GNU nano

a small and friendly text editor
version 8.0

Chris Allegretta

This manual documents the GNU `nano` text editor.

The contents of this manual are part of the GNU `nano` distribution.

You may contact the original author by e-mail: `chrisa@asty.org`
Or contact the current maintainer: `bensberg@coevern.nl`

For suggesting improvements: `nano-devel@gnu.org`

1 Introduction

GNU `nano` is a small and friendly text editor. Besides basic text editing, `nano` offers features like undo/redo, syntax coloring, interactive search-and-replace, auto-indentation, line numbers, word completion, file locking, backup files, and internationalization support.

The original goal for `nano` was to be a complete bug-for-bug emulation of Pico. But currently the goal is to be as compatible as is reasonable while offering a superset of Pico's functionality. See Chapter 9 [Pico Compatibility], page 35, for more details on how `nano` and Pico differ.

Since version 4.0, `nano` no longer hard-wraps overlong lines by default. It also by default uses linewise scrolling, and by default includes the line below the title bar in the editing area. In case you want the old, Pico behavior back, you can use the following options: `--breaklonglines`, `--jumpyscrolling`, and `--emptyline` (or `-bje`).

Since version 8.0, `^F` starts a forward search, `^B` starts a backward search, `M-F` searches the next occurrence forward, and `M-B` searches the next occurrence backward. If you want those keystrokes to do what they did before version 8.0, see the rebindings in the sample nanorc file.

Please report bugs via `https://savannah.gnu.org/bugs/?group=nano`.

Questions about using nano you can ask at `help-nano@gnu.org`.

For background information see `https://nano-editor.org/`.

2 Invoking

The usual way to invoke `nano` is:

```
nano [FILE]
```

But it is also possible to specify one or more options (see Chapter 6 [Command-line Options], page 8), and to edit several files in a row.

The cursor can be put on a specific line of a file by adding the line number with a plus sign before the filename, and even in a specific column by adding it with a comma. Negative numbers count from the end of the file or line. The line and column numbers may also be specified by gluing them with colons after the filename. (When a filename contains a colon followed by digits, escape the colon by preceding it with a triple backslash.)

The cursor can be put on the first or last occurrence of a specific string by specifying that string after `+/` or `+?` before the filename. The string can be made case sensitive and/or caused to be interpreted as a regular expression by inserting a `c` and/or `r` after the plus sign. These search modes can be explicitly disabled by using the uppercase variant of those letters: `C` and/or `R`. When the string contains spaces, it needs to be enclosed in quotes. A more complete command synopsis thus is:

```
nano [OPTION]... [[+LINE[,COLUMN]|+[crCR]{/|?}STRING] FILE]...
```

Normally, however, you set your preferred options in a `nanorc` file (see Chapter 8 [Nanorc Files], page 17). And when using `set positionlog` (making `nano` remember the cursor position when you close a file), you will rarely need to specify a line number.

As a special case: when instead of a filename a dash is given, `nano` will read data from standard input. This means you can pipe the output of a command straight into a buffer, and then edit it.

3 Editor Basics

3.1 Screen Layout

The default screen of `nano` consists of four areas. From top to bottom these are: the title bar, the edit window, the status bar, and two help lines.

The title bar consists of three sections: left, center and right. The section on the left displays the version of `nano` being used. The center section displays the current filename, or "New Buffer" if the file has not yet been named. The section on the right displays "Modified" if the file has been modified since it was last saved or opened.

The status bar is the third line from the bottom of the screen. It shows important and informational messages. Any error messages that occur from using the editor will appear on the status bar. Any questions that are asked of the user will be asked on the status bar, and any user input (search strings, filenames, etc.) will be input on the status bar.

The two help lines at the bottom of the screen show some of the most essential functions of the editor.

3.2 Entering Text

`nano` is a "modeless" editor. This means that all keystrokes, with the exception of Control and Meta sequences, enter text into the file being edited.

Characters not present on the keyboard can be entered in two ways:

- For characters with a single-byte code, pressing the Esc key twice and then typing a three-digit decimal number (from *000* to *255*) will make `nano` behave as if you typed the key with that value.
- For any possible character, pressing *M-V* (Alt+V) and then typing a series of hexadecimal digits (at most six, or concluded with *Enter* or *Space*) will enter the corresponding Unicode character into the buffer.

For example, typing *Esc Esc 2 3 4* will enter the character "ê" — useful when writing about a French party. Typing *M-V 0 0 2 2 c 4* will enter the symbol "◇", a little diamond.

Typing *M-V* followed by anything other than a hexadecimal digit will enter this keystroke verbatim into the buffer, allowing the user to insert literal control codes (except `^J`) or escape sequences.

3.3 Commands

Commands are given by using the Control key (Ctrl, shown as `^`) or the Meta key (Alt or Cmd, shown as *M-*).

- A control-key sequence is entered by holding down the Ctrl key and pressing the desired key.

- A meta-key sequence is entered by holding down the Meta key (normally the Alt key) and pressing the desired key.

If for some reason on your system the combinations with Ctrl or Alt do not work, you can generate them by using the Esc key. A control-key sequence is generated by pressing the Esc key twice and then pressing the desired key, and a meta-key sequence by pressing the Esc key once and then pressing the desired key.

3.4 The Cutbuffer

Text can be cut from a file a whole line at a time with *^K*. The cut line is stored in the cutbuffer. Consecutive strokes of *^K* will add each cut line to this buffer, but a *^K* after any other keystroke will overwrite the entire cutbuffer.

The contents of the cutbuffer can be pasted at the current cursor position with *^U*.

A line of text can be copied into the cutbuffer (without cutting it) with *M-6*.

3.5 The Mark

Text can be selected by first 'setting the Mark' with *^6* or *M-A* and then moving the cursor to the other end of the portion to be selected. The selected portion of text will be highlighted. This selection can now be cut or copied in its entirety with a single *^K* or *M-6*. Or the selection can be used to limit the scope of a search-and-replace (*^*) or spell-checking session (*^T^T*).

On some terminals, text can be selected also by holding down *Shift* while using the cursor keys. Holding down the *Ctrl* or *Alt* key too will increase the stride. Such a selection is cancelled upon any cursor movement where *Shift* isn't held.

Cutting or copying selected text toggles off the mark automatically. If needed, it can be toggled off manually with another *^6* or *M-A*.

3.6 Search and Replace

With the Search command (*^F* or *^W*) one can search the current buffer for the occurrence of any string. The default search mode is forward, case-insensitive, and for literal strings. But one can search backwards by toggling *M-B*, search case sensitively with *M-C*, and interpret regular expressions in the search string with *M-R*.

With the Replacement command (*M-R* or *^*) one can replace a given string (or regular expression) with another string. When a regular expression contains fragments between parentheses, the replacement string can refer back to these fragments via `\1` to `\9`.

For each occurrence of the search string you will be asked whether to replace it. You can choose Yes (replace it), or No (skip this one), or All (replace all remaining occurrences without asking any more), or Cancel (stop with replacing, but replacements that have already been made will not be undone).

If before a replacing session starts a region is marked, then only occurrences of the search string within the marked region will be replaced.

A regular expression always covers just one line — it cannot span multiple lines. And neither a search string nor a replacement string can contain a newline (LF).

3.7 Using the Mouse

When mouse support has been configured and enabled, a single mouse click places the cursor at the indicated position. Clicking a second time in the same position toggles the mark. Clicking in the two help lines executes the selected shortcut. To be able to select text with the left button, or paste text with the middle button, hold down the Shift key during those actions.

The mouse will work in the X Window System, and on the console when gpm is running.

3.8 Anchors

With *M-Ins* you can place an anchor (a kind of temporary bookmark) at the current line. With *M-PgUp* and *M-PgDn* you can jump to an anchor in the backward/forward direction. This jumping wraps around at the top and bottom.

When a line with an anchor is removed, the line where the cursor ends up inherits the anchor. After performing an operation on the entire buffer (like formatting it, piping it through a command, or doing an external spell check on it), any anchors that were present are gone. And when you close the buffer, all its anchors simply disappear; they are not saved.

Anchors are visualized in the margin when line numbers are activated.

3.9 Limitations

The recording and playback of keyboard macros works correctly only on a terminal emulator, not on a Linux console (VT), because the latter does not by default distinguish modified from unmodified arrow keys.

4 The Help Viewer

The built-in help in `nano` is available by pressing `^G`. It is fairly self-explanatory. It documents the various parts of the editor and the available keystrokes. Navigation is via the `^Y` (Page Up) and `^V` (Page Down) keys. `^X` exits from the help viewer.

5 The File Browser

When in the Read-File (^R) or Write-Out menu (^O), pressing ^T will invoke the file browser. Here, one can navigate directories in a graphical manner in order to find the desired file.

Basic movement in the file browser is accomplished with the arrow and other cursor-movement keys. More targeted movement is accomplished by searching, via ^W or w, or by changing directory, via ^_ or g. The behavior of the Enter key (or s) varies by what is currently selected. If the currently selected object is a directory, the file browser will enter and display the contents of the directory. If the object is a file, this filename and path are copied to the status bar, and the file browser exits.

6 Command-line Options

`nano` accepts the following options from the command line:

`-A`
`--smarthome`
: Make the Home key smarter. When Home is pressed anywhere but at the very beginning of non-whitespace characters on a line, the cursor will jump to that beginning (either forwards or backwards). If the cursor is already at that position, it will jump to the true beginning of the line.

`-B`
`--backup`
: When saving a file, back up the previous version of it, using the current filename suffixed with a tilde (`~`).

`-C` *`directory`*
`--backupdir=`*`directory`*
: Make and keep not just one backup file, but make and keep a uniquely numbered one every time a file is saved — when backups are enabled. The uniquely numbered files are stored in the specified directory.

`-D`
`--boldtext`
: For the interface, use bold instead of reverse video. This will be overridden by setting the options `titlecolor`, `statuscolor`, `promptcolor`, `minicolor`, `keycolor`, `functioncolor`, `numbercolor`, and/or `selectedcolor` in your nanorc file. See [`set keycolor`], page 19, for details.

`-E`
`--tabstospaces`
: Convert each typed tab to spaces — to the number of spaces that a tab at that position would take up. (Note: pasted tabs are not converted.)

`-F`
`--multibuffer`
: Read a file into a new buffer by default.

`-G`
`--locking`
: Enable vim-style file locking when editing files.

`-H`
`--historylog`
: Save the last hundred search strings and replacement strings and executed commands, so they can be easily reused in later sessions.

`-I`
`--ignorercfiles`
: Don't look at the system's nanorc file nor at the user's nanorc.

`-J`
`--guidestripe`
: Draw a vertical stripe at the given column, to help judge the width of the text. (The color of the stripe can be changed with `set stripecolor` in your nanorc file.)

`-K`
`--rawsequences`
: Interpret escape sequences directly, instead of asking `ncurses` to translate them. (If you need this option to get some keys to work properly, it means that the terminfo terminal description that is used does not fully match the actual behavior of your terminal. This can happen when you ssh into a BSD machine, for example.) Using this option disables `nano`'s mouse support.

`-L`
`--nonewlines`
: Don't automatically add a newline when a text does not end with one. (This can cause you to save non-POSIX text files.)

`-M`
`--trimblanks`
: Snip trailing whitespace from the wrapped line when automatic hard-wrapping occurs or when text is justified.

`-N`
`--noconvert`
: Disable automatic conversion of files from DOS/Mac format.

`-O`
`--bookstyle`
: When justifying, treat any line that starts with whitespace as the beginning of a paragraph (unless auto-indenting is on).

`-P`
`--positionlog`
: For the 200 most recent files, log the last position of the cursor, and place it at that position again upon reopening such a file.

`-Q "`*`regex`*`"`
`--quotestr="`*`regex`*`"`
: Set the regular expression for matching the quoting part of a line. The default value is `"^([ \t]*([!#%:;>|}]|//))+"`. (Note that `\t` stands for a literal Tab character.) This makes it possible to rejustify blocks of quoted text when composing email, and to rewrap blocks of line comments when writing source code.

`-R`
`--restricted`
: Restricted mode: don't read or write to any file not specified on the command line. This means: don't read or write history files; don't allow suspending; don't allow spell checking; don't allow a file to be appended to, prepended to, or saved under a different name if it already has one; and don't make backup files. Restricted mode can also be activated by invoking `nano` with any name beginning with `r` (e.g. `rnano`).

`-S`
`--softwrap`
: Display over multiple screen rows lines that exceed the screen's width. (You can make this soft-wrapping occur at whitespace instead of rudely at the screen's edge, by using also `--atblanks`.) (The old short option, `-$`, is deprecated.)

`-T` *number*
`--tabsize=`*number*
: Set the displayed tab length to *number* columns. The value of *number* must be greater than 0. The default value is `8`.

`-U`
`--quickblank`
: Make status-bar messages disappear after 1 keystroke instead of after 20. Note that option `-c` (`--constantshow`) overrides this. When option `--minibar` or `--zero` is in effect, `--quickblank` makes a message disappear after 0.8 seconds instead of after the default 1.5 seconds.

`-V`
`--version`
: Show the current version number and exit.

`-W`
`--wordbounds`
: Detect word boundaries differently by treating punctuation characters as parts of words.

`-X "`*characters*`"`
`--wordchars="`*characters*`"`
: Specify which other characters (besides the normal alphanumeric ones) should be considered as parts of words. When using this option, you probably want to omit `-W` (`--wordbounds`).

`-Y` *name*
`--syntax=`*name*
: Specify the syntax to be used for highlighting. See Section 8.2 [Syntax Highlighting], page 24, for more info.

`-Z`
`--zap` Let an unmodified *`Backspace`* or *`Delete`* erase the marked region (instead of a single character, and without affecting the cutbuffer).

`-a`
`--atblanks`
When doing soft line wrapping, wrap lines at whitespace instead of always at the edge of the screen.

`-b`
`--breaklonglines`
Automatically hard-wrap the current line when it becomes overlong. (This option is the opposite of `-w` (`--nowrap`) — the last one given takes effect.)

`-c`
`--constantshow`
Constantly display the cursor position (line number, column number, and character number) on the status bar. Note that this overrides option `-U` (`--quickblank`).

`-d`
`--rebinddelete`
Interpret the *`Delete`* and *`Backspace`* keys differently so that both work properly. You should only use this option when on your system either *`Backspace`* acts like Delete or *`Delete`* acts like Backspace.

`-e`
`--emptyline`
Do not use the line below the title bar, leaving it entirely blank.

`-f` *`file`*
`--rcfile=`*`file`*
Read only this *file* for setting nano's options, instead of reading both the system-wide and the user's nanorc files.

`-g`
`--showcursor`
Make the cursor visible in the file browser (putting it on the highlighted item) and in the help viewer. Useful for braille users and people with poor vision.

`-h`
`--help` Show a summary of command-line options and exit.

`-i`
`--autoindent`
Automatically indent a newly created line to the same number of tabs and/or spaces as the previous line (or as the next line if the previous line is the beginning of a paragraph).

`-j`
`--jumpyscrolling`
: Scroll the buffer contents per half-screen instead of per line.

`-k`
`--cutfromcursor`
: Make the 'Cut Text' command (normally `^K`) cut from the current cursor position to the end of the line, instead of cutting the entire line.

`-l`
`--linenumbers`
: Display line numbers to the left of the text area. (Any line with an anchor additionally gets a mark in the margin.)

`-m`
`--mouse`
: Enable mouse support, if available for your system. When enabled, mouse clicks can be used to place the cursor, set the mark (with a double click), and execute shortcuts. The mouse will work in the X Window System, and on the console when gpm is running. Text can still be selected through dragging by holding down the Shift key.

`-n`
`--noread`
: Treat any name given on the command line as a new file. This allows `nano` to write to named pipes: it will start with a blank buffer, and will write to the pipe when the user saves the "file". This way `nano` can be used as an editor in combination with for instance `gpg` without having to write sensitive data to disk first.

`-o` *directory*
`--operatingdir=`*directory*
: Set the operating directory. This makes `nano` set up something similar to a chroot.

`-p`
`--preserve`
: Preserve the `^Q` (XON) and `^S` (XOFF) sequences so data being sent to the editor can be stopped and started. Note that option `-/` (`--modernbindings`) overrides this.

`-q`
`--indicator`
: Display a "scrollbar" on the righthand side of the edit window. It shows the position of the viewport in the buffer and how much of the buffer is covered by the viewport.

`-r` *number*
`--fill=`*number*
: Set the target width for justifying and automatic hard-wrapping at this *number* of columns. If the value is 0 or less, wrapping

will occur at the width of the screen minus *number* columns, allowing the wrap point to vary along with the width of the screen if the screen is resized. The default value is `-8`.

`-s "`*program* [*argument* ...]`"`
`--speller="`*program* [*argument* ...]`"`
Use the given program to do spell checking and correcting. By default, `nano` uses the command specified in the `SPELL` environment variable. If `SPELL` is not set, and `--speller` is not specified either, then `nano` uses its own interactive spell corrector, which requires either `hunspell` or GNU `spell` to be installed.

`-t`
`--saveonexit`
Save a changed buffer without prompting (when exiting with `^X`). This can be handy when `nano` is used as the composer of an email program.

`-u`
`--unix` Save a file by default in Unix format. This overrides nano's default behavior of saving a file in the format that it had. (This option has no effect when you also use `--noconvert`.)

`-v`
`--view` Don't allow the contents of the file to be altered: read-only mode. This mode allows the user to open also other files for viewing, unless `--restricted` is given too. (Note that this option should NOT be used in place of correct file permissions to implement a read-only file.)

`-w`
`--nowrap` Do not automatically hard-wrap the current line when it becomes overlong. This is the default. (This option is the opposite of `-b` (`--breaklonglines`) — the last one given takes effect.)

`-x`
`--nohelp` Expert mode: don't show the two help lines at the bottom of the screen. This affects the location of the status bar as well, as in Expert mode it is located at the very bottom of the editor.

Note: When accessing the help system, Expert mode is temporarily disabled to display the help-system navigation keys.

`-y`
`--afterends`
Make *`Ctrl+Right`* and *`Ctrl+Delete`* stop at word ends instead of beginnings.

`-!`
`--magic` When neither the file's name nor its first line give a clue, try using libmagic to determine the applicable syntax.

`-%`
`--stateflags`
: Use the top-right corner of the screen for showing some state flags: `I` when auto-indenting, `M` when the mark is on, `L` when hard-wrapping (breaking long lines), `R` when recording a macro, and `S` when soft-wrapping. When the buffer is modified, a star (`*`) is shown after the filename in the center of the title bar.

`-_`
`--minibar`
: Suppress the title bar and instead show information about the current buffer at the bottom of the screen, in the space for the status bar. In this "mini bar" the filename is shown on the left, followed by an asterisk if the buffer has been modified. On the right are displayed the current line and column number, the code of the character under the cursor (in Unicode format: U+xxxx), the same flags as are shown by `--stateflags`, and a percentage that expresses how far the cursor is into the file (linewise). When a file is loaded or saved, and also when switching between buffers, the number of lines in the buffer is displayed after the filename. This number is cleared upon the next keystroke, or replaced with an [i/n] counter when multiple buffers are open. The line plus column numbers and the character code are displayed only when `--constantshow` is used, and can be toggled on and off with *M-C*. The state flags are displayed only when `--stateflags` is used.

`-0`
`--zero`
: Hide all elements of the interface (title bar, status bar, and help lines) and use all rows of the terminal for showing the contents of the buffer. The status bar appears only when there is a significant message, and disappears after 1.5 seconds or upon the next keystroke. With *M-Z* the title bar plus status bar can be toggled. With *M-X* the help lines.

`-/`
`--modernbindings`
: Use key bindings similar to the ones that most modern programs use: *^X* cuts, *^C* copies, *^V* pastes, *^Z* undoes, *^Y* redoes, *^F* searches forward, *^G* searches next, *^S* saves, *^O* opens a file, *^Q* quits, and (when the terminal permits) *^H* shows help. Furthermore, *^A* sets the mark, *^R* makes replacements, *^D* searches previous, *^P* shows the position, *^T* goes to a line, *^W* writes out a file, and *^E* executes a command. Note that this overrides option `-p` (`--preserve`).

Option `-z` (`--suspendable`) has been removed. Suspension is enabled by default, reachable via *^T^Z*. (If you want a plain *^Z* to suspend nano, add `bind ^Z suspend main` to your nanorc.)

7 Feature Toggles

Toggles allow you to change certain aspects of the editor while you are editing, aspects that you would normally specify via command-line options or nanorc options. Each toggle can be flicked via a Meta-key combination — the *Meta* key is normally the *Alt* key (see Section 3.3 [Commands], page 3, for more details). The following global toggles are available:

`Constant Cursor Position Display`
: *M-C* toggles the `-c` (`--constantshow`) command-line option.

`Smart Home Key`
: *M-H* toggles the `-A` (`--smarthome`) command-line option.

`Auto Indent`
: *M-I* toggles the `-i` (`--autoindent`) command-line option.

`Cut From Cursor To End-of-Line`
: *M-K* toggles the `-k` (`--cutfromcursor`) command-line option.

`Long-Line Wrapping`
: *M-L* toggles the `-b` (`--breaklonglines`) command-line option.

`Mouse Support`
: *M-M* toggles the `-m` (`--mouse`) command-line option.

`Line Numbers`
: *M-N* toggles the `-l` (`--linenumbers`) command-line option.

`Tabs To Spaces`
: *M-O* toggles the `-E` (`--tabstospaces`) command-line option.

`Whitespace Display`
: *M-P* toggles the displaying of whitespace (see [Whitespace], page 23).

`Soft Wrapping`
: *M-S* toggles the `-S` (`--softwrap`) command-line option.

`Expert`
: *M-X* toggles the `-x` (`--nohelp`) command-line option.

`Syntax Coloring`
: *M-Y* toggles syntax coloring, when your nanorc defines syntaxes (see Section 8.2 [Syntax Highlighting], page 24).

`Hidden Interface`
: *M-Z* toggles the `-0` (`--zero`) command-line option, but without the `-x` (`--nohelp`) part. That is: it toggles just the title bar plus status bar (or the combined mini bar plus status bar), not the help lines. The latter are toggled with *M-X*.

8 Nanorc Files

Nanorc files can be used to configure `nano` to your liking without using command-line options. During startup `nano` will normally read two files: first the system-wide file, `/etc/nanorc` (the exact path may be different on your system), and then the user-specific file, either `~/.nanorc` or `$XDG_CONFIG_HOME/nano/nanorc` or `.config/nano/nanorc`, whichever exists first. However, if `--rcfile` is given, `nano` will skip the above files and will read just the specified settings file.

A nanorc file can contain `set` and `unset` commands for various options (see Section 8.1 [Settings], page 17). It can also contain commands that define syntax highlighting (see Section 8.2 [Syntax Highlighting], page 24) and commands that rebind keys (Section 8.3 [Rebinding Keys], page 27). Each command should be on a separate line, and all commands should be written in lowercase.

Options that do not take an argument are unset by default. So using the `unset` command is only needed when wanting to override a setting from the system's nanorc file in your own nanorc. Options that take an argument cannot be unset, but can be assigned the empty string.

Any command-line option overrides its nanorc setting, of course.

Quotes inside the *characters* parameters below should not be escaped. The last double quote on the line will be seen as the closing quote.

8.1 Settings

The supported settings in a nanorc file are:

`set afterends`
: Make *`Ctrl+Right`* and *`Ctrl+Delete`* stop at word ends instead of beginnings.

`set allow_insecure_backup`
: When backing up files, allow the backup to succeed even if its permissions can't be (re)set due to special OS considerations. You should NOT enable this option unless you are sure you need it.

`set atblanks`
: When soft line wrapping is enabled, make it wrap lines at blank characters (tabs and spaces) instead of always at the edge of the screen.

`set autoindent`
: Automatically indent a newly created line to the same number of tabs and/or spaces as the previous line (or as the next line if the previous line is the beginning of a paragraph).

`set backup`
: When saving a file, back up the previous version of it, using the current filename suffixed with a tilde (`~`).

`set backupdir "`*`directory`*`"`
: Make and keep not just one backup file, but make and keep a uniquely numbered one every time a file is saved — when backups are enabled with `set backup` or `--backup` or `-B`. The uniquely numbered files are stored in the specified directory.

`set boldtext`
: Use bold instead of reverse video for the title bar, status bar, key combos, function tags, line numbers, and selected text. This is overridden by setting the options `titlecolor`, `statuscolor`, `keycolor`, `functioncolor`, `numbercolor`, and/or `selectedcolor`.

`set bookstyle`
: When justifying, treat any line that starts with whitespace as the beginning of a paragraph (unless auto-indenting is on).

`set brackets "`*`characters`*`"`
: Set the characters treated as closing brackets when justifying paragraphs. This may not include blank characters. Only closing punctuation (see `set punct`), optionally followed by the specified closing brackets, can end sentences. The default value is `"")'>]}"`.

`set breaklonglines`
: Automatically hard-wrap the current line when it becomes overlong.

`set casesensitive`
: Do case-sensitive searches by default.

`set constantshow`
: Constantly display the cursor position on the status bar. Note that this overrides `quickblank`.

`set cutfromcursor`
: Use cut-from-cursor-to-end-of-line by default, instead of cutting the whole line.

`set emptyline`
: Do not use the line below the title bar, leaving it entirely blank.

`set errorcolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for the status bar when an error message is displayed. The default value is `bold,white,red`. See [`set keycolor`], page 19, for valid color names.

`set fill` *number*
: Set the target width for justifying and automatic hard-wrapping at this *number* of columns. If the value is 0 or less, wrapping will occur at the width of the screen minus *number* columns, allowing the wrap point to vary along with the width of the screen if the screen is resized. The default value is `-8`.

`set functioncolor [bold,][italic,]`*fgcolor*`,`*bgcolor*
: Use this color combination for the concise function descriptions in the two help lines at the bottom of the screen. See [`set keycolor`], page 19, for valid color names.

`set guidestripe` *number*
: Draw a vertical stripe at the given column, to help judge the width of the text. (The color of the stripe can be changed with `set stripecolor`.)

`set historylog`
: Save the last hundred search strings and replacement strings and executed commands, so they can be easily reused in later sessions.

`set indicator`
: Display a "scrollbar" on the righthand side of the edit window. It shows the position of the viewport in the buffer and how much of the buffer is covered by the viewport.

`set jumpyscrolling`
: Scroll the buffer contents per half-screen instead of per line.

`set keycolor [bold,][italic,]`*fgcolor*`,`*bgcolor*
: Use this color combination for the shortcut key combos in the two help lines at the bottom of the screen. Valid names for the foreground and background colors are: `red`, `green`, `blue`, `magenta`, `yellow`, `cyan`, `white`, and `black`. Each of these eight names may be prefixed with the word `light` to get a brighter version of that color. The word `grey` or `gray` may be used as a synonym for `lightblack`. On a Linux console, `light` does not have any effect for a background color.

  On terminal emulators that can do at least 256 colors, other valid (but unprefixable) color names are: `pink`, `purple`, `mauve`, `lagoon`, `mint`, `lime`, `peach`, `orange`, `latte`, `rosy`, `beet`, `plum`, `sea`, `sky`, `slate`, `teal`, `sage`, `brown`, `ocher`, `sand`, `tawny`, `brick`, `crimson`, and `normal` — where `normal` means the default foreground or background color. On such emulators, the color may also be specified as a three-digit hexadecimal number prefixed with `#`, with the digits representing the amounts of red, green, and blue, respectively. This tells `nano` to select from the available palette the color that approximates the given values.

Either *fgcolor* or ,*bgcolor* may be left out, and the pair may be preceded by `bold` and/or `italic` (separated by commas) to get a bold and/or slanting typeface, if your terminal can do those.

`set linenumbers`
: Display line numbers to the left of the text area. (Any line with an anchor additionally gets a mark in the margin.)

`set locking`
: Enable vim-style lock-files for when editing files.

`set magic` When neither the file's name nor its first line give a clue, try using libmagic to determine the applicable syntax. (Calling libmagic can be relatively time consuming. It is therefore not done by default.)

`set matchbrackets "`*`characters`*`"`
: Specify the opening and closing brackets that can be found by bracket searches. This may not include blank characters. The opening set must come before the closing set, and the two sets must be in the same order. The default value is `"(<[{)>]}"`.

`set minibar`
: Suppress the title bar and instead show information about the current buffer at the bottom of the screen, in the space for the status bar. In this "mini bar" the filename is shown on the left, followed by an asterisk if the buffer has been modified. On the right are displayed the current line and column number, the code of the character under the cursor (in Unicode format: U+xxxx), the same flags as are shown by `set stateflags`, and a percentage that expresses how far the cursor is into the file (linewise). When a file is loaded or saved, and also when switching between buffers, the number of lines in the buffer is displayed after the filename. This number is cleared upon the next keystroke, or replaced with an [i/n] counter when multiple buffers are open. The line plus column numbers and the character code are displayed only when `set constantshow` is used, and can be toggled on and off with *M-C*. The state flags are displayed only when `set stateflags` is used.

`set minicolor [bold,][italic,]`*`fgcolor,bgcolor`*
: Use this color combination for the mini bar. (When this option is not specified, the colors of the title bar are used.) See [`set keycolor`], page 19, for valid color names.

`set mouse` Enable mouse support, so that mouse clicks can be used to place the cursor, set the mark (with a double click), or execute shortcuts.

`set multibuffer`
: When reading in a file with `^R`, insert it into a new buffer by default.

`set noconvert`
: Don't convert files from DOS/Mac format.

`set nohelp`
: Don't display the help lists at the bottom of the screen.

`set nonewlines`
: Don't automatically add a newline when a text does not end with one. (This can cause you to save non-POSIX text files.)

`set nowrap`
: Deprecated option since it has become the default setting. When needed, use `unset breaklonglines` instead.

`set numbercolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for line numbers. See [`set keycolor`], page 19, for valid color names.

`set operatingdir "`*`directory`*`"`
: `nano` will only read and write files inside "directory" and its subdirectories. Also, the current directory is changed to here, so files are inserted from this directory. By default, the operating directory feature is turned off.

`set positionlog`
: Save the cursor position of files between editing sessions. The cursor position is remembered for the 200 most-recently edited files.

`set preserve`
: Preserve the XON and XOFF keys (`^Q` and `^S`).

`set promptcolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for the prompt bar. (When this option is not specified, the colors of the title bar are used.) See [`set keycolor`], page 19, for valid color names.

`set punct "`*`characters`*`"`
: Set the characters treated as closing punctuation when justifying paragraphs. This may not include blank characters. Only the specified closing punctuation, optionally followed by closing brackets (see `set brackets`), can end sentences. The default value is `"!.?"`.

`set quickblank`
: Make status-bar messages disappear after 1 keystroke instead of after 20. Note that option `constantshow` overrides this. When

option `minibar` or `zero` is in effect, `quickblank` makes a message disappear after 0.8 seconds instead of after the default 1.5 seconds.

`set quotestr "`*`regex`*`"`
: Set the regular expression for matching the quoting part of a line. The default value is `"^([ \t]*([!#%:;>|}]|//))+"`. (Note that `\t` stands for a literal Tab character.) This makes it possible to rejustify blocks of quoted text when composing email, and to rewrap blocks of line comments when writing source code.

`set rawsequences`
: Interpret escape sequences directly, instead of asking `ncurses` to translate them. (If you need this option to get some keys to work properly, it means that the terminfo terminal description that is used does not fully match the actual behavior of your terminal. This can happen when you ssh into a BSD machine, for example.) Using this option disables `nano`'s mouse support.

`set rebinddelete`
: Interpret the *`Delete`* and *`Backspace`* keys differently so that both work properly. You should only use this option when on your system either *`Backspace`* acts like Delete or *`Delete`* acts like Backspace.

`set regexp`
: Do regular-expression searches by default. Regular expressions in `nano` are of the extended type (ERE).

`set saveonexit`
: Save a changed buffer automatically on exit (`^`*`X`*); don't prompt.

`set scrollercolor `*`fgcolor,bgcolor`*
: Use this color combination for the indicator alias "scrollbar". See [`set keycolor`], page 19, for valid color names.

`set selectedcolor [bold,][italic,]`*`fgcolor,bgcolor`*
: Use this color combination for selected text. See [`set keycolor`], page 19, for valid color names.

`set showcursor`
: Put the cursor on the highlighted item in the file browser, and show the cursor in the help viewer, to aid braille users and people with poor vision.

`set smarthome`
: Make the Home key smarter. When Home is pressed anywhere but at the very beginning of non-whitespace characters on a line, the cursor will jump to that beginning (either forwards or backwards). If the cursor is already at that position, it will jump to the true beginning of the line.

`set softwrap`
: Display lines that exceed the screen's width over multiple screen lines. (You can make this soft-wrapping occur at whitespace instead of rudely at the screen's edge, by using also `set atblanks`.)

`set speller "`*`program`* `[`*`argument ...`*`]"`
: Use the given program to do spell checking and correcting. See [`--speller`], page 13, for details.

`set spotlightcolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for highlighting a search match. The default value is `black,lightyellow`. See [`set keycolor`], page 19, for valid color names.

`set stateflags`
: Use the top-right corner of the screen for showing some state flags: `I` when auto-indenting, `M` when the mark is on, `L` when hard-wrapping (breaking long lines), `R` when recording a macro, and `S` when soft-wrapping. When the buffer is modified, a star (`*`) is shown after the filename in the center of the title bar.

`set statuscolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for the status bar. See [`set keycolor`], page 19, for valid color names.

`set stripecolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for the vertical guiding stripe. See [`set keycolor`], page 19, for valid color names.

`set tabsize `*`number`*
: Use a tab size of *number* columns. The value of *number* must be greater than 0. The default value is `8`.

`set tabstospaces`
: Convert each typed tab to spaces — to the number of spaces that a tab at that position would take up. (Note: pasted tabs are not converted.)

`set titlecolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`*
: Use this color combination for the title bar. See [`set keycolor`], page 19, for valid color names.

`set trimblanks`
: Remove trailing whitespace from wrapped lines when automatic hard-wrapping occurs or when text is justified.

`set unix` Save a file by default in Unix format. This overrides nano's default behavior of saving a file in the format that it had. (This option has no effect when you also use `set noconvert`.)

`set whitespace "`*`characters`*`"`
: Set the two characters used to indicate the presence of tabs and spaces. They must be single-column characters. The default pair for a UTF-8 locale is "»·", and for other locales ">.".

`set wordbounds`
: Detect word boundaries differently by treating punctuation characters as part of a word.

`set wordchars "`*`characters`*`"`
: Specify which other characters (besides the normal alphanumeric ones) should be considered as parts of words. When using this option, you probably want to unset `wordbounds`.

`set zap`
: Let an unmodified *`Backspace`* or *`Delete`* erase the marked region (instead of a single character, and without affecting the cutbuffer).

`set zero`
: Hide all elements of the interface (title bar, status bar, and help lines) and use all rows of the terminal for showing the contents of the buffer. The status bar appears only when there is a significant message, and disappears after 1.5 seconds or upon the next keystroke. With *`M-Z`* the title bar plus status bar can be toggled. With *`M-X`* the help lines.

8.2 Syntax Highlighting

Coloring the different syntactic elements of a file is done via regular expressions (see the `color` command below). This is inherently imperfect, because regular expressions are not powerful enough to fully parse a file. Nevertheless, regular expressions can do a lot and are easy to make, so they are a good fit for a small editor like `nano`.

See `/usr/share/nano/` and `/usr/share/nano/extra/` for the syntax-coloring definitions that are available out of the box.

All regular expressions in `nano` are POSIX extended regular expressions (ERE). This means that `.`, `?`, `*`, `+`, `^`, `$`, and several other characters are special. The period `.` matches any single character, `?` means the preceding item is optional, `*` means the preceding item may be matched zero or more times, `+` means the preceding item must be matched one or more times, `^` matches the beginning of a line, and `$` the end, `\<` matches the start of a word, and `\>` the end, and `\s` matches a blank. It also means that lookahead and lookbehind are not possible. A complete explanation can be found in the manual of GNU grep: `info grep regular`.

Each regular expression in a `nanorc` file should be wrapped in double quotes (`""`). Multiple regular expressions can follow each other on a line by separating them with blanks. This means that a regular expression cannot contain a double quote followed by a blank. When you need this combination

inside a regular expression, then either the double quote or the blank should be put between square brackets (`[]`).

A separate syntax can be defined for each kind of file via the following commands in a nanorc file:

`syntax` *`name`* [`"`*`fileregex`*`" ...`]
: Start the definition of a syntax with this *name*. All subsequent `color` and other such commands will be added to this syntax, until a new `syntax` command is encountered.

  When `nano` is run, this syntax will be automatically activated (for the relevant buffer) if the absolute filename matches the extended regular expression *fileregex*. Or the syntax can be explicitly activated (for all buffers) by using the `-Y` or `--syntax` command-line option followed by the *name*.

  The `default` syntax is special: it takes no *fileregex*, and applies to files that don't match any syntax's regexes. The `none` syntax is reserved; specifying it on the command line is the same as not having a syntax at all.

`header "`*`regex`*`" ...`
: If from all defined syntaxes no *fileregex* matched, then compare this *regex* (or regexes) against the first line of the current file, to determine whether this syntax should be used for it.

`magic "`*`regex`*`" ...`
: If no *fileregex* matched and no `header` regex matched either, then compare this *regex* (or regexes) against the result of querying the `magic` database about the current file, to determine whether this syntax should be used for it. (This functionality only works when `libmagic` is installed on the system and will be silently ignored otherwise.)

`formatter` *`program`* [*`argument ...`*]
: Run the given *program* on the full contents of the current buffer.

`linter` *`program`* [*`argument ...`*]
: Use the given *program* to do a syntax check on the current buffer.

`comment "`*`string`*`"`
: Use the given string for commenting and uncommenting lines. If the string contains a vertical bar or pipe character (`|`), this designates bracket-style comments; for example, `"/*|*/"` for CSS files. The characters before the pipe are prepended to the line and the characters after the pipe are appended at the end of the line. If no pipe character is present, the full string is prepended; for example, `"#"` for Python files. If empty double quotes are specified, the comment/uncomment functions are disabled; for example, `""` for JSON. The default value is `"#"`.

`tabgives "`*`string`*`"`
: Make the <Tab> key produce the given *string*. Useful for languages like Python that want to see only spaces for indentation. This overrides the setting of the `tabstospaces` option.

`color [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`* `"`*`regex`*`" ...`
: Paint all pieces of text that match the extended regular expression "regex" with the given foreground and background colors, at least one of which must be specified. Valid color names are: `red`, `green`, `blue`, `magenta`, `yellow`, `cyan`, `white`, and `black`. Each of these eight names may be prefixed with the word `light` to get a brighter version of that color. The word `grey` or `gray` may be used as a synonym for `lightblack`. On a Linux console, `light` does not have any effect for a background color.

  On terminal emulators that can do at least 256 colors, other valid (but unprefixable) color names are: `pink`, `purple`, `mauve`, `lagoon`, `mint`, `lime`, `peach`, `orange`, `latte`, `rosy`, `beet`, `plum`, `sea`, `sky`, `slate`, `teal`, `sage`, `brown`, `ocher`, `sand`, `tawny`, `brick`, `crimson`, and `normal` — where `normal` means the default foreground or background color. On such emulators, the color may also be specified as a three-digit hexadecimal number prefixed with `#`, with the digits representing the amounts of red, green, and blue, respectively. This tells `nano` to select from the available palette the color that approximates the given values.

  The color pair may be preceded by `bold` and/or `italic` (separated by commas) to get a bold and/or slanting typeface, if your terminal can do those.

  All coloring commands are applied in the order in which they are specified, which means that later commands can recolor stuff that was colored earlier.

`icolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`* `"`*`regex`*`" ...`
: Same as above, except that the matching is case insensitive.

`color [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`* `start="`*`fromrx`*`" end="`*`torx`*`"`
: Paint all pieces of text whose start matches extended regular expression "fromrx" and whose end matches extended regular expression "torx" with the given foreground and background colors, at least one of which must be specified. This means that, after an initial instance of "fromrx", all text until the first instance of "torx" will be colored. This allows syntax highlighting to span multiple lines.

`icolor [bold,][italic,]`*`fgcolor`*`,`*`bgcolor`* `start="`*`fromrx`*`" end="`*`torx`*`"`
: Same as above, except that the matching is case insensitive.

`include "`*`syntaxfile`*`"`
: Read in self-contained color syntaxes from "syntaxfile". Note that "syntaxfile" may contain only the above commands, from `syntax` to `icolor`.

`extendsyntax` *`name command argument ...`*
: Extend the syntax previously defined as "*name*" with another *command*. This allows you to add a new `color`, `icolor`, `header`, `magic`, `formatter`, `linter`, `comment`, or `tabgives` command to an already defined syntax — useful when you want to slightly improve a syntax defined in one of the system-installed files (which normally are not writable).

8.3 Rebinding Keys

Key bindings can be changed via the following three commands in a nanorc file:

`bind key function menu`
: Rebinds `key` to `function` in the context of `menu` (or in all menus where the function exists when `all` is used).

`bind key "string" menu`
: Makes `key` produce `string` in the context of `menu` (or in all menus where the key exists when `all` is used). Besides literal text and/or control codes, the `string` may contain function names between braces. These functions will be invoked when the key is typed. To include a literal opening brace, use `{{}`.

`unbind key menu`
: Unbinds `key` from `menu` (or from all menus where the key exists when `all` is used).

Note that `bind key "{function}" menu` is equivalent to `bind key function menu`, except that for the latter form `nano` will check the availability of the `function` in the given `menu` at startup time (and report an error if it does not exist there), whereas for the first form `nano` will check at execution time that the `function` exists but not whether it makes any sense in the current menu. The user has to take care that a function name between braces (or any sequence of them) is appropriate. Strange behavior can result when it is not.

The format of `key` should be one of:

`^`*`X`*
: where *X* is a Latin letter, or one of several ASCII characters (@,], \, ^, _), or the word "Space". Example: `^C`.

`M-`*`X`*
: where *X* is any ASCII character except [, or the word "Space". Example: `M-8`.

`Sh-M-`*X* where *X* is a Latin letter. Example: `Sh-M-U`. By default, each Meta+letter keystroke does the same as the corresponding Shift+Meta+letter. But when any Shift+Meta bind is made, that will no longer be the case, for all letters.

`F`*n* where *n* is a numeric value from 1 to 24. Example: `F10`. (Often, `F13` to `F24` can be typed as `F1` to `F12` with Shift.)

`Ins` or `Del`

Rebinding `^M` (Enter) or `^I` (Tab) is probably not a good idea. Rebinding `^[` (Esc) is not possible, because its keycode is the starter byte of Meta keystrokes and escape sequences. Rebinding any of the dedicated cursor-moving keys (the arrows, Home, End, PageUp and PageDown) is not possible. On some terminals it's not possible to rebind `^H` (unless `--raw` is used) because its keycode is identical to that of the Backspace key.

Valid names for the `function` to be bound are:

`help` Invokes the help viewer.

`cancel` Cancels the current command.

`exit` Exits from the program (or from the help viewer or file browser).

`writeout` Writes the current buffer to disk, asking for a name.

`savefile` Writes the current file to disk without prompting.

`insert` Inserts a file into the current buffer (at the current cursor position), or into a new buffer when option `multibuffer` is set.

`whereis` Starts a forward search for text in the current buffer — or for filenames matching a string in the current list in the file browser.

`wherewas` Starts a backward search for text in the current buffer — or for filenames matching a string in the current list in the file browser.

`findprevious` Searches the next occurrence in the backward direction.

`findnext` Searches the next occurrence in the forward direction.

`replace` Interactively replaces text within the current buffer.

`cut` Cuts and stores the current line (or the marked region).

`copy` Copies the current line (or the marked region) without deleting it.

`paste` Pastes the currently stored text into the current buffer at the current cursor position.

`zap` Throws away the current line (or the marked region). (This function is bound by default to *`Alt+Delete`*.)

`chopwordleft`
: Deletes from the cursor position to the beginning of the preceding word. (This function is bound by default to *Shift+Ctrl+Delete*. If your terminal produces `^H` for *Ctrl+Backspace*, you can make *Ctrl+Backspace* delete the word to the left of the cursor by rebinding *^H* to this function.)

`chopwordright`
: Deletes from the cursor position to the beginning of the next word. (This function is bound by default to *Ctrl+Delete*.)

`cutrestoffile`
: Cuts all text from the cursor position till the end of the buffer.

`mark`
: Sets the mark at the current position, to start selecting text. Or, when it is set, unsets the mark.

`location`
: Reports the current position of the cursor in the buffer: the line, column, and character positions.

`wordcount`
: Counts and reports on the status bar the number of lines, words, and characters in the current buffer (or in the marked region).

`execute`
: Prompts for a program to execute. The program's output will be inserted into the current buffer (or into a new buffer when *M-F* is toggled).

`speller`
: Invokes a spell-checking program, either the default `hunspell` or GNU `spell`, or the one defined by `--speller` or `set speller`.

`formatter`
: Invokes a full-buffer-processing program (if the active syntax defines one). (The current buffer is written out to a temporary file, the program is run on it, and then the temporary file is read back in, replacing the contents of the buffer.)

`linter`
: Invokes a syntax-checking program (if the active syntax defines one). If this program produces lines of the form "filename:linenum:charnum: some message", then the cursor will be put at the indicated position in the mentioned file while showing "some message" on the status bar. You can move from message to message with *PgUp* and *PgDn*, and leave linting mode with *^C* or *Enter*.

`justify`
: Justifies the current paragraph (or the marked region). A paragraph is a group of contiguous lines that, apart from possibly the first line, all have the same indentation. The beginning of a paragraph is detected by either this lone line with a differing indentation or by a preceding blank line.

`fulljustify`
: Justifies the entire current buffer (or the marked region).

`indent`
: Indents (shifts to the right) the current line or the marked lines.

`unindent`
: Unindents (shifts to the left) the current line or the marked lines.

`comment`
: Comments or uncomments the current line or the marked lines, using the comment style specified in the active syntax.

`complete`
: Completes (when possible) the fragment before the cursor to a full word found elsewhere in the current buffer.

`left`
: Goes left one position (in the editor or browser).

`right`
: Goes right one position (in the editor or browser).

`up`
: Goes one line up (in the editor or browser).

`down`
: Goes one line down (in the editor or browser).

`scrollup`
: Scrolls the viewport up one row (meaning that the text slides down) while keeping the cursor in the same text position, if possible. (This function is bound by default to *`Alt+Up`*. If *`Alt+Up`* does nothing on your Linux console, see the FAQ: `https://nano-editor.org/dist/latest/faq.html#4.1`.)

`scrolldown`
: Scrolls the viewport down one row (meaning that the text slides up) while keeping the cursor in the same text position, if possible. (This function is bound by default to *`Alt+Down`*.)

`center`
: Scrolls the line with the cursor to the middle of the screen.

`prevword`
: Moves the cursor to the beginning of the previous word.

`nextword`
: Moves the cursor to the beginning of the next word.

`home`
: Moves the cursor to the beginning of the current line.

`end`
: Moves the cursor to the end of the current line.

`beginpara`
: Moves the cursor to the beginning of the current paragraph.

`endpara`
: Moves the cursor to the end of the current paragraph.

`prevblock`
: Moves the cursor to the beginning of the current or preceding block of text. (Blocks are separated by one or more blank lines.)

`nextblock`
: Moves the cursor to the beginning of the next block of text.

`toprow`
: Moves the cursor to the first row in the viewport.

`bottomrow`
: Moves the cursor to the last row in the viewport.

`pageup` Goes up one screenful.

`pagedown` Goes down one screenful.

`firstline`
: Goes to the first line of the file.

`lastline` Goes to the last line of the file.

`gotoline` Goes to a specific line (and column if specified). Negative numbers count from the end of the file (and end of the line).

`findbracket`
: Moves the cursor to the bracket (or brace or parenthesis, etc.) that matches (pairs) with the one under the cursor. See [`set matchbrackets`], page 20.

`anchor` Places an anchor at the current line, or removes it when already present. (An anchor is visible when line numbers are activated.)

`prevanchor`
: Goes to the first anchor before the current line.

`nextanchor`
: Goes to the first anchor after the current line.

`prevbuf` Switches to editing/viewing the previous buffer when multiple buffers are open.

`nextbuf` Switches to editing/viewing the next buffer when multiple buffers are open.

`verbatim` Inserts the next keystroke verbatim into the file, or begins Unicode input when a hexadecimal digit is typed (see Section 3.2 [Entering Text], page 3, for details).

`tab` Inserts a tab at the current cursor location.

`enter` Inserts a new line below the current one.

`delete` Deletes the character under the cursor.

`backspace`
: Deletes the character before the cursor.

`recordmacro`
: Starts the recording of keystrokes — the keystrokes are stored as a macro. When already recording, the recording is stopped.

`runmacro` Replays the keystrokes of the last recorded macro.

`undo` Undoes the last performed text action (add text, delete text, etc).

`redo` Redoes the last undone action (i.e., it undoes an undo).

`refresh` Refreshes the screen.

`suspend` Suspends the editor and returns control to the shell (until you tell the process to resume execution with *`fg`*).

`casesens` Toggles whether searching/replacing ignores or respects the case of the given characters.

`regexp` Toggles whether searching/replacing uses literal strings or regular expressions.

`backwards`
Toggles whether searching/replacing goes forward or backward.

`older` Retrieves the previous (earlier) entry at a prompt.

`newer` Retrieves the next (later) entry at a prompt.

`flipreplace`
Toggles between searching for something and replacing something.

`flipgoto` Toggles between searching for text and targeting a line number.

`flipexecute`
Switches from inserting a file to executing a command.

`flippipe` When executing a command, toggles whether the current buffer (or marked region) is piped to the command.

`flipnewbuffer`
Toggles between inserting into the current buffer and into a new empty buffer.

`flipconvert`
When reading in a file, toggles between converting and not converting it from DOS/Mac format. Converting is the default.

`dosformat`
When writing a file, switches to writing a DOS format (CR/LF).

`macformat`
When writing a file, switches to writing a Mac format.

`append` When writing a file, appends to the end instead of overwriting.

`prepend` When writing a file, 'prepends' (writes at the beginning) instead of overwriting.

`backup` When writing a file, creates a backup of the current file.

`discardbuffer`
When about to write a file, discard the current buffer without saving. (This function is bound by default only when option `--saveonexit` is in effect.)

`browser` Starts the file browser (in the Read File and Write Out menus), allowing to select a file from a list.

`gotodir` Goes to a directory to be specified, allowing to browse anywhere in the filesystem.

`firstfile`
Goes to the first file in the list when using the file browser.

`lastfile` Goes to the last file in the list when using the file browser.

`nohelp` Toggles the presence of the two-line list of key bindings at the bottom of the screen. (This toggle is special: it is available in all menus except the help viewer and the linter. All further toggles are available in the main menu only.)

`zero` Toggles the presence of title bar and status bar.

`constantshow`
Toggles the constant display of the current line, column, and character positions.

`softwrap` Toggles the displaying of overlong lines on multiple screen lines.

`linenumbers`
Toggles the display of line numbers in front of the text.

`whitespacedisplay`
Toggles the showing of whitespace.

`nosyntax` Toggles syntax highlighting.

`smarthome`
Toggles the smartness of the Home key.

`autoindent`
Toggles whether a newly created line will contain the same amount of leading whitespace as the preceding line — or as the next line if the preceding line is the beginning of a paragraph.

`cutfromcursor`
Toggles whether cutting text will cut the whole line or just from the current cursor position to the end of the line.

`breaklonglines`
Toggles whether long lines will be hard-wrapped to the next line. (The old name of this function, 'nowrap', is deprecated.)

`tabstospaces`
Toggles whether typed tabs will be converted to spaces.

`mouse` Toggles mouse support.

Valid names for `menu` are:

`main` The main editor window where text is entered and edited.

`help` The help-viewer menu.

`search` The search menu (AKA whereis).

`replace` The 'search to replace' menu.

`replacewith`
The 'replace with' menu, which comes up after 'search to replace'.

`yesno` The 'yesno' menu, where the Yes/No/All/Cancel question is asked.

`gotoline` The 'goto line (and column)' menu.

`writeout` The 'write file' menu.

`insert` The 'insert file' menu.

`browser` The 'file browser' menu, for selecting a file to be opened or inserted or written to.

`whereisfile`
The 'search for a file' menu in the file browser.

`gotodir` The 'go to directory' menu in the file browser.

`execute` The menu for inserting the output from an external command, or for filtering the buffer (or the marked region) through an external command, or for executing one of several tools.

`spell` The menu of the integrated spell checker where the user can edit a misspelled word.

`linter` The linter menu, which allows jumping through the linting messages.

`all` A special name that encompasses all menus. For `bind` it means all menus where the specified `function` exists; for `unbind` it means all menus where the specified `key` exists.

9 Pico Compatibility

`nano` emulates Pico quite closely, but there are some differences between the two editors:

`Hard-Wrapping`
: Unlike Pico, `nano` does not automatically hard-wrap the current line when it becomes overlong during typing. This hard-wrapping can be switched on with the `--breaklonglines` option. With that option, `nano` by default breaks lines at screen width minus eight columns, whereas Pico does it at screen width minus six columns. You can make `nano` do as Pico by using `--fill=-6`.

`Scrolling`
: By default, `nano` will scroll just one line (instead of half a screen) when the cursor is moved to a line that is just out of view. And when paging up or down, `nano` keeps the cursor in the same screen position as much as possible, instead of always placing it on the first line of the viewport. The Pico-like behavior can be obtained with the `--jumpyscrolling` option.

`Edit Area`
: Pico never uses the line directly below the title bar, leaving it always blank. `nano` includes this line in the editing area, in order to not waste space, and because in this way it is slightly clearer where the text starts. If you are accustomed to this line being empty, you can get it back with the `--emptyline` option.

`Interactive Replace`
: Instead of allowing you to replace either just one occurrence of a search string or all of them, `nano`'s replace function is interactive: it will pause at each found search string and query whether to replace this instance. You can then choose Yes, or No (skip this one), or All (don't ask any more), or Cancel (stop with replacing).

`Search and Replace History`
: When the option `-H` or `--historylog` is given (or set in a nanorc file), text entered as search or replace strings is stored. These strings can be accessed with the up/down arrow keys at their respective prompts, or you can type the first few characters and then use *`Tab`* to cycle through the matching strings. A retrieved string can subsequently be edited.

`Position History`
: When the option `-P` or `--positionlog` is given (or set in a nanorc file), `nano` will store the position of the cursor when you close a file, and will place the cursor in that position again when you later reopen the file.

`Current Cursor Position`
: The output of the "Display Cursor Position" command (`^C`) displays not only the current line and character position of the cursor, but also (between the two) the current column position.

`Spell Checking`
: In the internal spell checker misspelled words are sorted alphabetically and trimmed for uniqueness, such that the words 'apple' and 'Apple' will be prompted for correction separately.

`Writing Selected Text to Files`
: When using the Write-Out key (`^O`), text that has been selected using the marking key (`^^`) can not just be written out to a new (or existing) file, it can also be appended or prepended to an existing file.

`Reading Text from a Command`
: When using the Read-File key (`^R`), `nano` can not just read a file, it can also read the output of a command to be run (`^X`).

`Reading from Working Directory`
: By default, Pico will read files from the user's home directory (when using `^R`), but it will write files to the current working directory (when using `^O`). `nano` makes this symmetrical: always reading from and writing to the current working directory — the directory that `nano` was started in.

`File Browser`
: In the file browser, `nano` does not implement the Add, Copy, Rename, and Delete commands that Pico provides. In `nano` the browser is just a file browser, not a file manager.

`Toggles`
: Many options which alter the functionality of the program can be "toggled" on or off using Meta key sequences, meaning the program does not have to be restarted to turn a particular feature on or off. See Chapter 7 [Feature Toggles], page 16, for a list of options that can be toggled. Or see the list at the end of the main internal help text (`^G`) instead.

10 Building and its Options

Building `nano` from source is straightforward if you are familiar with compiling programs with autoconf support:

```
tar -xf nano-x.y.tar.gz
cd nano-x.y
./configure
make
make install
```

The possible options to `./configure` are:

`--disable-browser`
: Exclude the file browser that can be called with `^T` when wanting to read or write a file.

`--disable-color`
: Exclude support for syntax coloring. This also eliminates the `-Y` command-line option, which allows choosing a specific syntax.

`--disable-comment`
: Exclude the single-keystroke comment/uncomment function (`M-3`).

`--disable-extra`
: Exclude the Easter egg: a crawl of major contributors.

`--disable-formatter`
: Exclude the code for calling a formatting tool.

`--disable-help`
: Exclude the help texts (`^G`). This makes the binary much smaller, but also makes it difficult for new users to learn more than very basic things about using the editor.

`--disable-histories`
: Exclude the code for handling the history files: the search and replace strings that were used, the commands that were executed, and the cursor position at which each file was closed. This also eliminates the `-H` and `-P` command-line options, which switch on the storing of search/replace strings, executed commands, and cursor positions.

`--disable-justify`
: Exclude the text-justification functions (`^J` and `M-J`).

`--disable-libmagic`
: Exclude the code for using the library of magic-number tests (for determining the file type and thus which syntax to use for

coloring — in most cases the regexes for filename and header line will be enough).

`--disable-linenumbers`
: Exclude the ability to show line numbers. This also eliminates the `-l` command-line option, which turns line numbering on.

`--disable-linter`
: Exclude the code for calling a linting tool.

`--disable-mouse`
: Exclude all mouse functionality. This also eliminates the `-m` command-line option, which enables the mouse functionality.

`--disable-multibuffer`
: Exclude support for opening multiple files at a time and switching between them. This also eliminates the `-F` command-line option, which causes a file to be read into a separate buffer by default.

`--disable-nanorc`
: Exclude support for reading the nanorc files at startup. With such support, you can store custom settings in a system-wide and a per-user nanorc file rather than having to pass command-line options to get the desired behavior. See Chapter 8 [Nanorc Files], page 17, for more info. Disabling this also eliminates the `-I` command-line option, which inhibits the reading of nanorc files.

`--disable-operatingdir`
: Exclude the code for setting an operating directory. This also eliminates the `-o` command-line option, which sets the operating directory.

`--disable-speller`
: Exclude the code for spell checking. This also eliminates the `-s` command-line option, which allows specifying an alternate spell checker.

`--disable-tabcomp`
: Exclude tab completion (when nano asks for a filename or search string or replace string or command to execute).

`--disable-wordcomp`
: Exclude word completion (`^]`).

`--disable-wrapping`
: Exclude all hard-wrapping of overlong lines. This also eliminates the `-b` and `-w` command-line options, which switch automatic long-line wrapping on and off, respectively.

`--enable-tiny`
: This option implies all of the above. It also disables some other internals of the editor, like the function toggles, the marking of text, the undo/redo code, line anchors, the recording and playback of a macro, softwrapping, and the cut-to-end-of-line code. These things stay disabled also when using the enabling counterpart of the above options together with `--enable-tiny` to switch specific features back on.

`--enable-debug`
: Include some code for runtime debugging output. This can get messy, so chances are you only want this feature when you're working on the nano source.

`--disable-nls`
: Exclude Native Language support. This will disable the use of any available GNU `nano` translations.

`--enable-utf8`
: Include support for handling and displaying Unicode files. This requires a "wide" version of the curses library.

`--disable-utf8`
: Exclude support for handling and displaying Unicode files. Normally the configure script auto-detects whether to enable UTF-8 support or not. You can use this or the previous option to override that detection.

`--enable-altrcname=`*`name`*
: Use the file with the given *name* (in the user's home directory) as nano's settings file, instead of the default `.nanorc`.

Table of Contents