Nano:
THE THIRD WAY

Masterclass

Hate Vi's terseness or the Emacs key combinations? **Mike Saunders** introduces Nano, a text editor that packs a lot of punch for its diminutive size.

We seem to live in an increasingly polarised world. You either love Microsoft or you love Apple. You drink Coke or you drink Pepsi. You're either with us or you're against us. Time is money, and in this fast-paced world, who can possibly afford to consider anything other than the simple options repeatedly presented to us?

Of course, as a Linux user, you've already chosen a third way – a separate route from the two major commercial operating systems, Windows and OS X. You know that there's almost always another option aside from A or B. You've made a deliberate decision to try something different, and among the rewards are the operating system you've got on your trusty computer today. But what about text editors? If you look at nigh-on any discussion about this topic on the internet, it appears that there are only two players in the game: *Vi(m)* and *Emacs.* Both editors are hugely powerful and complex – indeed, one of the common criticisms of *Emacs* is that it's effectively an operating system that just happens to include a text editor. But they have steep learning curves and obscure functionality, behind cryptic commands and Twister-like key combos.

Let's take a look at a third option, *Nano* (**www.nano-editor.org**). This editor is great for a number of reasons:

> » It's small, lightweight and is included in most distributions;
> » It's easy to use and gives plenty of on-screen feedback;
> » It includes many power-user features to compete with *Vi(m)* and *Emacs.*

Nano runs in text mode at the command line; this may prompt you to think: "Why would I want to learn a text mode editor? *Kate*, *Gedit* and *FooEdit 2000* do everything I need to on my desktop." Well, firstly, all regular Linux users and administrators should be well-versed in a command line editor. If something goes wrong with the *X Window System*, for example, you'll end up at a CLI prompt and your skills will be essential for editing config files.

Secondly, if you set up a Linux machine as a server, it almost certainly won't include *X* or any kind of graphical tools, so you'll be using a text mode editor frequently. Learning some core skills beyond simply opening and saving files can make you work much more productively, so that's what we'll be looking at here.

As for a quick history, *Elm* was a Unix email client dating back to the late 80s. Then came *Pine*, which some users said was an acronym for 'Pine Is Not Elm'. *Pico* was *Pine*'s text editor, but wasn't totally free software by the GNU definition. So *Tip*, (*Tip Isn't Pico*), was created and later renamed to *Nano*, (*Nano's ANOther editor*). Fun, eh?

> **"Nano includes many power-user features to compete with Emacs."**

Finding your way around

Before we delve into *Nano*'s features, let's spend a moment to get fully acquainted with the interface for the editor. Entering **nano** on its own at the command line brings up *Nano* in its default state. If you've tried *Emacs* before but have been put off by the lack of feedback on the screen, you'll be pleasantly surprised by *Nano* – and the less said about *Vi* the better. You can start entering text straight away, using the cursor keys to move around.

Along the top is a bar containing the *Nano* version number and the name of the file being edited. Below, we haven't opened a file yet, so it says 'New Buffer'. To open a file, you simply type its name: **nano filename.txt**. Along the bottom is a mightily helpful quick reference to the most common keybindings, in this format:

`^O WriteOut`

This info is deliberately very short to cram as much as possible on to the screen, so see the 'Quick keybinding reference' box for the full details. Here, the ^ symbol refers to the Ctrl key on the bottom-left of your keyboard, so ^O means Ctrl+O (it can be lowercase or uppercase). WriteOut saves the file to disk, so if you press Ctrl+O you'll see the following prompt at the bottom:

`File Name to Write:`

If you've opened an existing text file, the filename will be filled in here automatically, ready for you to hit Enter. Otherwise, you can type in a new filename. However, here's where we start to see how powerful *Nano* really is. Hit Ctrl+T and the screen will change into an excellent two-column file browser. Use the cursor keys to navigate through directories to find the filename you want to save to.

› **Pain! Look at this horribly broken up text. If only there was a straightforward way to make it reflow nicely...**
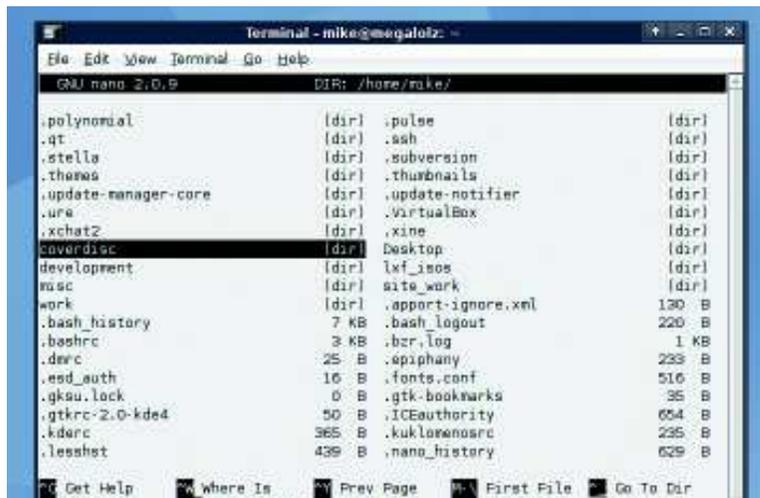
› *Nano* **comes with this file navigator that you explore with the arrow keys.**

As you'd expect, this file browser is available for other tasks. Hit Ctrl+X to exit the browser, then Ctrl+C to cancel the save operation. Now press Ctrl+R to read a file and insert it into the current editing session. When the filename prompt appears, hit Ctrl+T and find the file you want to include.

So that's opening, saving and inserting files covered. You'll always find the most important keybindings at the bottom of the screen, so you can't get lost. Let's look at cut and paste. *Nano* works differently from many editors in this regard, but you'll find it extremely efficient after using it for a while.

Type in five lines of random text, and then move the cursor to the second line. Hit Ctrl+K and the line will disappear. It doesn't matter where you position the cursor on the line; the whole thing will be 'kut' away. Now, the contents of the line are stored on the clipboard, so you can move somewhere else and press Ctrl+U to 'uncut' it (paste it into the document).

You can do this for multiple lines too. Position the cursor on the top line of the chunk you want to cut, and then press Ctrl+K on as many lines as you want to cut. This will overwrite the clipboard's previous contents with the chunk you cut out. When you're done, move the cursor to the place at which you want to paste the text, and hit Ctrl+U – all the lines removed in the cutting session will be dropped back into the document. Press Ctrl+U multiple times for multiple pastes.

The line number isn't displayed by default, but if you press Ctrl+C, you'll see a status message showing the line, column and character the cursor is currently on, plus a percentage indication of how far you are through the file.

Quick keybinding reference

» **Ctrl+X** Exit the editor. If you've edited text without saving, you'll be prompted as to whether you really want to exit.
» **Ctrl+O** Write (output) the current contents of the text buffer to a file. A filename prompt will appear; press Ctrl+T to open the file navigator shown above.
» **Ctrl+R** Read a text file into the current editing session. At the filename prompt, hit Ctrl+T for the file navigator.

» **Ctrl+K** Cut a line into the clipboard. You can press this repeatedly to copy multiple lines, which are then stored as one chunk.
» **Ctrl+J** Justify (fill out) a paragraph of text. By default, this reflows text to match the width of the editing window.
» **Ctrl+U** Uncut text, or rather, paste it from the clipboard. Note that after a Justify operation, this turns into unjustify.
» **Ctrl+T** Check spelling.

» **Ctrl+W** Find a word or phrase. At the prompt, use the cursor keys to go through previous search terms, or hit Ctrl+R to move into replace mode. Alternatively you can hit Ctrl+T to go to a specific line.
» **Ctrl+C** Show current line number and file information.
» **Ctrl+G** Get help; this provides information on navigating through files and common keyboard commands.

Power-users rejoice

Nano carries forward one of *Pico*'s most celebrated features – text justification. This is where a block of text is arranged to fit a certain page width by changing the wrap and reorganising the words. Few text editors include such functionality, especially at the lighter end of the range, but because *Pico* was used for email and newsgroup messages, where you'd want to reflow multiple levels of quoted text to fit on an 80-character-wide screen, this feature was given much attention.

In a new *Nano* session, enter the following:

```
the cat sat
on a very large
fluffy mat
```

Now, imagine that you want to rearrange this text so that it's all on a single line. In many text editors, you'd have to do this manually, going to the end of each line and hitting Delete.

Not so with *Nano*. Just position the cursor anywhere in those three lines and hit Ctrl+J, and you'll see this:

```
the cat sat on a very large fluffy mat
```

A status message appears towards the bottom of the screen, saying 'Can now UnJustify'. Hit Ctrl+U to revert.

By default, *Nano* justifies text to the width of the editor window. If you're running it inside an *X* terminal, create some text and use cut/paste to fill half of the window with it. Hit Ctrl+J to justify the text, then resize the window and hit Ctrl+J again. You'll see that the text is reflowed to fit the window size. *Nano* justifies based on paragraphs – chunks of text with blank lines separating them.

So far so good, but the real power comes with the **-r** option. Quit *Nano* and restart it with **nano -r30**. This tells *Nano* that we want the justify operation always to wrap to 30 characters (columns of text), regardless of the size of window.

There are some great uses for this feature. For instance, on the *Linux Format* website we send out a monthly newsletter to signed-up readers with news, features and so on (**www.linuxformat.com/newsletters**). Because we're geeks of the highest order, our terminal windows are always insanely huge. However, when sending out the newsletter we

> **If you're a programmer, take the time to set up syntax highlighting.**

> **With a quick tap of Ctrl+J, the text is smoothed out to the way it should be. There is peace around the Earth.**

want it to be limited to a certain width, so that it will display on virtually any machine. Different mail clients handle plain text differently, so to make things as simple and reliable as possible, we use **nano -r68 newsletter.txt**, type away and hit Ctrl+J when we want to format it for sending.

Another example: imagine you've found a great command on the internet with a large number of parameters, such as a **mencode** command for converting a video file. Due to the web page presentation, when you copy and paste the command, it ends up over 20 lines. Instead of going through and removing all newline characters, simply start *Nano* with **nano -r1000**. Paste the text in (Shift+Insert works in many *X* terminals), hit Ctrl+J and it'll be folded into one neat line that you can paste into a terminal. Alt+J justifies all paragraphs in the file at once.

On to some other features. *Nano* can automatically create backups of files after you've saved them. Pass the **-B** option (**nano -B config_file.txt**), and when you hit Ctrl+O to save a file, it will make a backup of the older version with a tilde character at the end (**config_file.txt~**). This is well worth doing if you want to edit a critical system file – if something goes wrong, you can retrieve the older version in a snap.

Search and replace is fully supported: hit Ctrl+W (Where is...?) and you'll be given a prompt. Enter a word to find it or hit Ctrl+R to do a replacement. If you hit Alt+R at this prompt, you'll see that it turns to **[Regexp]**, which enables you to search using regular expressions. Using the up and down cursor keys, you can navigate through previous search terms – even ones from earlier sessions. These are stored in your **home** directory in **.nano_history**.

Nano's interface is customisable: as you get more familiar with the editor, you won't need the quick reference for keybindings at the bottom. Using the **-x** command, you can tell *Nano* never to show it. With the **-c** option you'll always see the status bar (showing the line number you're on), while with **-S** (capital S), you can force *Nano* to scroll through text one line at a time, rather than per page. You can mix all of these:

```
nano -xcSr68 filename.txt
```

Here we want no help, a permanently visible status bar, smooth scrolling and text justification at 68 characters. Just like mum used to make!

> ## "Nano carries forward one of Pico's most celebrated features."
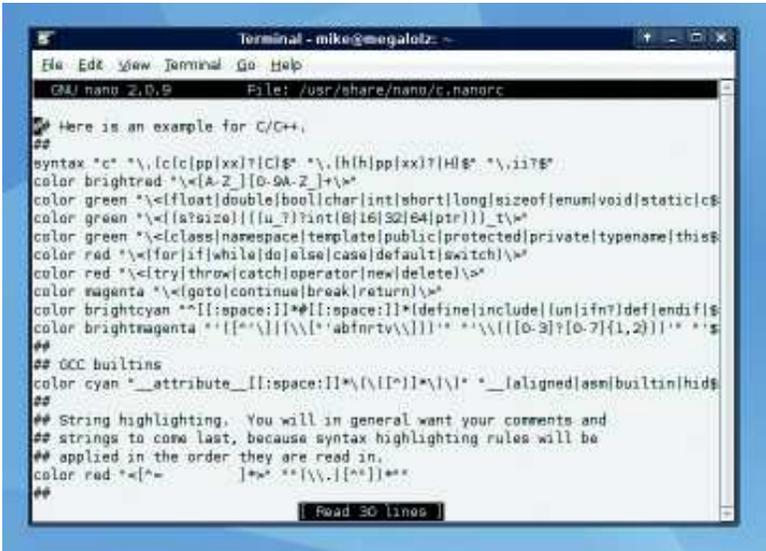
Developer delights

There's even syntax highlighting for various programming languages in *Nano*, although this is disabled by default for performance reasons. On very old machines, parsing large source code files to give every part its correct colour is rather CPU-intensive, but on any machine built in the last 10 years, you shouldn't notice the performance impact.

To enable syntax highlighting, open the *Nano* configuration file (see the box below) and scroll down to the bottom section. You'll see a bunch of lines like this:

```
## C/C++
# include "/usr/share/nano/c.nanorc"
```

The hash marks are comments, so by removing the hash mark from the second line, you tell *Nano* to read **c.nanorc** at startup – the file that describes syntax highlighting for C and C++ source files. If you look further down the list, you'll see that you can include syntax highlighting definitions for Python, Perl, Ruby, Java, HTML, Tex and other formats.

Let's create a highlighting file for a language not supported by default in *Nano*. Go into **/usr/share/nano** and copy **sh.nanorc** to the language you want to support, for example, **php.nanorc**. The reason we're using **sh.nanorc** is that it's the easiest to modify: if you look at **c.nanorc**, for instance, it's an entertainingly complex mixture of regular expressions that may just explode your brain.

In **php.nanorc**, the first non-comment line you'll see is:

```
syntax "sh" "\.sh$"
```

This tells *Nano* to use the formatting here in any files ending with **.sh**, so change both instances of **sh** to **php** here. Look down a couple of lines and you'll see:

```
color green "\<(case|do|done|elif|...
```

When the keywords **case**, **do**, **done**, **elif** and so on appear in the source code file, they're coloured **green**. You can delete keywords and add new ones, but make sure the structure of the line stays intact and that keywords are separated by **|** pipe characters. The line underneath probably looks like gibberish if you're not used to regular expressions:

```
color green "(\{|\}|\(|\)|\;)...
```

Let us explain: this defines the colour used for certain non-alphanumeric characters, such as brackets and semicolons. As with the keywords, these are separated by pipe characters, but because they're special characters that could be part of instructions to the syntax highlighting parsing engine, we have to 'escape' them using a backslash.

If we just had **{|}|(|)** to identify the brackets in the PHP code, it wouldn't work: *Nano*'s parser would get confused, thinking they're instructions. By using a backslash before

each one, we tell *Nano* that they're to be treated as individual characters. So you can place other non-alphanumeric characters in there, but put a backslash before each one.

Most of the other lines in your **php.nanorc** follow the same format. Towards the bottom you'll see this line:

```
color cyan "(^|[[:space:]])#.*$"
```

This tells the parser that all chunks of text starting with a hash symbol should be coloured **cyan** until the end of the line – in other words, shell script comments. In PHP, we can also use double slashes for comments, so make a copy of the above line and change the hash to **//**:

```
color cyan "(^|[[:space:]])//.*$"
```

With this addition, when you open PHP files, you'll see that double-slash comments are now coloured correctly.

There are other useful features for programmers too. When editing text, hit Alt+I to enable auto-indent mode, where the cursor will remain in the same column each time you hit Enter. This is particularly useful in languages where white space is syntactically important, such as Python.

When saving a file, you might want to convert it to a different format for another platform. Linux and Windows use different invisible line-ending characters, which can cause headaches. Try opening a Unix-created text file in Windows XP's *Notepad*, for instance – the newlines are all messed up.

So, when you've got the filename prompt, you can hit Alt+D to switch between Unix and Windows/DOS style text files. (Alt+M switches you to Mac formatted ones, but that's only really applicable to Mac OS 9 and earlier.) With the power of *Nano*, you can take someone's broken-up notes, flow it into line-break-free text, and then give it back to them in a format that their silly non-Linux OS can understand. Huzzah! **LXF**

> **Syntax highlighting makes source code much easier on the eyes.**

Editing the config file

Nano's main configuration file is **/etc/nanorc**. However, it's not the only one: **/etc/nanorc** can include other configuration files, such as the syntax highlighting definitions in **/usr/share/nano**. In **nanorc**, you can set default options so that you don't explicitly need to provide them at the command line. Each is stated in this format:

```
# set backup
```

This is equivalent to the **-B** option at the command line. Here, the line is commented out

with a hash mark, so *Nano* will perform the default action, which is to not save backups.

If you remove the hash mark, it will turn backups on by default, or if you explicitly want to tell *Nano* that it shouldn't make them, change the line to read like this:

```
unset backup
```

All the options are accompanied by a snippet of description text, so you shouldn't find that anything there is unfathomable.

> **Tweak some of the deeper workings of *Nano* in the configuration file.**