Info

The

On-line, Menu-driven

GNU Documentation System

Published by the Free Software Foundation
675 Massachusetts Avenue,
Cambridge, MA 02139 USA

1 Getting Started

This first part of the Info manual describes how to get around inside of Info. The second part of the manual describes various advanced Info commands, and how to write an Info as distinct from a Texinfo file. The third part is about how to generate Info files from Texinfo files.

This manual is primarily designed for use on a computer, so that you can try Info commands while reading about them. Reading it on paper is less effective, since you must take it on faith that the commands described really do what the manual says. By all means go through this manual now that you have it; but please try going through the on-line version as well.

There are two ways of looking at the online version of this manual:

1. Type `info` at your shell's command line. This approach uses a stand-alone program designed just to read Info files.
2. Type `emacs` at the command line; then type `C-h i` (Control `h`, followed by `i`). This approach uses the Info mode of the Emacs program, an editor with many other capabilities.

In either case, then type `mInfo` (just the letters), followed by ⟨RET⟩—the "Return" or "Enter" key. At this point, you should be ready to follow the instructions in this manual as you read them on the screen.

1.1 Starting Info on a Small Screen

(In Info, you only see this section if your terminal has a small number of lines; most readers pass by it without seeing it.)

Since your terminal has an unusually small number of lines on its screen, it is necessary to give you special advice at the beginning.

If you see the text '`--All----`' at near the bottom right corner of the screen, it means the entire text you are looking at fits on the screen. If you see '`--Top----`' instead, it means that there is more text below that does not fit. To move forward through the text and see another screen full, press the Space bar, ⟨SPC⟩. To move back up, press the key labeled '`Delete`' or ⟨DEL⟩.

1.2 How to use Info

You are talking to the program Info, for reading documentation.

Right now you are looking at one *Node* of Information. A node contains text describing a specific topic at a specific level of detail. This node's topic is "how to use Info".

The top line of a node is its *header*. This node's header (look at it now) says that it is the node named '`Help`' in the file '`info`'. It says that the '`Next`' node after this one is the node called '`Help-P`'. An advanced Info command lets you go to any node whose name you know.

Besides a '`Next`', a node can have a '`Previous`' or an '`Up`'. This node has a '`Previous`' which is '`Help-Small-Screen`', and an '`Up`' which is '`Getting Started`'. Some nodes have no '`Previous`' and some have no '`Up`'.

Now it is time to move on to the '`Next`' node, named '`Help-P`'.

`>>` Type '`n`' to move there. Type just one character;
do not type the quotes and do not type a ⟨RET⟩ afterward.

'`>>`' in the margin means it is really time to try a command.

1.3 Returning to the Previous node

This node is called '`Help-P`'. The '`Previous`' node, as you see, is '`Help`', which is the one you just came from using the *`n`* command. Another *`n`* command now would take you to the next node, '`Help-^L`'.

`>>` But do not do that yet. First, try the *`p`* command, which takes
you to the '`Previous`' node. When you get there, you can do an
`n` again to return here.

This all probably seems insultingly simple so far, but *do not* be led into skimming. Things will get more complicated soon. Also, do not try a new command until you are told it is time to. Otherwise, you may make Info skip past an important warning that was coming up.

`>>` Now do an *`n`* to get to the node '`Help-^L`' and learn more.

1.4 The Space, Delete, B and ^L commands.

This node's header tells you that you are now at node '`Help-^L`', and that *`p`* would get you back to '`Help-P`'. The node's title is underlined; it says what the node is about (most nodes have titles).

This is a big node and it does not all fit on your display screen. You can tell that there is more that is not visible because you can see the string '`--Top-----`' rather than '`--All----`' near the bottom right corner of the screen.

The Space, Delete and *`B`* commands exist to allow you to "move around" in a node that does not all fit on the screen at once. Space moves forward, to show what was below the bottom of the screen. Delete moves backward, to show what was above the top of the screen (there is not anything above the top until you have typed some spaces).

`>>` Now try typing a Space (afterward, type a Delete to return here).

When you type the space, the two lines that were at the bottom of the screen appear at the top, followed by more lines. Delete takes the two lines from the top and moves them to the bottom, *usually*, but if there are not a full screen's worth of lines above them they may not make it all the way to the bottom.

Space and Delete scroll through all the nodes in an Info file as a single logical sequence. In this sequence, a node's subnodes appear following their parent. If a node's menu is on the screen, Space takes you into the subnodes listed in the menu, one by one. Once you reach the end of a node, Space takes you to the next node or back to the parent node.

If your screen is ever garbaged, you can tell Info to print it out again by typing *`C-l`* (*`Control-L`*, that is—hold down "Control" and type an ⟨L⟩ or *`l`*).

`>>` Type *`C-l`* now.

To move back to the beginning of the node you are on, you can type a lot of Deletes. You can also type simply *`b`* for beginning.

`>>` Try that now. (We have put in enough verbiage to push this past
the first screenful, but screens are so big nowadays that perhaps it
isn't enough. You may need to shrink your Emacs or Info window.)
Then come back, with Spaces.

If your screen is very tall, all of this node might fit at once. In that case, "b" won't do anything. Sorry; what can we do?

You have just learned a considerable number of commands. If you want to use one but have trouble remembering which, you should type a ⟨?⟩ which prints out a brief list of commands. When you are finished looking at the list, make it go away by typing a ⟨SPC⟩.

`>>` Type a ⟨?⟩ now. After it finishes, type a ⟨SPC⟩.

(If you are using the standalone Info reader, type 'l' to return here.)

From now on, you will encounter large nodes without warning, and will be expected to know how to use Space and Delete to move around in them without being told. Since not all terminals have the same size screen, it would be impossible to warn you anyway.

`>>` Now type *`n`* to see the description of the *`m`* command.

1.5 Menus

Menus and the *`m`* command

With only the *`n`* and *`p`* commands for moving between nodes, nodes are restricted to a linear sequence. Menus allow a branching structure. A menu is a list of other nodes you can move to. It is actually just part of the text of the node formatted specially so that Info can interpret it. The beginning of a menu is always identified by a line which starts with '`* Menu:`'. A node contains a menu if and only if it has a line in it which starts that way. The only menu you can use at any moment is the one in the node you are in. To use a menu in any other node, you must move to that node first.

After the start of the menu, each line that starts with a '`*`' identifies one subtopic. The line usually contains a brief name for the subtopic (followed by a '`:`'), the name of the node that talks about that subtopic, and optionally some further description of the subtopic. Lines in the menu that do not start with a '`*`' have no special meaning—they are only for the human reader's benefit and do not define additional subtopics. Here is an example:

```
* Foo:  FOO's Node      This tells about FOO
```

The subtopic name is Foo, and the node describing it is '`FOO's Node`'. The rest of the line is just for the reader's Information. [[But this line is not a real menu item, simply because there is no line above it which starts with '`* Menu:`'.]]

When you use a menu to go to another node (in a way that will be described soon), what you specify is the subtopic name, the first thing in the menu line. Info uses it to find the menu line, extracts the node name from it, and goes to that node. The reason that there is both a subtopic name and a node name is that the node name must be meaningful to the computer and may therefore have to be ugly looking. The subtopic name can be chosen just to be convenient for the user to specify. Often the node name is convenient for the user to specify and so both it and the subtopic name are the same. There is an abbreviation for this:

```
* Foo::   This tells about FOO
```

This means that the subtopic name and node name are the same; they are both '`Foo`'.

```
>> Now use Spaces to find the menu in this node, then come back to
   the front with a b and some Spaces.  As you see, a menu is
   actually visible in its node.  If you cannot find a menu in a node
   by looking at it, then the node does not have a menu and the
   m command is not available.
```

The command to go to one of the subnodes is *`m`*—but *do not do it yet!* Before you use *`m`*, you must understand the difference between commands and arguments. So far, you have learned several commands that do not need arguments. When you type one, Info processes it and is instantly ready for another command. The *`m`* command is different: it is incomplete without the *name of the subtopic*. Once you have typed *`m`*, Info tries to read the subtopic name.

Now look for the line containing many dashes near the bottom of the screen. There is one more line beneath that one, but usually it is blank. If it is empty, Info is ready for a command, such as *`n`* or *`b`* or Space or *`m`*. If that line contains text ending in a colon, it mean Info is trying to read the *argument* to a command. At such times, commands do not work, because Info tries to use them as the argument. You must either type the argument and finish the command you started, or type *`Control-g`* to cancel the command. When you have done one of those things, the line becomes blank again.

The command to go to a subnode via a menu is *`m`*. After you type the *`m`*, the line at the bottom of the screen says '`Menu item: `'. You must then type the name of the subtopic you want, and end it with a ⟨RET⟩.

You can abbreviate the subtopic name. If the abbreviation is not unique, the first matching subtopic is chosen. Some menus put the shortest possible abbreviation for each subtopic name in capital letters, so you can see how much you need to type. It does not matter whether you use upper case or lower case when you type the subtopic. You should not put any spaces at the end, or inside of the item name, except for one space where a space appears in the item in the menu.

You can also use the *completion* feature to help enter the subtopic name. If you type the Tab key after entering part of a name, it will magically fill in more of the name—as much as follows uniquely from what you have entered.

If you move the cursor to one of the menu subtopic lines, then you do not need to type the argument: you just type a Return, and it stands for the subtopic of the line you are on.

Here is a menu to give you a chance to practice.

`>>` Now type just an *m* and see what happens:

Now you are "inside" an *m* command. Commands cannot be used now; the next thing you will type must be the name of a subtopic.

You can change your mind about doing the *m* by typing Control-g.

`>>` Try that now; notice the bottom line clear.

`>>` Then type another *m*.

`>>` Now type '`BAR`' item name. Do not type Return yet.

While you are typing the item name, you can use the Delete key to cancel one character at a time if you make a mistake.

`>>` Type one to cancel the '`R`'. You could type another '`R`' to
replace it. You do not have to, since '`BA`' is a valid abbreviation.

`>>` Now you are ready to go. Type a ⟨RET⟩.

After visiting Help-FOO, you should return here.

`>>` Type *n* to see more commands.

1.5.1 The *u* command

Congratulations! This is the node '`Help-FOO`'. Unlike the other nodes you have seen, this one has an '`Up`': '`Help-M`', the node you just came from via the *m* command. This is the usual convention—the nodes you reach from a menu have '`Up`' nodes that lead back to the menu. Menus move Down in the tree, and '`Up`' moves Up. '`Previous`', on the other hand, is usually used to "stay on the same level but go backwards"

You can go back to the node '`Help-M`' by typing the command *u* for "Up". That puts you at the *front* of the node—to get back to where you were reading you have to type some ⟨SPC⟩s.

`>>` Now type *u* to move back up to '`Help-M`'.

1.6 Some advanced Info commands

The course is almost over, so please stick with it to the end.

If you have been moving around to different nodes and wish to retrace your steps, the *l* command (*l* for *last*) will do that, one node-step at a time. As you move from node to node, Info records the nodes where you have been in a special history list. The *l* command revisits nodes in the history list; each successive *l* command moves one step back through the history.

If you have been following directions, an *l* command now will get you back to '`Help-M`'. Another *l* command would undo the *u* and get you back to '`Help-FOO`'. Another *l* would undo the *m* and get you back to '`Help-M`'.

>> Try typing three *`l`*'s, pausing in between to see what each
`l` does.

Then follow directions again and you will end up back here.

Note the difference between *`l`* and *`p`*: *`l`* moves to where *you* last were, whereas *`p`* always moves to the node which the header says is the '`Previous`' node (from this node, to '`Help-M`').

The '`d`' command gets you instantly to the Directory node. This node, which is the first one you saw when you entered Info, has a menu which leads (directly, or indirectly through other menus), to all the nodes that exist.

>> Try doing a '`d`', then do an *`l`* to return here (yes,
do return).

Sometimes, in Info documentation, you will see a cross reference. Cross references look like this: See [Help-Cross], page 6. That is a real, live cross reference which is named '`Cross`' and points at the node named '`Help-Cross`'.

If you wish to follow a cross reference, you must use the '`f`' command. The '`f`' must be followed by the cross reference name (in this case, '`Cross`'). While you enter the name, you can use the Delete key to edit your input. If you change your mind about following any reference, you can use *`Control-g`* to cancel the command.

Completion is available in the '`f`' command; you can complete among all the cross reference names in the current node by typing a Tab.

>> Type '`f`', followed by '`Cross`', and a ⟨RET⟩.

To get a list of all the cross references in the current node, you can type *`?`* after an '`f`'. The '`f`' continues to await a cross reference name even after printing the list, so if you don't actually want to follow a reference, you should type a *`Control-g`* to cancel the '`f`'.

>> Type "`f?`" to get a list of the cross references in this node. Then
type a *`Control-g`* and see how the '`f`' gives up.

>> Now type *`n`* to see the last node of the course.

The node reached by the cross reference in Info

This is the node reached by the cross reference named '`Cross`'.

While this node is specifically intended to be reached by a cross reference, most cross references lead to nodes that "belong" someplace else far away in the structure of Info. So you cannot expect the footnote to have a '`Next`', '`Previous`' or '`Up`' pointing back to where you came from. In general, the *`l`* (el) command is the only way to get back there.

>> Type *`l`* to return to the node where the cross reference was.

1.7 Quitting Info

To get out of Info, back to what you were doing before, type *`q`* for *Quit*.

This is the end of the course on using Info. There are some other commands that are meant for experienced users; they are useful, and you can find them by looking in the directory node for documentation on Info. Finding them will be a good exercise in using Info in the usual manner.

`>>` Type ‘`d`’ to go to the Info directory node; then type
‘`mInfo`’ and Return, to get to the node about Info and
see what other help is available.

2 Info for Experts

This chapter describes various advanced Info commands, and how to write an Info as distinct from a Texinfo file. (However, in most cases, writing a Texinfo file is better, since you can use it *both* to generate an Info file and to make a printed manual. See section "Overview of Texinfo" in *Texinfo: The GNU Documentation Format*.)

2.1 Advanced Info Commands

`g`, `s`, `1`, – `9`, and `e`

If you know a node's name, you can go there by typing `g`, the name, and ⟨RET⟩. Thus, `gTop`⟨RET⟩ would go to the node called '`Top`' in this file (its directory node). `gExpert`⟨RET⟩ would come back here.

Unlike `m`, `g` does not allow the use of abbreviations.

To go to a node in another file, you can include the filename in the node name by putting it at the front, in parentheses. Thus, `g(dir)Top`⟨RET⟩ would go to the Info Directory node, which is node '`Top`' in the file '`dir`'.

The node name '`*`' specifies the whole file. So you can look at all of the current file by typing `g*`⟨RET⟩ or all of any other file with `g(FILENAME)`⟨RET⟩.

The `s` command allows you to search a whole file for a string. It switches to the next node if and when that is necessary. You type `s` followed by the string to search for, terminated by ⟨RET⟩. To search for the same string again, just `s` followed by ⟨RET⟩ will do. The file's nodes are scanned in the order they are in in the file, which has no necessary relationship to the order that they may be in in the tree structure of menus and '`next`' pointers. But normally the two orders are not very different. In any case, you can always do a `b` to find out what node you have reached, if the header is not visible (this can happen, because `s` puts your cursor at the occurrence of the string, not at the beginning of the node).

`Meta-s` is equivalent to `s`. That is for compatibility with other GNU packages that use `M-s` for a similar kind of search command.

If you grudge the system each character of type-in it requires, you might like to use the commands `1`, `2`, `3`, `4`, ... `9`. They are short for the `m` command together with an argument. `1` goes through the first item in the current node's menu; `2` goes through the second item, etc.

If your display supports multiple fonts, and you are using Emacs' Info mode to read Info files, the '`*`' for the fifth menu item is underlined, and so is the '`*`' for the ninth item; these underlines make it easy to see at a glance which number to use for an item.

On ordinary terminals, you won't have underlining. If you need to actually count items, it is better to use `m` instead, and specify the name.

The Info command `e` changes from Info mode to an ordinary Emacs editing mode, so that you can edit the text of the current node. Type `C-c C-c` to switch back to Info. The `e` command is allowed only if the variable `Info-enable-edit` is non-`nil`.

2.2 Adding a new node to Info

To add a new topic to the list in the Info directory, you must:

1. Create some nodes, in some file, to document that topic.
2. Put that topic in the menu in the directory. See Section 2.3 [Menus], page 11.

Usually, the way to create the nodes is with Texinfo see section "Overview of Texinfo" in *Texinfo: The GNU Documentation Format*); this has the advantage that you can also make a printed manual from them. However, if you want to edit an Info file, here is how.

The new node can live in an existing documentation file, or in a new one. It must have a ⟨^_⟩ character before it (invisible to the user; this node has one but you cannot see it), and it ends with either a ⟨^_⟩, a ⟨^L⟩, or the end of file. Note: If you put in a ⟨^L⟩ to end a new node, be sure that there is a ⟨^_⟩ after it to start the next one, since ⟨^L⟩ cannot *start* a node. Also, a nicer way to make a node boundary be a page boundary as well is to put a ⟨^L⟩ *right after* the ⟨^_⟩.

The ⟨^_⟩ starting a node must be followed by a newline or a ⟨^L⟩ newline, after which comes the node's header line. The header line must give the node's name (by which Info finds it), and state the names of the '`Next`', '`Previous`', and '`Up`' nodes (if there are any). As you can see, this node's '`Up`' node is the node '`Top`', which points at all the documentation for Info. The '`Next`' node is '`Menus`'.

The keywords *Node*, *Previous*, *Up*, and *Next*, may appear in any order, anywhere in the header line, but the recommended order is the one in this sentence. Each keyword must be followed by a colon, spaces and tabs, and then the appropriate name. The name may be terminated with a tab, a comma, or a newline. A space does not end it; node names may contain spaces. The case of letters in the names is insignificant.

A node name has two forms. A node in the current file is named by what appears after the '`Node: `' in that node's first line. For example, this node's name is '`Add`'. A node in another file is named by '`(`*filename*`)`*node-within-file*', as in '`(info)Add`' for this node. If the file name starts with "./", then it is relative to the current directory; otherwise, it is relative starting from the standard Info file directory of your site. The name '`(`*filename*`)Top`' can be abbreviated to just '`(`*filename*`)`'. By convention, the name '`Top`' is used for the "highest" node in any single file—the node whose '`Up`' points out of the file. The Directory node is '`(dir)`'. The '`Top`' node of a document file listed in the Directory should have an '`Up: (dir)`' in it.

The node name `*` is special: it refers to the entire file. Thus, *g*`*` shows you the whole current file. The use of the node `*` is to make it possible to make old-fashioned, unstructured files into nodes of the tree.

The '`Node:`' name, in which a node states its own name, must not contain a filename, since Info when searching for a node does not expect one to be there. The '`Next`', '`Previous`' and '`Up`' names may contain them. In this node, since the '`Up`' node is in the same file, it was not necessary to use one.

Note that the nodes in this file have a file name in the header line. The file names are ignored by Info, but they serve as comments to help identify the node for the user.

2.3 How to Create Menus

Any node in the Info hierarchy may have a *menu*—a list of subnodes. The *m* command searches the current node's menu for the topic which it reads from the terminal.

A menu begins with a line starting with '`* Menu:`'. The rest of the line is a comment. After the starting line, every line that begins with a '`* `' lists a single topic. The name of the topic–the argument that the user must give to the *m* command to select this topic—comes right after the star and space, and is followed by a colon, spaces and tabs, and the name of the node which discusses that topic. The node name, like node names following '`Next`', '`Previous`' and '`Up`', may be terminated with a tab, comma, or newline; it may also be terminated with a period.

If the node name and topic name are the same, then rather than giving the name twice, the abbreviation '`* NAME::`' may be used (and should be used, whenever possible, as it reduces the visual clutter in the menu).

It is considerate to choose the topic names so that they differ from each other very near the beginning—this allows the user to type short abbreviations. In a long menu, it is a good idea to capitalize the beginning of each item name which is the minimum acceptable abbreviation for it (a long menu is more than 5 or so entries).

The nodes listed in a node's menu are called its "subnodes", and it is their "superior". They should each have an '`Up:`' pointing at the superior. It is often useful to arrange all or most of the subnodes in a sequence of '`Next`' and '`Previous`' pointers so that someone who wants to see them all need not keep revisiting the Menu.

The Info Directory is simply the menu of the node '`(dir)Top`'—that is, node '`Top`' in file '`.../info/dir`'. You can put new entries in that menu just like any other menu. The Info Directory is *not* the same as the file directory called '`info`'. It happens that many of Info's files live on that file directory, but they do not have to; and files on that directory are not automatically listed in the Info Directory node.

Also, although the Info node graph is claimed to be a "hierarchy", in fact it can be *any* directed graph. Shared structures and pointer cycles are perfectly possible, and can be used if they are appropriate to the meaning to be expressed. There is no need for all the nodes in a file to form a connected structure. In fact, this file has two connected components. You are in one of them, which is under the node '`Top`'; the other contains the node '`Help`' which the *h* command goes to. In fact, since there is no garbage collector, nothing terrible happens if a substructure is not pointed to, but such a substructure is rather useless since nobody can ever find out that it exists.

2.4 Creating Cross References

A cross reference can be placed anywhere in the text, unlike a menu item which must go at the front of a line. A cross reference looks like a menu item except that it has '`*note`' instead of `*`. It *cannot* be terminated by a '`)`', because '`)`''s are so often part of node names. If you wish to enclose a cross reference in parentheses, terminate it with a period first. Here are two examples of cross references pointers:

```
*Note details: commands.  (See *note 3: Full Proof.)
```

They are just examples. The places they “lead to” do not really exist!

2.5 Tags Tables for Info Files

You can speed up the access to nodes of a large Info file by giving it a tags table. Unlike the tags table for a program, the tags table for an Info file lives inside the file itself and is used automatically whenever Info reads in the file.

To make a tags table, go to a node in the file using Emacs Info mode and type *`M-x Info-tagify`*. Then you must use *`C-x C-s`* to save the file.

Once the Info file has a tags table, you must make certain it is up to date. If, as a result of deletion of text, any node moves back more than a thousand characters in the file from the position recorded in the tags table, Info will no longer be able to find that node. To update the tags table, use the `Info-tagify` command again.

An Info file tags table appears at the end of the file and looks like this:

```
^_

Tag Table:
File: info, Node: Cross-refs^?21419
File: info,  Node: Tags^?22145
^_
End Tag Table
```

Note that it contains one line per node, and this line contains the beginning of the node’s header (ending just after the node name), a Delete character, and the character position in the file of the beginning of the node.

2.6 Checking an Info File

When creating an Info file, it is easy to forget the name of a node when you are making a pointer to it from another node. If you put in the wrong name for a node, this is not detected until someone tries to go through the pointer using Info. Verification of the Info file is an automatic process which checks all pointers to nodes and reports any pointers which are invalid. Every ‘`Next`’, ‘`Previous`’, and ‘`Up`’ is checked, as is every menu item and every cross reference. In addition, any ‘`Next`’ which does not have a ‘`Previous`’ pointing back is reported. Only pointers within the file are checked, because checking pointers to other files would be terribly slow. But those are usually few.

To check an Info file, do *`M-x Info-validate`* while looking at any node of the file with Emacs Info mode.

2.7 Emacs Info-mode Variables

The following variables may modify the behaviour of Info-mode in Emacs; you may wish to set one or several of these variables interactively, or in your '`~/.emacs`' init file. See section "Examining and Setting Variables" in *The GNU Emacs Manual*.

`Info-enable-edit`
: Set to `nil`, disables the '`e`' (`Info-edit`) command. A non-`nil` value enables it. See Section 2.2 [Add], page 10.

`Info-enable-active-nodes`
: When set to a non-`nil` value, allows Info to execute Lisp code associated with nodes. The Lisp code is executed when the node is selected.

`Info-directory-list`
: The list of directories to search for Info files. Each element is a string (directory name) or `nil` (try default directory).

`Info-directory`
: The standard directory for Info documentation files. Only used when the function `Info-directory` is called.

3 Creating an Info File from a Makeinfo file

`makeinfo` is a utility that converts a Texinfo file into an Info file; `texinfo-format-region` and `texinfo-format-buffer` are GNU Emacs functions that do the same.

See section "Creating an Info File" in *the Texinfo Manual*, to learn how to create an Info file from a Texinfo file.

See section "Overview of Texinfo" in *Texinfo: The GNU Documentation Format*, to learn how to write a Texinfo file.

4 Using the Stand-alone Info Reader

What is Info?

This text documents the use of the GNU Info program, version 2.10.

Info is a program which is used to view info files on an ASCII terminal. *info files* are the result of processing texinfo files with the program `makeinfo` or with the Emacs command `M-x texinfo-format-buffer`. Finally, *texinfo* is a documentation language which allows a printed manual and online documentation (an info file) to be produced from a single source file.

4.1 Command Line Options

GNU Info accepts several options to control the initial node being viewed, and to specify which directories to search for info files. Here is a template showing an invocation of GNU Info from the shell:

`info [--`*option-name option-value*`]` *menu-item...*

The following *option-names* are available when invoking Info from the shell:

`--directory` *directory-path*
`-d` *directory-path*

Adds *directory-path* to the list of directory paths searched when Info needs to find a file. You may issue `--directory` multiple times; once for each directory which contains info files. Alternatively, you may specify a value for the environment variable `INFOPATH`; if `--directory` is not given, the value of `INFOPATH` is used. The value of `INFOPATH` is a colon separated list of directory names. If you do not supply `INFOPATH` or `--directory-path` a default path is used.

`--file` *filename*
`-f` *filename*

Specifies a particular info file to visit. Instead of visiting the file `dir`, Info will start with `(`*filename*`)Top` as the first file and node.

`--node` *nodename*
`-n` *nodename*

Specifies a particular node to visit in the initial file loaded. This is especially useful in conjunction with `--file`[1]. You may specify `--node` multiple times; for an interactive Info, each *nodename* is visited in its own window, for a non-interactive Info (such as when `--output` is given) each *nodename* is processed sequentially.

---

[1] Of course, you can specify both the file and node in a `--node` command; but don't forget to escape the open and close parentheses from the shell as in: `info --node '(emacs)Buffers'`

`--output` *filename*
`-o` *filename*

Specify *filename* as the name of a file to output to. Each node that Info visits will be output to *filename* instead of interactively viewed. A value of `-` for *filename* specifies the standard output.

`--subnodes`

This option only has meaning when given in conjunction with `--output`. It means to recursively output the nodes appearing in the menus of each node being output. Menu items which resolve to external info files are not output, and neither are menu items which are members of an index. Each node is only output once.

`--help`
`-h`

Produces a relatively brief description of the available Info options.

`--version`

Prints the version information of Info and exits.

menu-item

Remaining arguments to Info are treated as the names of menu items. The first argument would be a menu item in the initial node visited, while the second argument would be a menu item in the first argument's node. You can easily move to the node of your choice by specifying the menu names which describe the path to that node. For example,

```
info emacs buffers
```

first selects the menu item '`Emacs`' in the node '`(dir)Top`', and then selects the menu item '`Buffers`' in the node '`(emacs)Top`'.

4.2 Moving the Cursor

Many people find that reading screens of text page by page is made easier when one is able to indicate particular pieces of text with some kind of pointing device. Since this is the case, GNU Info (both the Emacs and standalone versions) have several commands which allow you to move the cursor about the screen. The notation used in this manual to describe keystrokes is identical to the notation used within the Emacs manual, and the GNU Readline manual. See section "Character Conventions" in *the GNU Emacs Manual*, if you are unfamiliar with the notation.

The following table lists the basic cursor movement commands in Info. Each entry consists of the key sequence you should type to execute the cursor movement, the `M-x`[2] command name (displayed in parentheses), and a short description of what the command does. All of the cursor motion commands can take an *numeric* argument (see Section 4.9 [Miscellaneous Commands], page 28), to find out how to supply them. With a numeric argument, the motion commands are simply executed that many times; for example, a

---

[2] `M-x` is also a command; it invokes `execute-extended-command`. See section "Executing an extended command" in *the GNU Emacs Manual*, for more detailed information.

numeric argument of 4 given to `next-line` causes the cursor to move down 4 lines. With a negative numeric argument, the motion is reversed; an argument of -4 given to the `next-line` command would cause the cursor to move *up* 4 lines.

`C-n` (`next-line`)
Moves the cursor down to the next line.

`C-p` (`prev-line`)
Move the cursor up to the previous line.

`C-a` (`beginning-of-line`)
Move the cursor to the start of the current line.

`C-e` (`end-of-line`)
Moves the cursor to the end of the current line.

`C-f` (`forward-char`)
Move the cursor forward a character.

`C-b` (`backward-char`)
Move the cursor backward a character.

`M-f` (`forward-word`)
Moves the cursor forward a word.

`M-b` (`backward-word`)
Moves the cursor backward a word.

`M-<` (`beginning-of-node`)
`b` Moves the cursor to the start of the current node.

`M->` (`end-of-node`)
Moves the cursor to the end of the current node.

`M-r` (`move-to-window-line`)
Moves the cursor to a specific line of the window. Without a numeric argument, `M-r` moves the cursor to the start of the line in the center of the window. With a numeric argument of *n*, `M-r` moves the cursor to the start of the *n*th line in the window.

4.3 Moving Text Within a Window

Sometimes you are looking at a screenful of text, and only part of the current paragraph you are reading is visible on the screen. The commands detailed in this section are used to shift which part of the current node is visible on the screen.

`SPC` (`scroll-forward`)
`C-v` Shift the text in this window up. That is, show more of the node which is currently below the bottom of the window. With a numeric argument, show that many more lines at the bottom of the window; a numeric argument of 4 would shift all of the text in the window up 4 lines (discarding the top 4 lines), and show you four new lines at the bottom of the window. Without a numeric

argument, ⟨SPC⟩ takes the bottom two lines of the window and places them at the top of the window, redisplaying almost a completely new screenful of lines.

`DEL` (`scroll-backward`)
`M-v` Shift the text in this window down. The inverse of `scroll-forward`.

The `scroll-forward` and `scroll-backward` commands can also move forward and backward through the node structure of the file. If you press ⟨SPC⟩ while viewing the end of a node, or ⟨DEL⟩ while viewing the beginning of a node, what happens is controlled by the variable `scroll-behaviour`. See Section 4.10 [Variables], page 30, for more information.

`C-l` (`redraw-display`)
Redraw the display from scratch, or shift the line containing the cursor to a specified location. With no numeric argument, '`C-l`' clears the screen, and then redraws its entire contents. Given a numeric argument of *n*, the line containing the cursor is shifted so that it is on the *n*th line of the window.

`C-x w` (`toggle-wrap`)
Toggles the state of line wrapping in the current window. Normally, lines which are longer than the screen width *wrap*, i.e., they are continued on the next line. Lines which wrap have a '`\`' appearing in the rightmost column of the screen. You can cause such lines to be terminated at the rightmost column by changing the state of line wrapping in the window with `C-x w`. When a line which needs more space than one screen width to display is displayed, a '`$`' appears in the rightmost column of the screen, and the remainder of the line is invisible.

4.4 Selecting a New Node

This section details the numerous Info commands which select a new node to view in the current window.

The most basic node commands are '`n`', '`p`', '`u`', and '`l`'.

When you are viewing a node, the top line of the node contains some Info *pointers* which describe where the next, previous, and up nodes are. Info uses this line to move about the node structure of the file when you use the following commands:

`n` (`next-node`)
Selects the 'Next' node.

`p` (`prev-node`)
Selects the 'Prev' node.

`u` (`up-node`)
Selects the 'Up' node.

You can easily select a node that you have already viewed in this window by using the '`l`' command – this name stands for "last", and actually moves through the list of already visited nodes for this window. '`l`' with a negative numeric argument moves forward through the history of nodes for this window, so you can quickly step between two adjacent (in viewing history) nodes.

`l` (`history-node`)
: Selects the most recently selected node in this window.

Two additional commands make it easy to select the most commonly selected nodes; they are '`t`' and '`d`'.

`t` (`top-node`)
: Selects the node '`Top`' in the current info file.

`d` (`dir-node`)
: Selects the directory node (i.e., the node '`(dir)`').

Here are some other commands which immediately result in the selection of a different node in the current window:

`<` (`first-node`)
: Selects the first node which appears in this file. This node is most often '`Top`', but it doesn't have to be.

`>` (`last-node`)
: Selects the last node which appears in this file.

`]` (`global-next-node`)
: Moves forward or down through node structure. If the node that you are currently viewing has a '`Next`' pointer, that node is selected. Otherwise, if this node has a menu, the first menu item is selected. If there is no '`Next`' and no menu, the same process is tried with the '`Up`' node of this node.

`[` (`global-prev-node`)
: Moves backward or up through node structure. If the node that you are currently viewing has a '`Prev`' pointer, that node is selected. Otherwise, if the node has an '`Up`' pointer, that node is selected, and if it has a menu, the last item in the menu is selected.

You can get the same behaviour as `global-next-node` and `global-prev-node` while simply scrolling through the file with ⟨SPC⟩ and ⟨DEL⟩; See Section 4.10 [Variables], page 30, for more information.

`g` (`goto-node`)
: Reads the name of a node and selects it. No completion is done while reading the node name, since the desired node may reside in a separate file. The node must be typed exactly as it appears in the info file. A file name may be included as with any node specification, for example

```
g(emacs)Buffers
```

finds the node '`Buffers`' in the info file '`emacs`'.

`C-x k` (`kill-node`)
: Kills a node. The node name is prompted for in the echo area, with a default of the current node. *Killing* a node means that Info tries hard to forget about it, removing it from the list of history nodes kept for the window where that node is found. Another node is selected in the window which contained the killed node.

`C-x C-f` (`view-file`)

Reads the name of a file and selects the entire file. The command

`C-x C-f` *filename*

is equivalent to typing

`g(`*filename*`)*`

`C-x C-b` (`list-visited-nodes`)

Makes a window containing a menu of all of the currently visited nodes. This window becomes the selected window, and you may use the standard Info commands within it.

`C-x b` (`select-visited-node`)

Selects a node which has been previously visited in a visible window. This is similar to '`C-x C-b`' followed by '`m`', but no window is created.

4.5 Searching an Info File

GNU Info allows you to search for a sequence of characters throughout an entire info file, search through the indices of an info file, or find areas within an info file which discuss a particular topic.

`s` (`search`)

Reads a string in the echo area and searches for it.

`C-s` (`isearch-forward`)

Interactively searches forward through the info file for a string as you type it.

`C-r` (`isearch-backward`)

Interactively searches backward through the info file for a string as you type it.

`i` (`index-search`)

Looks up a string in the indices for this info file, and selects a node where the found index entry points to.

`,` (`next-index-match`)

Moves to the node containing the next matching index item from the last '`i`' command.

The most basic searching command is '`s`' (`search`). The '`s`' command prompts you for a string in the echo area, and then searches the remainder of the info file for an occurrence of that string. If the string is found, the node containing it is selected, and the cursor is left positioned at the start of the found string. Subsequent '`s`' commands show you the default search string within '`[`' and '`]`'; pressing ⟨RET⟩ instead of typing a new string will use the default search string.

Incremental searching is similar to basic searching, but the string is looked up while you are typing it, instead of waiting until the entire search string has been specified.

4.6 Selecting Cross References

We have already discussed the '`Next`', '`Prev`', and '`Up`' pointers which appear at the top of a node. In addition to these pointers, a node may contain other pointers which refer you to a different node, perhaps in another info file. Such pointers are called *cross references*, or *xrefs* for short.

4.6.1 Parts of an Xref

Cross references have two major parts: the first part is called the *label*; it is the name that you can use to refer to the cross reference, and the second is the *target*; it is the full name of the node that the cross reference points to.

The target is separated from the label by a colon '`:`'; first the label appears, and then the target. For example, in the sample menu cross reference below, the single colon separates the label from the target.

```
* Foo Label: Foo Target.  More information about Foo.
```

Note the '`.`' which ends the name of the target. The '`.`' is not part of the target; it serves only to let Info know where the target name ends.

A shorthand way of specifying references allows two adjacent colons to stand for a target name which is the same as the label name:

```
* Foo Commands::  Commands pertaining to Foo.
```

In the above example, the name of the target is the same as the name of the label, in this case `Foo Commands`.

You will normally see two types of cross references while viewing nodes: *menu* references, and *note* references. Menu references appear within a node's menu; they begin with a '`*`' at the beginning of a line, and continue with a label, a target, and a comment which describes what the contents of the node pointed to contains.

Note references appear within the body of the node text; they begin with `*Note`, and continue with a label and a target.

Like '`Next`', '`Prev`' and '`Up`' pointers, cross references can point to any valid node. They are used to refer you to a place where more detailed information can be found on a particular subject. Here is a cross reference which points to a node within the Texinfo documentation: See section "Writing an Xref" in *the Texinfo Manual*, for more information on creating your own texinfo cross references.

4.6.2 Selecting Xrefs

The following table lists the Info commands which operate on menu items.

`1` (`menu-digit`)
`2` ... `9` Within an Info window, pressing a single digit, (such as '`1`'), selects that menu item, and places its node in the current window. For convenience, there is one exception; pressing '`0`' selects the *last* item in the node's menu.

`0` (`last-menu-item`)
: Select the last item in the current node's menu.

`m` (`menu-item`)
: Reads the name of a menu item in the echo area and selects its node. Completion is available while reading the menu label.

`M-x find-menu`
: Moves the cursor to the start of this node's menu.

This table lists the Info commands which operate on note cross references.

`f` (`xref-item`)
`r`
: Reads the name of a note cross reference in the echo area and selects its node. Completion is available while reading the cross reference label.

Finally, the next few commands operate on menu or note references alike:

`TAB` (`move-to-next-xref`)
: Moves the cursor to the start of the next nearest menu item or note reference in this node. You can then use ⟨RET⟩ (`select-reference-this-line` to select the menu or note reference.

`M-TAB` (`move-to-prev-xref`)
: Moves the cursor the start of the nearest previous menu item or note reference in this node.

`RET` (`select-reference-this-line`)
: Selects the menu item or note reference appearing on this line.

4.7 Manipulating Multiple Windows

A *window* is a place to show the text of a node. Windows have a view area where the text of the node is displayed, and an associated *mode line*, which briefly describes the node being viewed.

GNU Info supports multiple windows appearing in a single screen; each window is separated from the next by its modeline. At any time, there is only one *active* window, that is, the window in which the cursor appears. There are commands available for creating windows, changing the size of windows, selecting which window is active, and for deleting windows.

4.7.1 The Mode Line

A *mode line* is a line of inverse video which appears at the bottom of an info window. It describes the contents of the window just above it; this information includes the name of the file and node appearing in that window, the number of screen lines it takes to display the node, and the percentage of text that is above the top of the window. It can also tell you if the indirect tags table for this info file needs to be updated, and whether or not the info file was compressed when stored on disk.

Here is a sample mode line for a window containing an uncompressed file named ‘`dir`’, showing the node ‘`Top`’.

```
-----Info: (dir)Top, 40 lines --Top---------------------------------------
            ^^   ^   ^^^        ^^
          (file)Node #lines    where
```

When a node comes from a file which is compressed on disk, this is indicated in the mode line with two small ‘`z`’’s. In addition, if the info file containing the node has been split into subfiles, the name of the subfile containing the node appears in the modeline as well:

```
--zz-Info: (emacs)Top, 291 lines --Top-- Subfile: emacs-1.Z---------------
```

When Info makes a node internally, such that there is no corresponding info file on disk, the name of the node is surrounded by asterisks (‘`*`’). The name itself tells you what the contents of the window are; the sample mode line below shows an internally constructed node showing possible completions:

```
-----Info: *Completions*, 7 lines --All-----------------------------------
```

4.7.2 Window Commands

It can be convenient to view more than one node at a time. To allow this, Info can display more than one *window*. Each window has its own mode line (see Section 4.7.1 [The Mode Line], page 24) and history of nodes viewed in that window (see Section 4.4 [`history-node`], page 20).

`C-x o` (`next-window`)

Selects the next window on the screen. Note that the echo area can only be selected if it is already in use, and you have left it temporarily. Normally, ‘`C-x o`’ simply moves the cursor into the next window on the screen, or if you are already within the last window, into the first window on the screen. Given a numeric argument, ‘`C-x o`’ moves over that many windows. A negative argument causes ‘`C-x o`’ to select the previous window on the screen.

`M-x prev-window`

Selects the previous window on the screen. This is identical to ‘`C-x o`’ with a negative argument.

`C-x 2` (`split-window`)

Splits the current window into two windows, both showing the same node. Each window is one half the size of the original window, and the cursor remains in the original window. The variable `automatic-tiling` can cause all of the windows on the screen to be resized for you automatically, please see Section 4.10 [automatic-tiling], page 30 for more information.

`C-x 0` (`delete-window`)

Deletes the current window from the screen. If you have made too many windows and your screen appears cluttered, this is the way to get rid of some of them.

`C-x 1` (`keep-one-window`)
: Deletes all of the windows excepting the current one.

`ESC C-v` (`scroll-other-window`)
: Scrolls the other window, in the same fashion that ‘`C-v`’ might scroll the current window. Given a negative argument, the `"other"` window is scrolled backward.

`C-x ^` (`grow-window`)
: Grows (or shrinks) the current window. Given a numeric argument, grows the current window that many lines; with a negative numeric argument, the window is shrunk instead.

`C-x t` (`tile-windows`)
: Divides the available screen space among all of the visible windows. Each window is given an equal portion of the screen in which to display its contents. The variable `automatic-tiling` can cause `tile-windows` to be called when a window is created or deleted. See Section 4.10 [`automatic-tiling`], page 30.

4.7.3 The Echo Area

The *echo area* is a one line window which appears at the bottom of the screen. It is used to display informative or error messages, and to read lines of input from you when that is necessary. Almost all of the commands available in the echo area are identical to their Emacs counterparts, so please refer to that documentation for greater depth of discussion on the concepts of editing a line of text. The following table briefly lists the commands that are available while input is being read in the echo area:

`C-f` (`echo-area-forward`)
: Moves forward a character.

`C-b` (`echo-area-backward`)
: Moves backward a character.

`C-a` (`echo-area-beg-of-line`)
: Moves to the start of the input line.

`C-e` (`echo-area-end-of-line`)
: Moves to the end of the input line.

`M-f` (`echo-area-forward-word`)
: Moves forward a word.

`M-b` (`echo-area-backward-word`)
: Moves backward a word.

`C-d` (`echo-area-delete`)
: Deletes the character under the cursor.

`DEL` (`echo-area-rubout`)
: Deletes the character behind the cursor.

`C-g` (`echo-area-abort`)
: Cancels or quits the current operation. If completion is being read, '`C-g`' discards the text of the input line which does not match any completion. If the input line is empty, '`C-g`' aborts the calling function.

`RET` (`echo-area-newline`)
: Accepts (or forces completion of) the current input line.

`C-q` (`echo-area-quoted-insert`)
: Inserts the next character verbatim. This is how you can insert control characters into a search string, for example.

printing character (`echo-area-insert`)
: Inserts the character.

`M-TAB` (`echo-area-tab-insert`)
: Inserts a TAB character.

`C-t` (`echo-area-transpose-chars`)
: Transposes the characters at the cursor.

The next group of commands deal with *killing*, and *yanking* text. For an in depth discussion of killing and yanking, see section "Killing and Deleting" in *the GNU Emacs Manual*

`M-d` (`echo-area-kill-word`)
: Kills the word following the cursor.

`M-DEL` (`echo-area-backward-kill-word`)
: Kills the word preceding the cursor.

`C-k` (`echo-area-kill-line`)
: Kills the text from the cursor to the end of the line.

`C-x DEL` (`echo-area-backward-kill-line`)
: Kills the text from the cursor to the beginning of the line.

`C-y` (`echo-area-yank`)
: Yanks back the contents of the last kill.

`M-y` (`echo-area-yank-pop`)
: Yanks back a previous kill, removing the last yanked text first.

Sometimes when reading input in the echo area, the command that needed input will only accept one of a list of several choices. The choices represent the *possible completions*, and you must respond with one of them. Since there are a limited number of responses you can make, Info allows you to abbreviate what you type, only typing as much of the response as is necessary to uniquely identify it. In addition, you can request Info to fill in as much of the response as is possible; this is called *completion*.

The following commands are available when completing in the echo area:

`TAB` (`echo-area-complete`)
`SPC`
: Inserts as much of a completion as is possible.

`?` (`echo-area-possible-completions`)
: Displays a window containing a list of the possible completions of what you have typed so far. For example, if the available choices are:

```
bar
foliate
food
forget
```

and you have typed an '`f`', followed by '`?`', the possible completions would contain:

```
foliate
food
forget
```

i.e., all of the choices which begin with '`f`'. Pressing ⟨SPC⟩ or ⟨TAB⟩ would result in '`fo`' appearing in the echo area, since all of the choices which begin with '`f`' continue with '`o`'. Now, typing '`l`' followed by '`TAB`' results in '`foliate`' appearing in the echo area, since that is the only choice which begins with '`fol`'.

`ESC C-v` (`echo-area-scroll-completions-window`)
: Scrolls the completions window, if that is visible, or the "other" window if not.

4.8 Printing Out Nodes

You may wish to print out the contents of a node as a quick reference document for later use. Info provides you with a command for doing this. In general, we recommend that you use TeX to format the document and print sections of it, by running `tex` on the texinfo source file.

`M-x print-node`
: Pipes the contents of the current node through the command in the environment variable `INFO_PRINT_COMMAND`. If the variable doesn't exist, the node is simply piped to `lpr`.

4.9 Miscellaneous Commands

GNU Info contains several commands which self-document GNU Info:

`M-x describe-command`
: Reads the name of an Info command in the echo area and then displays a brief description of what that command does.

`M-x describe-key`
: Reads a key sequence in the echo area, and then displays the name and documentation of the Info command that the key sequence invokes.

`M-x describe-variable`
: Reads the name of a variable in the echo area and then displays a brief description of what the variable affects.

`M-x where-is`
: Reads the name of an Info command in the echo area, and then displays a key sequence which can be typed in order to invoke that command.

`C-h` (`get-help-window`)
`?`
: Creates (or moves into) the window displaying `*Help*`, and places a node containing a quick reference card into it. This window displays the most concise information about GNU Info available.

`h` (`get-info-help-node`)
: Tries hard to visit the node `(info)Help`. The info file '`info.texi`' distributed with GNU Info contains this node. Of course, the file must first be processed with `makeinfo`, and then placed into the location of your info directory.

Here are the commands for creating a numeric argument:

`C-u` (`universal-argument`)
: Starts (or multiplies by 4) the current numeric argument. '`C-u`' is a good way to give a small numeric argument to cursor movement or scrolling commands; '`C-u C-v`' scrolls the screen 4 lines, while '`C-u C-u C-n`' moves the cursor down 16 lines.

`M-1` (`add-digit-to-numeric-arg`)
`M-2` ... `M-9`
: Adds the digit value of the invoking key to the current numeric argument. Once Info is reading a numeric argument, you may just type the digits of the argument, without the Meta prefix. For example, you might give '`C-l`' a numeric argument of 32 by typing:

    C-u 3 2 C-l

  or

    M-3 2 C-l

'`C-g`' is used to abort the reading of a multi-character key sequence, to cancel lengthy operations (such as multi-file searches) and to cancel reading input in the echo area.

`C-g` (`abort-key`)
: Cancels current operation.

The '`q`' command of Info simply quits running Info.

`q` (`quit`)
: Exits GNU Info.

If the operating system tells GNU Info that the screen is 60 lines tall, and it is actually only 40 lines tall, here is a way to tell Info that the operating system is correct.

`M-x set-screen-height`
: Reads a height value in the echo area and sets the height of the displayed screen to that value.

Finally, Info provides a convenient way to display footnotes which might be associated with the current node that you are viewing:

`ESC C-f` (`show-footnotes`)
: Shows the footnotes (if any) associated with the current node in another window. You can have Info automatically display the footnotes associated with a node when the node is selected by setting the variable `automatic-footnotes`. See Section 4.10 [`automatic-footnotes`], page 30.

4.10 Manipulating Variables

GNU Info contains several *variables* whose values are looked at by various Info commands. You can change the values of these variables, and thus change the behaviour of Info to more closely match your environment and info file reading manner.

`M-x set-variable`
: Reads the name of a variable, and the value for it, in the echo area and then sets the variable to that value. Completion is available when reading the variable name; often, completion is available when reading the value to give to the variable, but that depends on the variable itself. If a variable does *not* supply multiple choices to complete over, it expects a numeric value.

`M-x describe-variable`
: Reads the name of a variable in the echo area and then displays a brief description of what the variable affects.

Here is a list of the variables that you can set in Info.

`automatic-footnotes`
: When set to `On`, footnotes appear and disappear automatically. This variable is `On` by default. When a node is selected, a window containing the footnotes which appear in that node is created, and the footnotes are displayed within the new window. The window that Info creates to contain the footnotes is called '`*Footnotes*`'. If a node is selected which contains no footnotes, and a '`*Footnotes*`' window is on the screen, the '`*Footnotes*`' window is deleted. Footnote windows created in this fashion are not automatically tiled so that they can use as little of the display as is possible.

`automatic-tiling`
: When set to `On`, creating or deleting a window resizes other windows. This variable is `Off` by default. Normally, typing '`C-x 2`' divides the current window into two equal parts. When `automatic-tiling` is set to `On`, all of the windows are resized automatically, keeping an equal number of lines visible in each window. There are exceptions to the automatic tiling; specifically, the windows '`*Completions*`' and '`*Footnotes*`' are *not* resized through automatic tiling; they remain their original size.

`visible-bell`
: When set to `On`, GNU Info attempts to flash the screen instead of ringing the bell. This variable is `Off` by default. Of course, Info can only flash the screen if

the terminal allows it; in the case that the terminal does not allow it, the setting of this variable has no effect. However, you can make Info perform quietly by setting the `errors-ring-bell` variable to `Off`.

`errors-ring-bell`
: When set to `On`, errors cause the bell to ring. The default setting of this variable is `On`.

`gc-compressed-files`
: When set to `On`, Info garbage collects files which had to be uncompressed. The default value of this variable is `Off`. Whenever a node is visited in Info, the info file containing that node is read into core, and Info reads information about the tags and nodes contained in that file. Once the tags information is read by Info, it is never forgotten. However, the actual text of the nodes does not need to remain in core unless a particular info window needs it. For non-compressed files, the text of the nodes does not remain in core when it is no longer in use. But de-compressing a file can be a time consuming operation, and so Info tries hard not to do it twice. `gc-compressed-files` tells Info it is okay to garbage collect the text of the nodes of a file which was compressed on disk.

`show-index-match`
: When set to `On`, the portion of the matched search string is highlighted in the message which explains where the matched search string was found. The default value of this variable is `On`. When Info displays the location where an index match was found, (see Section 4.5 [`next-index-match`], page 22), the portion of the string that you had typed is highlighted by displaying it in the inverse case from its surrounding characters.

`scroll-behaviour`
: Controls what happens when forward scrolling is requested at the end of a node, or when backward scrolling is requested at the beginning of a node. The default value for this variable is `Continuous`. There are three possible values for this variable:

  `Continuous`
  : Tries to get the first item in this node's menu, or failing that, the '`Next`' node, or failing that, the '`Next`' of the '`Up`'. This behaviour is identical to using the '`]`' (`global-next-node`) and '`[`' (`global-prev-node`) commands.

  `Next Only`
  : Only tries to get the '`Next`' node.

  `Page Only`
  : Simply gives up, changing nothing. If `scroll-behaviour` is `Page Only`, no scrolling command can change the node that is being viewed.

`scroll-step`
: The number of lines to scroll when the cursor moves out of the window. Scrolling happens automatically if the cursor has moved out of the visible portion of the node text when it is time to display. Usually the scrolling is done so as to put the cursor on the center line of the current window. However, if the variable

`scroll-step` has a nonzero value, Info attempts to scroll the node text by that many lines; if that is enough to bring the cursor back into the window, that is what is done. The default value of this variable is 0, thus placing the cursor (and the text it is attached to) in the center of the window. Setting this variable to 1 causes a kind of "smooth scrolling" which some people prefer.

`ISO-Latin`

When set to `On`, Info accepts and displays ISO Latin-1 characters. By default, Info assumes an ASCII character set. `ISO-Latin` tells Info that it is running in an environment where the European standard character set is in use, and allows you to input such characters to Info, as well as display them.