`AA` 116
`abbr` 81
`acronym` 81
`ae` 116
`AE` 116
`afourlatex` 165
`afourpaper` 164
`afourwide` 165
`alias` 153
`anchor` 59
`appendix` 49
`appendixsec` 50
`appendixsection` 50
`appendixsubsec` 50
`appendixsubsubsec` 51

apply ... 138
asis ... 98
author ... 35

B

b (bold font) ... 84
bullet ... 117
bye ... 45, 46

C

c (comment) ... 10
caption ... 103
cartouche ... 92
center ... 34
centerchap ... 49
chapheading ... 49
chapter ... 48
cite ... 81
code ... 75
columnfractions ... 100
comma ... 113
command ... 80
comment ... 10
contents ... 38
copying ... 32
copyright ... 32, 117
cropmarks ... 165

D

defcodeindex ... 111
defcv ... 135
deffn ... 130
deffnx ... 129
defindex ... 111
definfoenclose ... 154
defivar ... 136
defmac ... 131
defmethod ... 137
defop ... 136
defopt ... 132
defspec ... 131
deftp ... 134
deftypecv ... 135
deftypefn ... 132
deftypefun ... 133
deftypeivar ... 136
deftypeop ... 137
deftypevar ... 134
deftypevr ... 133
defun ... 130
defvar ... 131
defvr ... 131
dfn ... 80
dircategory ... 179
direntry ... 179
display ... 89
dmn ... 115
docbook ... 142
documentdescription ... 41
documentencoding ... 148
documentlanguage ... 147
dotaccent ... 116
dotless ... 116
dots ... 117

E

email ... 82
emph ... 83
end ... 85, 94
end titlepage ... 37
enddots ... 117
enumerate ... 96
env ... 79
equiv ... 121
error ... 121
euro ... 118
evenfooting ... 228
evenheading ... 228
everyfooting ... 228
everyheading ... 228
example ... 86
exampleindent ... 43
exclamdown ... 116
exdent ... 90
expansion ... 120

F

file ... 80
filll TEX dimension ... 36
finalout ... 164
firstparagraphindent ... 43
float ... 102
flushleft ... 90
flushright ... 90
fn-name ... 128
foobar ... 129, 132
footnote ... 106
footnotestyle ... 107
format ... 90
forward-word ... 127
ftable ... 99

G

group ... 125

H

H (Hungarian umlaut accent) ... 116
hbox ... 163
heading ... 50

headings ... 37
headitem ... 100
headword ... 155
html ... 142
hyphenation ... 124

I

i (italic font) ... 84
ifclear ... 144
ifdocbook ... 140, 142
ifhtml ... 140, 142
ifinfo ... 140
ifnotdocbook ... 141
ifnothtml ... 141
ifnotinfo ... 141
ifnotplaintext ... 141
ifnottex ... 141
ifnotxml ... 141
ifplaintext ... 140
ifset ... 144
iftex ... 140
ifxml ... 140, 142
ignore ... 10
image ... 104
include ... 222
indent ... 92
indicateurl ... 82
Info-validate ... 234
inforef ... 72
insertcopying ... 33
isearch-backward ... 129
isearch-forward ... 129
item ... 95, 98, 100
itemize ... 95
itemx ... 99

K

kbd ... 76
kbdinputstyle ... 76
key ... 77

L

l ... 116
L ... 116
LaTeX ... 117
lisp ... 88
listoffloats ... 103
lowersections ... 51

M

macro ... 150
majorheading ... 49
makeinfo-buffer ... 173
makeinfo-kill-job ... 174
makeinfo-recenter-output-buffer ... 174
makeinfo-region ... 173
math ... 118
menu ... 60
minus ... 118
multitable ... 99

N

need ... 126
next-error ... 173
node ... 55
noindent ... 91
novalidate ... 157

O

o ... 116
O ... 116
occur ... 234
occur-mode-goto-occurrence ... 19
oddfooting ... 228
oddheading ... 228
oe ... 116
OE ... 116
option ... 80
ordf ... 116
ordm ... 116

P

page ... 125
page, within @titlepage ... 34
pagesizes ... 165
paragraphindent ... 42
phoo ... 154
point ... 121
pounds ... 118
print ... 120
printindex ... 45
pxref ... 71

Q

questiondown ... 116
quotation ... 86

R

r (roman font) ... 84
raisesections ... 51
ref ... 70
registeredsymbol ... 118
result ... 120
ringaccent ... 116
rmacro ... 150

S

samp ... 78
sansserif (sans serif font) ... 84
sc (small caps font) ... 83
section ... 49
setchapternewpage ... 41
setcontentsaftertitlepage ... 39
setfilename ... 30
setshortcontentsaftertitlepage ... 39
settitle ... 31
shortcaption ... 103
shortcontents ... 38
shorttitlepage ... 34
slanted (slanted font) ... 84
smallbook ... 164
smalldisplay ... 89
smallexample ... 89
smallformat ... 89, 90
smalllisp ... 89
sp (line spacing) ... 125
sp (titlepage line spacing) ... 34
ss ... 116
strong ... 83
subheading ... 50
subsection ... 50
subsubheading ... 51
subsubsection ... 51
subtitle ... 35
summarycontents ... 38
syncodeindex ... 110
synindex ... 111

T

t (typewriter font) ... 84
tab ... 100
table ... 97
tex ... 142
TeX ... 117
texinfo-all-menus-update ... 21
texinfo-every-node-update ... 21
texinfo-format-buffer ... 24, 174
texinfo-format-region ... 24, 174
texinfo-indent-menu-description ... 23
texinfo-insert-@code ... 17
texinfo-insert-@dfn ... 17
texinfo-insert-@end ... 17
texinfo-insert-@example ... 18
texinfo-insert-@item ... 17
texinfo-insert-@kbd ... 17
texinfo-insert-@node ... 17
texinfo-insert-@noindent ... 17
texinfo-insert-@samp ... 17
texinfo-insert-@table ... 17
texinfo-insert-@var ... 18
texinfo-insert-braces ... 18
texinfo-insert-node-lines ... 23
texinfo-make-menu ... 21
texinfo-master-menu ... 20
texinfo-multiple-files-update ... 222
texinfo-multiple-files-update (in brief) ... 23
texinfo-sequential-node-update ... 23
texinfo-show-structure ... 19, 233
texinfo-start-menu-description ... 18
texinfo-tex-buffer ... 25
texinfo-tex-print ... 25
texinfo-tex-region ... 25
texinfo-update-node ... 21
thischapter ... 229
thischaptername ... 229
thisfile ... 229
thispage ... 229
thistitle ... 229
tie (unbreakable interword space) ... 125
tieaccent ... 116
title ... 35
titlefont ... 34
titlepage ... 34
today ... 229
top ... 39
top (@-command) ... 58

U

u (breve accent) ... 116
ubaraccent ... 116
udotaccent ... 116
unmacro ... 151
unnumbered ... 49
unnumberedsec ... 50
unnumberedsubsec ... 50
unnumberedsubsubsec ... 51
up-list ... 18
uref ... 72

V

v (check accent) ... 116
value ... 143
var ... 79
verb ... 78
verbatim ... 87
verbatiminclude ... 88
vskip TeX vertical skip ... 36
vtable ... 99

W

w (prevent line break) ... 124

X

xml ... 142
xref ... 66

Concept Index

!

¡ ... 116

$

$Id ... 217
$Id expansion, preventing ... 124

(

(dir) as Up node of Top node ... 58

-

--commands-in-node-names ... 169
--css-include ... 169
--delete ... 180
--dir-file=*name* ... 180
--docbook ... 169
--enable-encoding ... 148, 169
--entry=*text* ... 180
--error-limit=*limit* ... 169
--fill-column=*width* ... 169
--footnote-style, ignored in HTML output ... 182
--footnote-style=*style* ... 169
--force ... 169
--help ... 169, 180
--html ... 169
--ifdocbook ... 170
--ifhtml ... 170
--ifinfo ... 170
--ifplaintext ... 170
--iftex ... 170
--ifxml ... 170
--info-dir=*dir* ... 181
--info-file=*file* ... 180
--item=*text* ... 181
--macro-expand=*file* ... 170
--no-headers ... 170
--no-ifdocbook ... 170
--no-ifhtml ... 170
--no-ifinfo ... 170
--no-ifplaintext ... 170
--no-iftex ... 170
--no-ifxml ... 170
--no-number-footnotes ... 170
--no-number-sections ... 171
--no-pointer-validate ... 171
--no-split ... 171
--no-validate ... 171
--no-warn ... 171
--number-sections ... 171
--output=*file* ... 171
--paragraph-indent=*indent* ... 171
--plaintext ... 170
--quiet ... 181
--reference-limit=*limit* ... 171
--remove ... 181
--section=*sec* ... 181
--split-size=*num* ... 172
--verbose ... 172
--version ... 172, 181
--xml ... 172
-D *dir* ... 181
-d *name* ... 180
-D *var* ... 169
-E *file* ... 170
-e *limit* ... 169
-e *text* ... 180
-F ... 169
-f *width* ... 169
-h ... 169, 180
-I *dir* ... 170
-i *file* ... 180
-o *file* ... 171
-P *dir* ... 171
-p *indent* ... 171
-r ... 181
-r *limit* ... 171
-s *sec* ... 181
-s *style* ... 169
-V ... 172, 181

.

.cshrc initialization file ... 162
.profile initialization file ... 162

<

<abbr> tag ... 81
<acronym> tag ... 81
<blockquote> HTML tag ... 86
<lineannotation> Docbook tag ... 84
<meta> HTML tag, and charset specification ... 148
<meta> HTML tag, and document description ... 41
<note> Docbook tag ... 86
<small> tag ... 83
<thead> HTML tag ... 100
<title> HTML tag ... 31
<URL: convention, not used ... 73

?

¿ ... 116

@

'@' as continuation in definition commands 128
@-command list 190
@-command syntax 209
@-commands 8
@-commands in @node, limited support 172
@-commands in nodename 57
@import specifications, in CSS files 183
@include file sample 224
@menu parts 61
@node line writing 56
@value in @node lines 173
@w, for blank items 95

\

'\input' source line ignored 30

8

8-bit characters, in HTML cross-references 188

A

a 116
A4 paper, printing on 164
A5 paper, printing on 164
å 116
Å 116
Abbreviations for keys 77
Abbreviations, tagging 81
Abstract of document 41
Accents, inserting 116
Acronyms, tagging 81
Acute accent 116
Adding a new Info file 177
æ 116
Æ 116
Aliases, command 153
All-permissive copying license 221
Allow line break 123
Alphabetical @-command list 190
Alt attribute for images 104
Anchors 59
Another Info directory 177
Arguments, repeated and optional 129
ASCII text output 170
Aspect ratio of images 105
autoexec.bat 178
automake, and version info 218
Automatic pointer creation with makeinfo 58
Automatically insert nodes, menus 19
Auxiliary files, avoiding 157

B

B5 paper, printing on 165
Back-end output formats 4
Backslash in macros 150
Backslash, and macros 151
Badly referenced nodes 234
Bastard title page 34
Batch formatting for Info 175
Beebe, Nelson 5
Beginning a Texinfo file 28
Beginning line of a Texinfo file 30
Berry, Karl 14
Big points 105
Black rectangle in hardcopy 164
Blank lines 125
Blank lines, as paragraph separator 9
Body of a macro 150
Bold font 84
Bolio 14
Book characteristics, printed 7
Book, printing small 164
border-pattern of Window 135, 136
BoTEX 14
Box with rounded corners 92
Box, ugly black in hardcopy 164
Braces and argument syntax 210
Braces, in macro arguments 151
Braces, inserting 113
Braces, when to use 8
Breaks in a line 123
Breve accent 116
Buffer formatting and printing 24
Bugs, reporting 3
Bullets, inserting 117

C

Captions, for floats 103
Caron accent 116
Cascading Style Sheets, and HTML output ... 183
Case in node name 57
Case, not altering in @code 75
Catching errors with Info formatting 230
Catching errors with TEX formatting 231
Catching mistakes 230
Categories, choosing 179
Caveats for macro usage 152
Cedilla accent 116
Centimeters 105
Chapter structuring 47
Chapters, formatting one at a time 157
Character set, declaring 148
Characteristics, printed books or manuals 7
Characters, basic input 9
Characters, invalid in node name 57
Chassell, Robert J. 14
Check accent 116
Checking for badly referenced nodes 234
Checklist for bug reports 3
Ciceros 105

Circumflex accent ... 116
`code`, value for `@kbdinputstyle` ... 77
Collapsing whitespace around continuations ... 128
Colon in nodename ... 57
Colon, last in `INFOPATH` ... 178
Column widths, defining for multitables ... 100
Combining indices ... 110
Comma in nodename ... 57
Comma, in macro arguments ... 151
Command aliases ... 153
Command definitions ... 138
Command names, indicating ... 80
Command syntax ... 209
Commands to insert special characters ... 113
Commands using raw TEX ... 141
Commands, inserting them ... 17
Commas, inserting ... 113
Comments ... 10
Comments, in CSS files ... 183
Compile command for formatting ... 161
Compressed files, reading ... 180
Conditionally visible text ... 140
Conditionals, nested ... 146
Conditions for copying Texinfo ... 2
Contents, after title page ... 39
Contents, Table of ... 38
Contents-like outline of file structure ... 19
Continuation lines in definition commands ... 128
Conventions for writing definitions ... 138
Conventions, syntactic ... 9
Copying conditions ... 2
Copying Permissions ... 32
Copying software ... 44
Copying text, including ... 33
Copyright holder for FSF works ... 33
Copyright page ... 36
Copyright page, for plain text ... 33
Copyright word, always in English ... 32
Correcting mistakes ... 230
`cp` (concept) index ... 108
Create nodes, menus automatically ... 19
Creating an Info file ... 168
Creating an unsplit file ... 235
Creating index entries ... 109
Creating pointers with `makeinfo` ... 58
Cropmarks for printing ... 165
Cross reference parts ... 65
Cross references ... 64
Cross references using `@inforef` ... 72
Cross references using `@pxref` ... 71
Cross references using `@ref` ... 70
Cross references using `@xref` ... 66
Cross-reference targets, arbitrary ... 59
Cross-references, in HTML output ... 184
CSS, and HTML output ... 183
Custom page sizes ... 165
Customize Emacs package (`Development/Docs/Texinfo`) ... 161
Customized highlighting ... 154
Customizing of TEX for Texinfo ... 162
CVS $Id ... 217

D

Dashes in source ... 9
Debugging the Texinfo structure ... 230
Debugging with Info formatting ... 230
Debugging with TEX formatting ... 231
Default font ... 84
Defining indexing entries ... 109
Defining macros ... 150
Defining new indices ... 111
Defining new Texinfo commands ... 150
Definition command headings, continuing ... 128
Definition commands ... 127
Definition conventions ... 138
Definition lists, typesetting ... 98
Definition template ... 127
Definitions grouped together ... 129
Delimiter character, for verbatim ... 78
Depth of text area ... 165
Description for menu, start ... 18
Description of document ... 41
Details of macro usage ... 152
Didôt points ... 105
Different cross reference commands ... 64
Dimension formatting ... 115
Dimensions and image sizes ... 105
Dir categories, choosing ... 179
'`dir`' directory for Info installation ... 176
'`dir`' file listing ... 177
'`dir`' file, creating your own ... 179
'`dir`' files and Info directories ... 177
Dir files, compressed ... 180
'`dir`', created by `install-info` ... 180
Display formatting ... 89
Displayed equations ... 119
`distinct`, value for `@kbdinputstyle` ... 77
Distorting images ... 105
Distribution ... 44
Docbook output ... 5
Docbook, including raw ... 141
Document description ... 41
Document input encoding ... 148
Document language, declaring ... 147
Document Permissions ... 32
Documentation identification ... 217
Dot accent ... 116
Dotless i, j ... 116
Dots, inserting ... 117
Double-colon menu entries ... 61
DTD, for Texinfo XML ... 5
Dumas, Patrice ... 184
Dumping a .fmt file ... 163
DVI file ... 156
DVI output ... 4

`dvips` ... 4

E

Ellipsis, inserting ... 117
Em dash, compared to minus sign ... 118
Em dash, producing ... 9
Emacs ... 16
Emacs shell, format, print from ... 159
Emacs-W3 ... 4
Emphasizing text ... 83
Emphasizing text, font for ... 83
En dash, producing ... 9
`enable` ... 134
Encoding, declaring ... 148
'End' node footnote style ... 107
End of header line ... 32
End titlepage starts headings ... 37
Ending a Sentence ... 114
Ending a Texinfo file ... 45
Entries for an index ... 109
Entries, making index ... 108
Enumeration ... 96
Environment indentation ... 43
Environment variable `INFOPATH` ... 178
Environment variable `TEXINFO_OUTPUT_FORMAT` ... 172
Environment variable `TEXINPUTS` ... 162
`epsf.tex` ... 105
`epsf.tex`, installing ... 162
Equations, displayed ... 119
Equivalence, indicating ... 121
Error message, indicating ... 121
Errors, parsing ... 173
Es-zet ... 116
European A4 paper ... 164
Evaluation glyph ... 120
Example beginning of Texinfo file ... 28
Example indentation ... 43
Example menu ... 61
`example`, value for `@kbdinputstyle` ... 77
Examples in smaller fonts ... 89
Examples, formatting them ... 86
Examples, glyphs for ... 119
Expanding macros ... 151
Expansion of 8-bit characters in HTML cross-references ... 188
Expansion, indicating ... 120
expansion, of node names in HTML cross-references ... 185
Expressions in a program, indicating ... 75

F

FDL, GNU Free Documentation License ... 237
Feminine ordinal ... 116
File beginning ... 28
File ending ... 45
File name collision ... 30
File section structure, showing it ... 19
Final output ... 163
Finding badly referenced nodes ... 234
Fine-tuning, and hyphenation ... 124
First line of a Texinfo file ... 30
First node ... 57
First paragraph, suppressing indentation of ... 43
Fixed-width font ... 84
Float environment ... 102
Floating accents, inserting ... 116
Floating, not yet implemented ... 102
Floats, in general ... 102
Floats, list of ... 103
Floats, making unnumbered ... 102
Floats, numbering of ... 103
Flooding ... 71
`fn` (function) index ... 108
Fonts for indices ... 111
Fonts for printing, not Info ... 84
Footings ... 226
Footnotes ... 106
Force line break ... 123
Forcing indentation ... 92
Forcing line and page breaks ... 123
Format a dimension ... 115
Format and print hardcopy ... 156
Format and print in Texinfo mode ... 159
Format file, dumping ... 163
Format with the compile command ... 161
Format, print from Emacs shell ... 159
Formats for images ... 104
Formatting a file for Info ... 168
Formatting commands ... 8
Formatting examples ... 86
Formatting for Info ... 24
Formatting for printing ... 24
Formatting headings and footings ... 226
Formatting requirements ... 161
Formatting with `tex` and `texindex` ... 156
Formulas, mathematical ... 118
Fox, Brian ... 14
Free Documentation License, including entire ... 218
Free Software Directory ... 179
Frequently used commands, inserting ... 17
Frontmatter, text in ... 28
Full texts, GNU ... 217
Function definitions ... 138

G

General syntactic conventions ... 9
Generating menus with indices ... 45
Generating plain text files ... 170
German S ... 116
GIF images, unsupported due to patents ... 105
Global Document Commands ... 41

Globbing ... 156
Glyphs ... 119
GNU Emacs ... 16
GNU Emacs shell, format, print from ... 159
GNU Free Documentation License, including entire ... 218
GNU sample texts ... 217
Going to other Info files' nodes ... 62
Grave accent ... 116
Grave accent, vs. left quote ... 9
Group (hold text together vertically) ... 125
Grouping two definitions together ... 129

H

Hacek accent ... 116
Hardcopy, printing it ... 156
'`hboxes`', overfull ... 163
Header for Texinfo files ... 29
Header of a Texinfo file ... 30
Heading row, in table ... 100
Headings ... 226
Headings, indentation after ... 43
Headings, page, begin to appear ... 37
Height of images ... 105
Height of text area ... 165
`help2man` ... 5
Hierarchical documents, and menus ... 60
Highlighting text ... 74
Highlighting, customized ... 154
Hints ... 211
History of Texinfo ... 14
Holder of copyright for FSF works ... 33
Holding text together vertically ... 125
`href`, producing HTML ... 72
HTML cross-reference 8-bit character expansion ... 188
HTML cross-reference command expansion ... 186
HTML cross-reference link basics ... 184
HTML cross-reference mismatch ... 188
HTML cross-reference node name expansion ... 185
HTML cross-references ... 184
HTML output ... 4, 182
HTML output, and encodings ... 148
HTML output, browser compatibility of ... 182
HTML output, split ... 182
HTML, and CSS ... 183
HTML, including raw ... 141
`http-equiv`, and charset specification ... 148
Hungarian umlaut accent ... 116
Hurricanes ... 70
Hyphen, compared to minus ... 118
Hyphenation patterns, language-dependent ... 147
Hyphenation, helping TEX do ... 124
Hyphenation, preventing ... 124
Hyphens in source, two or three in a row ... 9

I

ı (dotless i) ... 116
Identification of documentation ... 217
If text conditionally visible ... 140
Ignored before `@setfilename` ... 30
Ignored text ... 10
Image formats ... 104
Images, alternate text for ... 104
Images, inserting ... 104
Images, scaling ... 105
Inches ... 105
Include file sample ... 224
Include files ... 222
Include files requirements ... 223
Include files, and section levels ... 51
Including a file verbatim ... 88
Including permissions text ... 33
Indentation undoing ... 90
Indentation, forcing ... 92
Indentation, omitting ... 91
Indenting environments ... 43
Indenting paragraphs, control of ... 42
Indenting, suppressing of first paragraph ... 43
Index entries ... 109
Index entries, making ... 108
Index entry writing ... 109
Index file names ... 156
Index font types ... 109
Indexing table entries automatically ... 99
Indicating commands, definitions, etc. ... 74
Indicating evaluation ... 120
Indices ... 108
Indices, combining them ... 110
Indices, defining new ... 111
Indices, printing and menus ... 45
Indices, sorting ... 156
Indices, two letter names ... 110
Indirect subfiles ... 175
Info batch formatting ... 175
Info file installation ... 176
Info file name, choosing ... 30
Info file, listing a new ... 177
Info file, splitting manually ... 236
Info files ... 5
Info format, and menus ... 60
Info formatting ... 24
Info installed in another directory ... 177
Info output ... 4
Info output, and encoding ... 148
Info validating a large file ... 235
Info, creating an online file ... 168
`Info-directory-list` ... 178
Info; other files' nodes ... 62
`INFOPATH` ... 178
Initialization file for TEX input ... 162
Input encoding, declaring ... 148
Insert nodes, menus automatically ... 19
Inserting @ (literal '`@`') ... 113

Inserting accents . . . 116
Inserting dots . . . 117
Inserting ellipsis . . . 117
Inserting frequently used commands . . . 17
Inserting indentation . . . 92
Inserting space . . . 114
Inserting special characters and symbols . . . 113
'INSTALL' file, generating . . . 170
install-info . . . 180
Installing an Info file . . . 176
Installing Info in another directory . . . 177
Internationalization . . . 147
Introduction to Texinfo . . . 3
Introduction, as part of file . . . 44
Invalid characters in node names . . . 57
Invoking macros . . . 151
Invoking nodes, including in dir file . . . 180
ISO 639 codes . . . 147
ISO 8859-15 . . . 118
Italic font . . . 84
Itemization . . . 95

J

ȷ (dotless j) . . . 116
JPG image format . . . 105

K

Keyboard input . . . 76
Keys, recommended names . . . 77
Keyword expansion, preventing . . . 124
Keywords, indicating . . . 75
Knuth, Donald . . . 7
ky (keystroke) index . . . 108

L

ł . . . 116
Ł . . . 116
Language codes . . . 147
Language, declaring . . . 147
Larger or smaller pages . . . 165
LaTeX, processing with texi2dvi . . . 158
Latin 9 . . . 118
Legal paper, printing on . . . 165
Length of file names . . . 30
Less cluttered menu entry . . . 61
License agreement . . . 44
License for all-permissive copying . . . 221
License for verbatim copying . . . 220
Line breaks . . . 123
Line breaks, preventing . . . 124
Line length, column widths as fraction of . . . 100
Line spacing . . . 125
Lisp example . . . 88
Lisp examples in smaller fonts . . . 89
List of @-commands . . . 190
List of floats . . . 103
Listing a new Info file . . . 177
Lists and tables, making . . . 94
Local variables . . . 161
Local Variables: section, for encoding . . . 148
Location of menus . . . 60
Looking for badly referenced nodes . . . 234
Lowering and raising sections . . . 51
lpr (DVI print command) . . . 158
lpr-d, replacements on MS-DOS/MS-Windows . . . 158
Lynx . . . 4

M

Macro definitions . . . 138, 150
Macro details . . . 152
Macro expansion, indicating . . . 120
Macro invocation . . . 151
Macro names, valid characters in . . . 150
Macron accent . . . 116
Macros . . . 150
Macros in definition commands . . . 128
Macros, undefining . . . 151
Magnified printing . . . 165
Mailto link . . . 82
makeinfo . . . 168
makeinfo inside Emacs . . . 173
makeinfo options . . . 168
Making a printed manual . . . 156
Making a tag table automatically . . . 175
Making a tag table manually . . . 236
Making cross references . . . 64
Making line and page breaks . . . 123
Making lists and tables . . . 94
Man page output, not supported . . . 5
Man page, reference to . . . 72
Manual characteristics, printed . . . 7
Margins on page, not controllable . . . 165
Marking text within a paragraph . . . 74
Marking words and phrases . . . 74
Masculine ordinal . . . 116
Master menu . . . 40
Mathematical expressions . . . 118, 142
Menu description, start . . . 18
Menu entries with two colons . . . 61
Menu example . . . 61
Menu location . . . 60
Menu parts . . . 61
Menu writing . . . 60
Menu, master . . . 40
Menus . . . 60
Menus generated with indices . . . 45
Menus, omitting . . . 170
META key . . . 78
Meta-syntactic chars for arguments . . . 129
Methods, object-oriented . . . 136
Millimeters . . . 105

Minimal requirements for formatting 161
Minimal Texinfo file (requirements) 10
Mismatched HTML cross-reference source and target 188
Mistakes, catching 230
Mode, using Texinfo 16
Monospace font 84
Mozilla 4
Multiple dashes in source 9
Multiple spaces 115
Multitable column widths 100
Multitable rows 100
Must have in Texinfo file 10
Mutually recursive macros 150

N

Names for indices 110
Names of index files 156
Names of macros, valid characters of 150
Names recommended for keys 77
Naming a 'Top' Node in references 70
NASA, as acronym 81
Navigation bar, in HTML output 182
Navigation links, omitting 170
Need space at page bottom 126
Nesting conditionals 146
New index defining 111
New Info file, listing it in '`dir`' file 177
New Texinfo commands, defining 150
Newlines, as blank lines 9
Next node of Top node 58
Node line requirements 57
Node line writing 56
node name expansion, in HTML cross-references 185
Node name must be unique 57
Node name, should not contain @-commands .. 57
Node names, choosing 55
Node names, invalid characters in 57
Node separators, omitting 170
Node, 'Top' 39
Node, defined 55
Nodes in other Info files 62
Nodes, catching mistakes 230
Nodes, checking for badly referenced 234
Non-breakable space, fixed 124
Non-breakable space, variable 125
Not ending a sentence 114
Numbering of floats 103

O

O'Dea, Brendan 5
ø 116
Ø 116
Object-oriented programming 135
Oblique font 84
Obtaining TEX 166
Occurrences, listing with `@occur` 234
œ 116
Œ 116
Omitting indentation 91
Optional and repeated arguments 129
Options for `makeinfo` 168
Ordinals, Romance 116
Ordinary TEX commands, using 141
Other Info files' nodes 62
Outline of file structure, showing it 19
Output file splitting 171
Output formats 4
Output formats, supporting more 5
Overfull '`hboxes`' 163
Overview of Texinfo 3
Owner of copyright for FSF works 33

P

Page breaks 123, 125
Page delimiter in Texinfo mode 19
Page headings 226
Page numbering 226
Page sizes for books 164
Page sizes, customized 165
`page-delimiter` 19
Pages, starting odd 41
Paper size, A4 164
Paragraph indentation control 42
Paragraph separator 9
Paragraph, marking text within 74
Parameters to macros 150
Parentheses in nodename 57
Parsing errors 173
Part of file formatting and printing 24
Parts of a cross reference 65
Parts of a master menu 40
Parts of a menu 61
Patches, contributing 3
PDF output 5, 166
`pdfcolor.tex`, installing 162
`pdftex` 5, 166
`pdftex`, and images 104
Period in nodename 57
Periods, inserting 114
Permissions text, including 33
Permissions, printed 36
`pg` (program) index 108
Picas 105
Pictures, inserting 104
Pinard, François 14
Plain TEX 141
Plain text output 4, 170
PNG image format 105
Point, indicating in a buffer 121
Pointer creation with `makeinfo` 58
Pointer validation with `makeinfo` 172

Pointer validation, suppressing 157, 171
Points (dimension) 105
Predefined names for indices 110
Preparing for TEX 162
Preventing first paragraph indentation 43
Preventing line and page breaks 123
Previous node of Top node 58
Print and format in Texinfo mode 159
Print, format from Emacs shell 159
Printed book and manual characteristics 7
Printed output, indicating 120
Printed permissions 36
Printing a region or buffer 24
Printing an index 45
Printing cropmarks 165
Printing DVI files, on MS-DOS/MS-Windows 158
Printing hardcopy 156
Problems, catching 230
Program names, indicating 80
Prototype row, column widths defined by 100

Q

Quotation characters (''), in source 9
Quotations 86

R

Ragged left 90
Ragged right 90
Raising and lowering sections 51
Raw formatter commands 141
RCS $Id 217
Recommended names for keys 77
Rectangle, black in hardcopy 164
Recursion, mutual 150
Recursive macro invocations 150
Reference to @-commands 190
References 64
References using `@inforef` 72
References using `@pxref` 71
References using `@ref` 70
References using `@xref` 66
Referring to other Info files 62
Region formatting and printing 24
Region printing in Texinfo mode 159
Reid, Brian 14
Repeated and optional arguments 129
Reporting bugs 3
Required in Texinfo file 10
Requirements for formatting 161
Requirements for include files 223
Requirements for updating commands 22
Reserved words, indicating 75
Restrictions on node names 57
Result of an expression 120
`ridt.eps` 105
Ring accent 116
Roman font 84
Romance ordinals 116
Rounded rectangles, around examples 92
Rows, of a multitable 100
Running an Info formatter 24
Running `Info-validate` 235
Running macros 151
Running `makeinfo` in Emacs 173

S

Sample `@include` file 224
Sample function definition 138
Sample Texinfo file, no comments 216
Sample Texinfo file, with comments 12
Sample Texinfo files 216
Sample texts, GNU 217
Sans serif font 84
Scaled points 105
Scaling images 105
Schwab, Andreas 14
Scribe 14
Sea surges 70
Section structure of a file, showing it 19
Sections, raising and lowering 51
Sentence ending punctuation 114
Sentence non-ending punctuation 114
'Separate' footnote style 107
SGML-tools output format 5
Sharp S 116
Shell formatting with `tex` and `texindex` 156
Shell printing, on MS-DOS/MS-Windows 158
Shell, format, print from 159
Shell, running `makeinfo` in 173
Short captions, for lists of floats 103
Short table of contents 38
Showing the section structure of a file 19
Showing the structure of a file 233
Shrubbery 144
Site-wide Texinfo configuration file 162
Size of printed book 164
Slanted font 84
Slanted typewriter font, for `@kbd` 76
Small book size 164
Small caps font 83
Small examples 89
Software copying permissions 44
Sorting indices 156
Source file format 4
Source files, characters used 9
Space, inserting horizontal 115
Space, inserting vertical 125
Spaces in macros 150
Spaces in node name 57
Spaces, in menus 61
Spacing, inserting 114
Special characters, inserting 113

Special displays ... 102
Special insertions ... 113
Special typesetting commands ... 117
Specifying index entries ... 109
Split HTML output ... 182
Splitting an Info file manually ... 236
Splitting of output files ... 171
ß ... 116
Stallman, Richard M. ... 14
Start of header line ... 30
Starting chapters ... 41
Structure of a file, showing it ... 19
Structure, catching mistakes in ... 230
Structuring of chapters ... 47
Subsection-like commands ... 50
Subsub commands ... 51
Suggestions for Texinfo, making ... 3
Summary of document ... 41
Suppressing first paragraph indentation ... 43
Suppressing indentation ... 91
Syntactic conventions ... 9
Syntactic tokens, indicating ... 75
Syntax, of @-commands ... 209
Syntax, optional & repeated arguments ... 129

T

Table of contents ... 38
Table of contents, after title page ... 39
Table of contents, for floats ... 103
Tables and lists, making ... 94
Tables with indexes ... 99
Tables, making multi-column ... 99
Tables, making two-column ... 97
Tabs; don't use! ... 9
Tag table, making automatically ... 175
Tag table, making manually ... 236
Targets for cross-references, arbitrary ... 59
Template for a definition ... 127
TEX commands, using ordinary ... 141
TEX index sorting ... 156
TEX input initialization ... 162
TEX, how to obtain ... 166
`texi2dvi` ... 157
`texi2dvi` (shell script) ... 157
`texi2roff`, unsupported software ... 7
`texindex` ... 156
Texinfo commands, defining new ... 150
Texinfo file beginning ... 28
Texinfo file ending ... 45
Texinfo file header ... 29
Texinfo file minimum ... 10
Texinfo file section structure, showing it ... 19
Texinfo history ... 14
Texinfo mode ... 16
Texinfo overview ... 3
Texinfo printed book characteristics ... 7
Texinfo requires `@setfilename` ... 30
Texinfo, introduction to ... 3
`texinfo.cnf` ... 31
`texinfo.cnf` installation ... 162
`texinfo.dtd` ... 5
`texinfo.tex`, installing ... 162
`TEXINFO_OUTPUT_FORMAT` ... 172
`TEXINPUTS` ... 162
Text width and height ... 165
Text, conditionally visible ... 140
Text, marking up ... 74
Thin space between number, dimension ... 115
Tie-after accent ... 116
Tied space ... 125
Tilde accent ... 116
`time-stamp.el` ... 218
Tips ... 211
Title page ... 34
Title page, bastard ... 34
Title page, for plain text ... 33
Titlepage end starts headings ... 37
Top node ... 39
Top node example ... 40
Top node is first ... 57
'Top' node naming for references ... 70
`tp` (data type) index ... 108
Tree structuring ... 47
Two 'First' Lines for `@deffn` ... 129
Two letter names for indices ... 110
Two named items for `@table` ... 99
Two part menu entry ... 61
'`txi-cc.tex`' ... 147
Typesetting commands for dots, etc. ... 117
Typewriter font ... 84

U

Ugly black rectangles in hardcopy ... 164
Umlaut accent ... 116
Unbreakable space, fixed ... 124
Unbreakable space, variable ... 125
Uncluttered menu entry ... 61
Undefining macros ... 151
Underbar accent ... 116
Underdot accent ... 116
Unicode quotation characters ... 9
Uniform resource locator, indicating ... 82
Uniform resource locator, referring to ... 72
Unique nodename requirement ... 57
Unnumbered float, creating ... 102
Unprocessed text ... 10
Unsplit file creation ... 235
Up node of Top node ... 58
`UPDATED` Automake variable ... 218
Updating nodes and menus ... 19
Updating requirements ... 22
URI syntax for Info ... 7
URL, indicating ... 82
URL, referring to ... 72

Usage tips . . . 211
User input . . . 76
User options, marking . . . 132
User-defined Texinfo commands . . . 150
Using Texinfo in general . . . 3

V

Validating a large file . . . 235
Validation of pointers . . . 172
Value of an expression, indicating . . . 120
Variables, object-oriented . . . 135
Verbatim copying license . . . 220
Verbatim environment . . . 87
Verbatim in-line text . . . 78
Verbatim, include file . . . 88
`VERSION` Automake variable . . . 218
Version control keywords, preventing expansion of . . . 124
Version number, for install-info . . . 181
Vertically holding text together . . . 125
Visibility of conditional text . . . 140
`vr` (variable) index . . . 108

W

W3 consortium . . . 4
Weinberg, Zack . . . 14
Weisshaus, Melissa . . . 14
White space in node name . . . 57
White space, excessive . . . 123
Whitespace in macros . . . 150
Whitespace, collapsed around continuations . . . 128
Whitespace, inserting . . . 115
Width of images . . . 105
Width of text area . . . 165
Widths, defining multitable column . . . 100
Wildcards . . . 156
Words and phrases, marking them . . . 74
Writing a menu . . . 60
Writing an `@node` line . . . 56
Writing index entries . . . 109

X

`xdvi` . . . 4
XML output . . . 5
XML, including raw . . . 141
`XPM image format` . . . 104

Y

Years, in copyright line . . . 33

Z

Zaretskii, Eli . . . 14
Zuhn, David D. . . . 14