GNU nano

a small and friendly text editor
version 2.9.8

Chris Allegretta

This manual documents the GNU `nano` editor.

This manual is part of the GNU `nano` distribution.

You may contact the author by e-mail: `chrisa@asty.org`

1 Introduction

GNU `nano` is a small and friendly text editor. Besides basic text editing, `nano` offers many extra features, such as an interactive search-and-replace, undo/redo, syntax coloring, smooth scrolling, auto-indentation, go-to-line-and-column-number, feature toggles, 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 possible while offering a superset of Pico's functionality. See Chapter 9 [Pico Compatibility], page 30, for more details on how `nano` and Pico differ.

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

2 Invoking

The usual way to invoke `nano` is:

```
nano [FILE]
```

But it is also possible to specify one or more options (see the next section), and to edit several files in a row. Additionally, 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. So a more complete command synopsis is:

```
nano [OPTION]... [[+LINE[,COLUMN]|+,COLUMN] FILE]...
```

Normally, however, you set your preferred options in a `nanorc` file (see Chapter 7 [Nanorc Files], page 15). 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 Command-line Options

`nano` takes 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`
: Use bold text instead of reverse video text.

`-E`
`--tabstospaces`
: Convert typed tabs to spaces.

`-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.

`-K`
`--rebindkeypad`
: Interpret the numeric keypad keys so that they all work properly. You should only need to use this option if they don't, as mouse support won't work properly with this option enabled.

`-L`
`--nonewlines`
: Don't automatically add a newline when a file does not end with one.

`-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`
`--morespace`
: Use the blank line below the title bar as extra editing space.

`-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, used when justifying. The default value is `"^([ \t]*([#:>|}]|//))+"`. Note that `\t` stands for a literal Tab character.

`-R`
`--restricted`
: Restricted mode: don't read or write to any file not specified on the command line; don't read any nanorc files nor history files; don't allow suspending nor 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 use backup files. This restricted mode is also accessible by invoking `nano` with any name beginning with `r` (e.g. `rnano`).

`-S`
`--smooth`
: Enable smooth scrolling. Text will scroll line-by-line, instead of the usual chunk-by-chunk behavior.

`-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`
: Do quick status-bar blanking: status-bar messages will disappear after 1 keystroke instead of 25. Note that option `-c` (`--constantshow`) overrides this.

`-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. This overrides option `-W` (`--wordbounds`).

`-Y` *name*
`--syntax=`*name*
: Specify the syntax to be used for highlighting. See Section 7.2 [Syntax Highlighting], page 20, for more info.

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

`-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 key differently so that both Backspace and Delete work properly. You should only need to use this option if Backspace acts like Delete on your system.

`-g`
`--showcursor`
: Make the cursor visible in the file browser, putting it on the highlighted item. Useful for braille users.

`-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).

`-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.

`-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.

`-q`
`--quiet` Obsolete option. Recognized but ignored.

`-r` *number*
`--fill=`*number*
Hard-wrap lines at column *number* (by inserting a newline character). If the given value is 0 or less, wrapping will occur at the

width of the screen minus the given amount, allowing the wrapping width to vary along with the width of the screen if and when it is resized. The default value is `-8`. This option conflicts with `-w` (`--nowrap`); the last one given takes effect.

`-s` *program*
`--speller=`*program*
: 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 the GNU `spell` program to be installed.

`-t`
`--tempfile`
: Don't ask whether to save a modified buffer when exiting with `^X`, but assume yes. This option is useful when `nano` is used as the composer of a mailer 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. Note that this option should NOT be used in place of correct file permissions to implement a read-only file.

`-w`
`--nowrap`
: Don't hard-wrap long lines at any length. This option conflicts with `-r` (`--fill`); the last one given takes effect.

`-x`
`--nohelp`
: Expert Mode: don't show the Shortcut List 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 stop at word ends instead of beginnings.

`-z`
`--suspend`
: Enable the ability to suspend `nano` using the system's suspend keystroke (usually `^Z`).

`-$`
`--softwrap`
: Enable 'soft wrapping'. This will make `nano` attempt to display the entire contents of any line, even if it is longer than the screen width, by continuing it over multiple screen lines. Since `$` normally refers to a variable in the Unix shell, you should specify this option last when using other options (e.g. `nano -wS$`) or pass it separately (e.g. `nano -wS -$`).

`-b`
`-e`
`-f`
`-j`
: Ignored, for compatibility with Pico.

4 Editor Basics

4.1 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 six-digit hexadecimal number (starting with *0* or *1*) 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.

4.2 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.

4.3 The Cutbuffer

Text can be cut from a file, a whole line at a time, by using the 'Cut Text' command (default key binding: *^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 back into the file with the 'Uncut Text' command (default key binding: *^U*).

A line of text can be copied into the cutbuffer (without cutting it) with the 'Copy Text' command (default key binding: *M-6*).

4.4 The Mark

Text can be selected by first 'setting the Mark' (default key bindings: *^6* and *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*).

On some terminals, it is also possible to select text by holding down *Shift* together with the cursor keys. Such a selection is cancelled upon any cursor movement where *Shift* isn't held.

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

4.5 Screen Layout

The default screen of nano consists of five areas. From top to bottom these are: the title bar, a blank line, 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. These two lines are called the Shortcut List.

4.6 Search and Replace

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

A regular expression in a search string always covers just one line; it cannot span multiple lines. And when replacing (with *^* or *M-R*) the replacement string cannot contain a newline (LF).

4.7 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 shortcut list 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.

4.8 Limitations

Justifications (^J) are not yet covered by the general undo system. So after a justification that is not immediately undone, earlier edits cannot be undone any more. The workaround is, of course, to exit without saving.

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.

5 Built-in Help

The built-in help system 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 system.

6 Feature Toggles

Toggles allow you to change on-the-fly certain aspects of the editor which would normally be specified via command-line options. They are invoked via Meta-key sequences (see Section 4.2 [Commands], page 9, for more info). The following global toggles are available:

`Backup Files`
: *Meta-B* toggles the `-B` (`--backup`) command-line option.

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

`Multiple File Buffers`
: *Meta-F* toggles the `-F` (`--multibuffer`) command-line option.

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

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

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

`Long-Line Wrapping`
: *Meta-L* toggles the `-w` (`--nowrap`) command-line option.

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

`No Conversion From DOS/Mac Format`
: *Meta-N* toggles the `-N` (`--noconvert`) command-line option.

`More Space For Editing`
: *Meta-O* toggles the `-O` (`--morespace`) command-line option.

`Whitespace Display`
: *Meta-P* toggles the displaying of whitespace (see [Whitespace], page 20).

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

`Smooth Scrolling`
: *Meta-S* toggles the `-S` (`--smooth`) command-line option.

`Expert/No Help`
: *Meta-X* toggles the `-x` (`--nohelp`) command-line option.

`Color Syntax Highlighting`
: *Meta-Y* toggles color syntax highlighting (if your nanorc defines syntaxes — see Section 7.2 [Syntax Highlighting], page 20).

`Suspension`
: *Meta*-*Z* toggles the `-z` (`--suspend`) command-line option.

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

`Soft Wrapping`
: *Meta*-*$* toggles the `-$` (`--softwrap`) command-line option.

7 Nanorc Files

The nanorc files contain the default settings for `nano`. They should be in Unix format, not in DOS or Mac format. During startup, `nano` will first read the system-wide settings, from /etc/nanorc (the exact path might be different), and then the user-specific settings, either from `~/.nanorc` or from `$XDG_CONFIG_HOME/nano/nanorc` or from `.config/nano/nanorc`, whichever exists first.

A nanorc file accepts a series of "set" and "unset" commands, which can be used to configure `nano` on startup without using command-line options. Additionally, there are some commands to define syntax highlighting and to rebind keys — see Section 7.2 [Syntax Highlighting], page 20, and Section 7.3 [Rebinding Keys], page 22. `nano` will read one command per line.

Options in nanorc files take precedence over `nano`'s defaults, and command-line options override nanorc settings. Also, 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 of the system's nanorc file in your own nanorc. Options that take an argument cannot be unset.

Quotes inside string parameters don't have to be escaped with backslashes. The last double quote in the string will be treated as its end. For example, for the `brackets` option, `""')>]}"` will match `"`, `'`, `)`, `>`, `]`, and `}`.

7.1 Settings

The supported settings in a nanorc file are:

`set afterends`
: Make Ctrl+Right 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 backwards`
: Obsolete option. Recognized but ignored. `^Q` is available to start a backward search.

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

`set brackets "`*`string`*`"`
: 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 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. (The old form of this option, `set cut`, is deprecated.)

`set errorcolor` *`fgcolor,bgcolor`*
: Use this color combination for the status bar when an error message is displayed. See [`set functioncolor`], page 16, for valid color names.

`set fill` *`number`*
: Hard-wrap lines at column number *number*. If *number* is 0 or less, the maximum line length will be the screen width less *number* columns. The default value is `-8`. This option conflicts with `nowrap`; the last one given takes effect.

`set functioncolor` *`fgcolor,bgcolor`*
: Use this color combination for the concise function descriptions in the two help lines at the bottom of the screen. Valid names for foreground and background color are: `white`, `black`, `blue`, `green`, `red`, `cyan`, `yellow`, `magenta`, and `normal` — where

`normal` means the default foreground or background color. The name of the foreground color may be prefixed with `bright`. And either *fgcolor* or ,*bgcolor* may be left out.

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

`set keycolor` *`fgcolor,bgcolor`*
: Use this color combination for the shortcut key combos in the two help lines at the bottom of the screen. See [`set functioncolor`], page 16, for valid color names.

`set linenumbers`
: Display line numbers to the left of the text area.

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

`set matchbrackets "`*`string`*`"`
: Set 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 morespace`
: Use the blank line below the title bar as extra editing space.

`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`
: When a file does not end with a newline, don't automatically add one.

`set nopauses`
: Don't pause between warnings at startup. This means that only the last one will be visible (when there are multiple ones).

`set nowrap`
: Don't hard-wrap text at all. This option conflicts with `fill`; the last one given takes effect.

`set numbercolor` *`fgcolor,bgcolor`*
: Use this color combination for line numbers. See [`set functioncolor`], page 16, 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 punct "`*`string`*`"`
: 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`
: Do quick status-bar blanking: status-bar messages will disappear after 1 keystroke instead of 25. Note that `constantshow` overrides this.

`set quiet` Obsolete option. Recognized but ignored.

`set quotestr "`*`regex`*`"`
: The email-quote string, used to justify email-quoted paragraphs. This is an extended regular expression. The default value is `"^([ \t]*([#:>|}]|//))+"`. Note that `\t` stands for a literal Tab character.

`set rebinddelete`
: Interpret the Delete key differently so that both Backspace and Delete work properly. You should only need to use this option if Backspace acts like Delete on your system.

`set rebindkeypad`
: Interpret the numeric keypad keys so that they all work properly. You should only need to use this option if they don't, as mouse support won't work properly with this option enabled.

`set regexp`
: Do extended regular expression searches by default.

`set selectedcolor` *`fgcolor,bgcolor`*
: Use this color combination for selected text. See [`set functioncolor`], page 16, for valid color names.

`set showcursor`
: Put the cursor on the highlighted item in the file browser, to aid braille users.

`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 smooth`
: Use smooth scrolling by default.

`set softwrap`
: Enable soft line wrapping for easier viewing of very long lines.

`set speller "`*`program`*`"`
: Use the given program to do spell checking and correcting. See [`--speller`], page 7, for details.

`set statuscolor` *`fgcolor,bgcolor`*
: Use this color combination for the status bar. See [`set functioncolor`], page 16, for valid color names.

`set suspend`
: Allow `nano` to be suspended.

`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 typed tabs to spaces.

`set tempfile`
: Save automatically on exit, don't prompt.

`set titlecolor` *`fgcolor,bgcolor`*
: Use this color combination for the title bar. See [`set functioncolor`], page 16, for valid color names.

`set trimblanks`
: Remove trailing whitespace from wrapped lines when automatic hard-wrapping occurs or when text is justified. (The old form of this option, `set justifytrim`, is deprecated.)

`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 view` Disallow file modification.

`set whitespace "`*`string`*`"`
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 "`*`string`*`"`
Specify which other characters (besides the normal alphanumeric ones) should be considered as parts of words. This overrides the option `wordbounds`.

7.2 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`.

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 if the current filename matches the extended regular expression *fileregex*. Or the syntax can be explicitly activated 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 *fileregex*. 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.)

`linter` *`program`* [*`arg ...`*]
: Use the given *program* to do a syntax check on the current buffer. (This overrides the speller function.)

`formatter` *`program`* [*`arg ...`*]
: Use the given *program* to automatically reformat the text in the current buffer — useful for a programming language like Go. (This overrides the speller and linter functions.)

`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 `"#"`.

`color` *`fgcolor,bgcolor`* `"`*`regex`*`" ...`
: Display all pieces of text that match the extended regular expression "regex" with foreground color "fgcolor" and background color "bgcolor", at least one of which must be specified. Valid colors for foreground and background are: white, black, red, blue, green, yellow, magenta, and cyan. You may use the prefix "bright" to get a stronger color highlight for the foreground. If your terminal supports transparency, not specifying a "bgcolor" tells `nano` to attempt to use a transparent background.

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

`color` *`fgcolor,bgcolor`* `start="`*`fromrx`*`" end="`*`torx`*`"`
: Display all pieces of text whose start matches extended regular expression "fromrx" and whose end matches extended regular expression "torx" with foreground color "fgcolor" and background color "bgcolor", 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` *`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`* `[`*`arg ...`*`]`
: Extend the syntax previously defined as "*name*" with another *command*. This allows you to add a new `color`, `icolor`, `header`, `magic`, `comment`, `linter`, or `formatter` 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).

7.3 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 by using `all`).

`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). The `string` can consist of text or commands or a mix of them. (To enter a command into the `string`, precede its keystroke with *`M-V`*.)

`unbind key menu`
: Unbinds `key` from `menu` (or from all menus where it exists by using `all`).

The format of `key` should be one of:

`^` followed by an ASCII character or the word "Space". Example: `^C`.

`M-` followed by a ASCII character or the word "Space". Example: `M-C`.

`F` followed by a numeric value from 1 to 16. Example: `F10`.

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 the 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.

`searchagain`
Repeats the last search command without prompting.

`findprevious`
As `searchagain`, but always in the backward direction.

`findnext` As `searchagain`, but always in the forward direction.

`replace` Interactively replaces text within the current buffer.

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

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

`uncut` Copies the currently stored text into the current buffer at the current cursor position.

`mark` Sets the mark at the current position, to start selecting text.

`cutwordleft`
Cuts from the cursor position to the beginning of the preceding word.

`cutwordright`
Cuts from the cursor position to the beginning of the next word.

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

`curpos` Shows the current cursor position: the line, column, and character positions.

`wordcount`
Counts the number of words, lines and characters in the current buffer.

`speller` Invokes a spell-checking program (or linting program, or formatter program, if the active syntax defines such a thing).

`justify` Justifies the current paragraph. 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.

`indent` Indents (shifts to the right) the currently marked text.

`unindent` Unindents (shifts to the left) the currently marked text.

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

`complete` Completes 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.

`scrolldown` Scrolls the viewport down one row (meaning that the text slides up) while keeping the cursor in the same text position, if possible.

`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.

`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 (brace, parenthesis, etc.) that matches (pairs) with the one under the cursor.

`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.

`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 (if the suspending function is enabled, see the "suspendenable" entry below).

`casesens` Toggles case sensitivity in searching (search/replace menus only).

`regexp` Toggles whether searching/replacing is based on literal strings or regular expressions.

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

`prevhistory`
: Shows the previous history entry in the prompt menus (e.g. search).

`nexthistory`
: Shows the next history entry in the prompt menus (e.g. search).

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

`flipgoto` Toggles between searching for text and targeting a line number. (The form `gototext` is deprecated.)

`flipexecute`
Toggles between inserting a file and 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.

`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 `--tempfile` is in effect.)

`browser` Starts the file browser, 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 when using the file browser (reading or writing files).

`lastfile` Goes to the last file when using the file browser (reading or writing files).

`nohelp` Toggles the presence of the two-line list of key bindings at the bottom of the screen.

`constantshow`
Toggles the constant display of the current line, column, and character positions. (The form `constupdate` is deprecated.)

`morespace`
Toggles the presence of the blank line that 'separates' the title bar from the file text.

`smoothscroll`
: Toggles smooth scrolling (when moving around with the arrow keys).

`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. (The form `cuttoend` is deprecated.)

`nowrap`
: Toggles whether long lines will be hard-wrapped to the next line.

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

`backupfile`
: Toggles whether a backup will be made of the file to be edited.

`multibuffer`
: Toggles whether a file is inserted into the current buffer or read into a new buffer.

`mouse`
: Toggles mouse support.

`noconvert`
: Toggles automatic conversion of files from DOS/Mac format.

`suspendenable`
: Toggles whether the suspend shortcut (normally `^Z`) will suspend the editor.

Valid names for `menu` are:

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

`search`
: The search menu (AKA whereis).

`replace`
: The 'search to replace' menu.

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

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

`writeout`
: The 'write file' menu.

`insert`
: The 'insert file' menu.

`extcmd`
: The menu for inserting output from an external command, reached from the insert menu.

`help`
: The help-viewer menu.

`spell`
: The interactive spell checker Yes/no menu.

`linter`
: The linter menu.

`browser`
: The file browser, for choosing a file to read from or write to.

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

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

`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.

8 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.

9 Pico Compatibility

`nano` attempts to emulate Pico as closely as possible, but there are some differences between the editors:

`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 the a nanorc file), text entered as search or replace strings is stored. These strings can be accessed with the up/down arrow keys, 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.

`Hard-Wrapping`
: By default, `nano` hard-wraps lines at screen width minus eight columns, whereas Pico does it at screen width minus six columns. You can make `nano` do the same as Pico by using `--fill=-6`.

`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 6 [Feature Toggles], page 13, 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 Configure Options

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

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

The possible options to `./configure` are:

`--disable-browser`
: Disable the mini file browser that can be called with `^T` when reading or writing files.

`--disable-color`
: Disable support for the syntax coloring of files. This also eliminates the `-Y` command-line option, which chooses a specific syntax.

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

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

`--disable-help`
: Disable the help function. Doing this makes the binary much smaller, but makes it difficult for new users to learn more than very basic things about using the editor.

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

`--disable-justify`
: Disable the justify and unjustify functions.

`--disable-libmagic`
: Disable the use of the library of magic-number tests (for determining the file type and thus which syntax to use for colouring — often the tests on filename extension and header line will be enough).

`--disable-linenumbers`
: Disable the line-numbering function (*M-#*). This also eliminates the `-l` command-line option, which turns line numbering on.

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

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

`--disable-nanorc`
: Disable 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 7 [Nanorc Files], page 15, for more info. Disabling this also eliminates the `-I` command-line option, which inhibits the reading of nanorc files.

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

`--disable-speller`
: Disable use of the spell checker. This also eliminates the `-s` command-line option, which allows specifying an alternate spell checker.

`--disable-tabcomp`
: Disable tab completion (when nano asks for a filename or a search string).

`--disable-wordcomp`
: Disable word completion (*^J*).

`--disable-wrapping`
: Disable all hard-wrapping of overlong lines. This also eliminates the `-w` command-line option, which switches long-line wrapping off.

`--enable-tiny`
: This option implies all of the above. It also disables some other internals of the editor, like the marking code, the cut-to-end-of-line code, and the function toggles. By using the enabling counterpart of the above options together with `--enable-tiny`, specific features can be switched back on — but a few cannot.

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

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

`--disable-wrapping-as-root`
: Disable hard-wrapping of overlong lines by default when `nano` is run as root.

`--enable-utf8`
: Enable support for reading and writing Unicode files. This will require either a wide version of curses, or a UTF-8-enabled version of Slang.

`--disable-utf8`
: Disable support for reading and writing 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`.

`--with-slang`
: Compile `nano` against Slang instead of against ncurses or other curses libraries.

Table of Contents