
Terminator Documentation

Release 2.1.1 stable

Stephen Boddy

Jul 08, 2023

Contents:

1	What is Terminator?	3
1.1	Licensing	4
1.2	Document history	5
1.3	Getting Started	6
1.4	Preferences Window	22
1.5	Layouts and the Layout Launcher	39
1.6	The Grouping Menu	43
1.7	Plugins	51
1.8	Advanced Usage	61
1.9	Frequently Asked Questions	66
1.10	Getting involved	72
	Bibliography	83

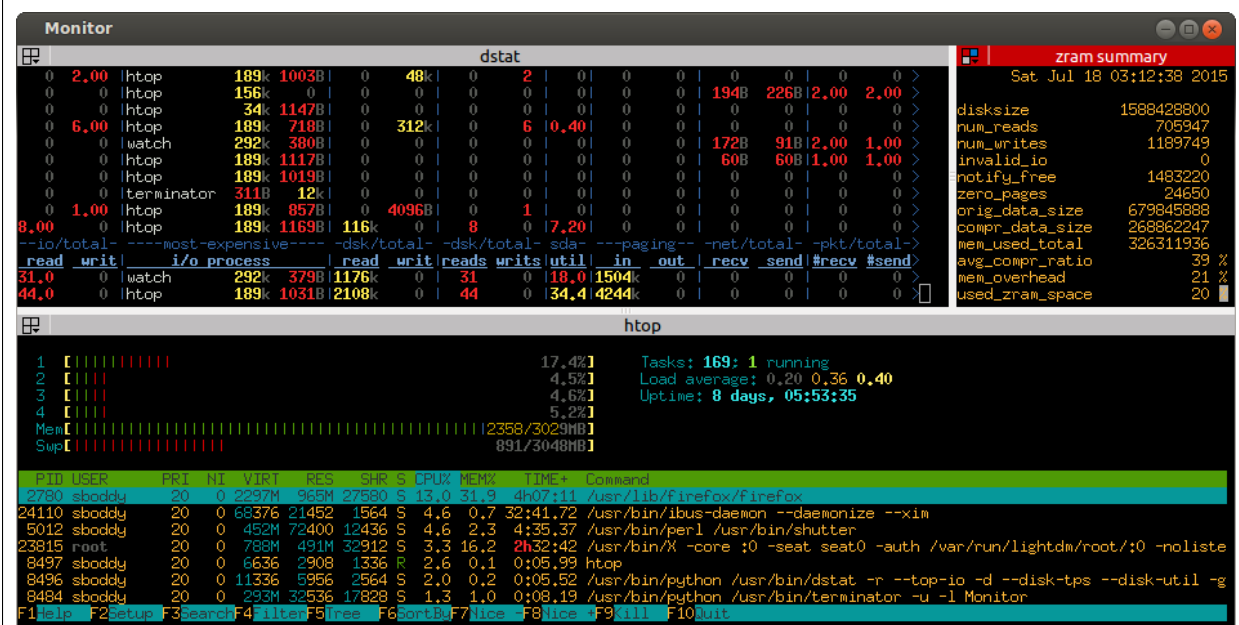


Sometimes it is not always clear just how many little shortcuts and features there are in Terminator. This manual hopes to reduce the confusion.

What is Terminator?

At its simplest Terminator is a terminal emulator like xterm, gnome-terminal, konsole, etc. At its most complex it lets you fly... metaphorically at least. Take a look at the following list:

From the simple...



- Arrange terminals in a grid-like structure
- Tabs
- Drag and drop re-ordering of terminals
- Lots of keyboard shortcuts

- Save multiple layouts and profiles via GUI preferences editor
- Simultaneous typing to arbitrary groups of terminals
- Extensible through plugins
- Dynamic and customizable layouts

and lots more...

To the ridiculous...

In case it's not obvious this is faked up. I use more complex setups, but I'm not putting real work into the documentation.



1.1 Licensing

The Terminator **Application** is written and distributed under the terms of the GNU GPL v2 licence. Please note that it is not v2+.

For specifics of any included **Plugins** please see the [Plugins](#) page.

The **ConfigObj** library was sourced from [voidspace.org.uk](#), and is licensed under the [BSD 3-Clause](#) licence, as stated [here](#).

Man pages and **Misc documents** have no explicitly different licensing, so it is assumed that they fall under the applications [GNU GPL v2](#) licence insofar as it can be said to apply to non-source code files.

The main **Terminator icon** was created by *Cory Kontros*, and provided under the [CC-BY-SA](#) licence.

This **Manual** and **API documentation** are wholly new pieces created by the current maintainer *Steve Boddy*, and are distributed under the [CC-BY-SA licence](#), as are the horrific attempts by yours truly at using Cory's icon to provide page identities. It is updated by *Felix Mölder* and also published under the [CC-BY-SA licence](#).

The **Documentation Theme** is the [Read The Docs](#) theme by *Dave Snider*, which is distributed under the [MIT licence](#). The theme is available on [GitHub](#).



1.2 Document history

Documentation process started 2015-07-17 by Stephen Boddy.

All contributions and improvements are welcome.

Up-dated for	Date	Author / Editor	Notes
0.97, r1598	2015-08-07	Stephen Boddy	Initial creation
0.97, r1621	2015-08-21	Stephen Boddy	Minor changes/corrections Added links to dev docs
0.98	2015-08-26	Stephen Boddy	Stick a fork in it, it's done
0.98, r1663	2015-09-30	Stephen Boddy	Add the new PuTTY paste mode Add new Remotinator commands FAQ for other Terminator Add Bug handling flow Minor changes/corrections
0.98, r1667	2015-10-01	Stephen Boddy	Add the new Smart copy mode
1.91, r1759	2017-03-29	Stephen Boddy	Updates for the GTK3 Port
1.92	2017-04-18	Markus Frosch	Updates for 1.92 release
2.0	2020-10-06	Matt Rose	Updates for 2.0 release
2.0.1	2020-10-11	Matt Rose	Minor bugfix release
2.1	2021-01-04	Matt Rose	Updates for 2.1 release

For a more detailed overview and changelogs see [release notes](#) and [changelogs](#).

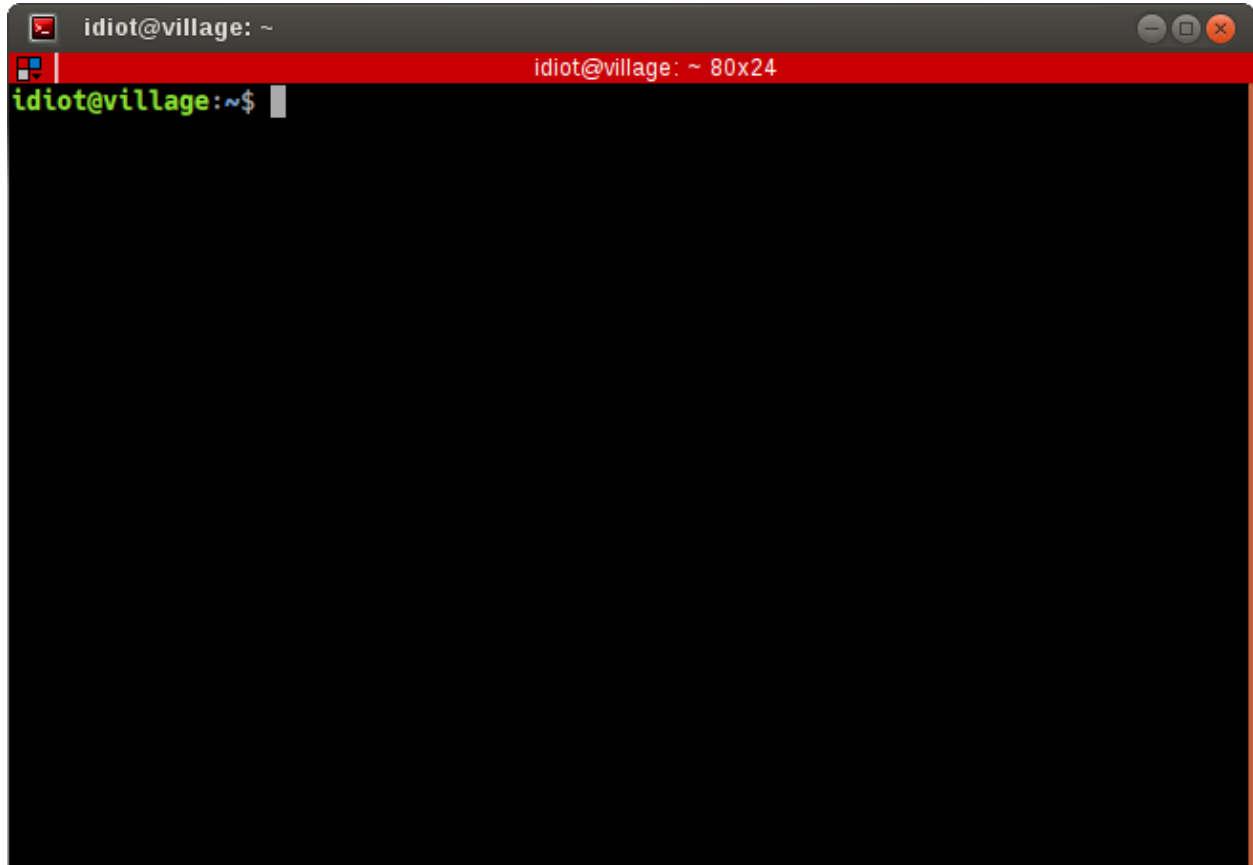
Note: Ideally this documentation should be kept up-to-date with the changes as they go in. This way things don't get missed. There could be some lag between releases, but it should definitely be updated for a new release.



1.3 Getting Started

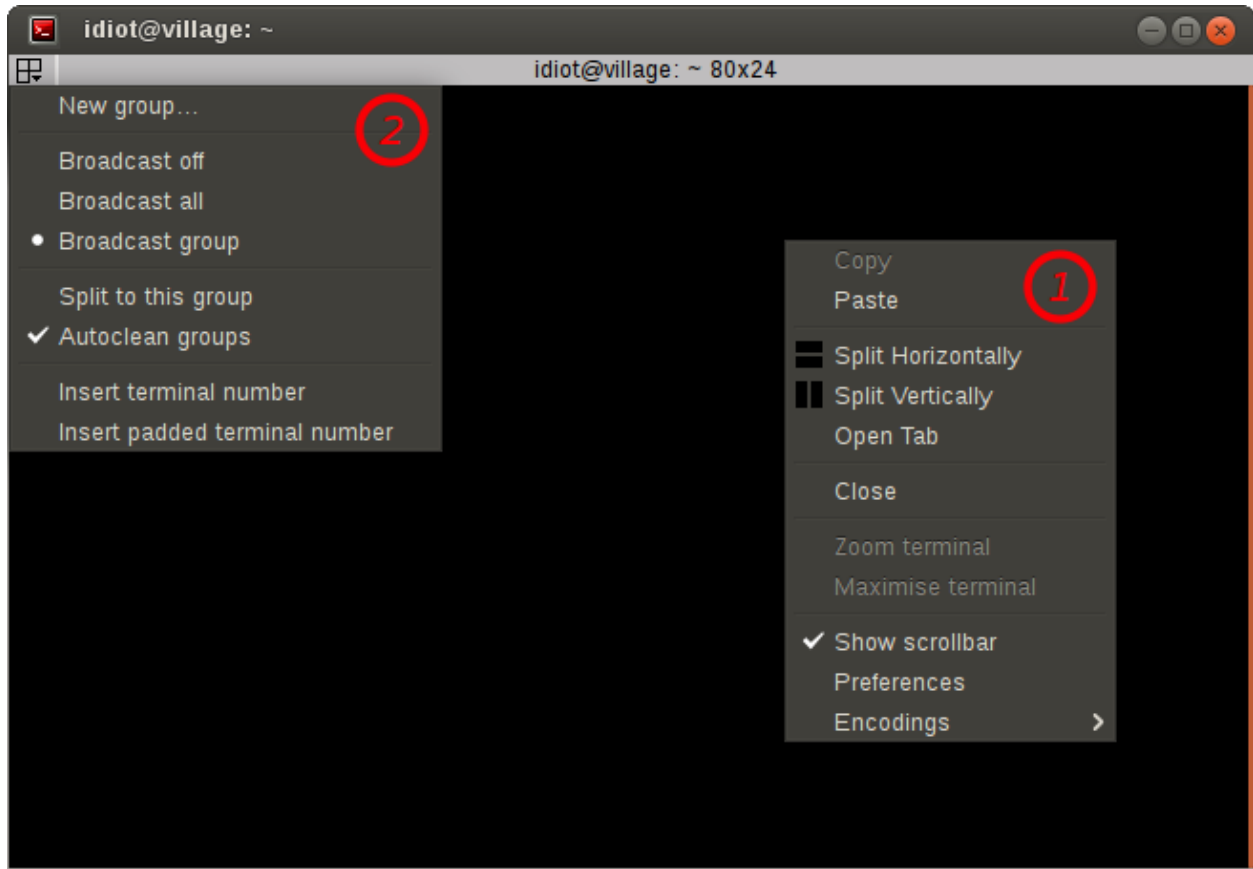
This page is an introduction and tutorial that will get you familiar with Terminator's features. Additional functional areas are explored in other pages, but at the end of this page you'll be getting a good idea of the power of Terminator.

When you start Terminator for the first time you will get a default, minimal window, looks like the following:



There may be some cosmetic differences, but it should look fairly similar. It may in fact look a little too minimal to some of you, but this is a deliberate policy. Keep the focus on the terminal, not on a cluttered interface. This is why we don't waste space on a traditional menu bar and toolbar. Even the terminal scrollbar and titlebar (the red strip on the top) can be turned off, although you may lose ease-of-access to some of Terminator's more powerful features then.

Many functions are triggered with keyboard shortcuts. But mousers aren't completely abandoned. Lets look again at the basic interface, but with the two primary menus showing:



Note: You will never see a window that looks like this, as it is impossible to have both menus up at the same time.

1. *The Context Menu* - This is the main menu reached with `right-click` in a terminal, and will let you access all the settings, profiles, shortcuts and configurations. However, it is kept brief to avoid the mega-menus that sometimes grow unchecked. In case you have enabled the `PuTTY style paste` from the **Preferences**, you can use `shift+F10` to access the *context-menu*.
2. *The Grouping Menu* - This is reached with a `left-click` on the trio of coloured boxes in the titlebar. Later, when we cover Grouping and broadcasting to multiple terminals we will cover this properly. For now it is enough to know where it is and how to trigger it.

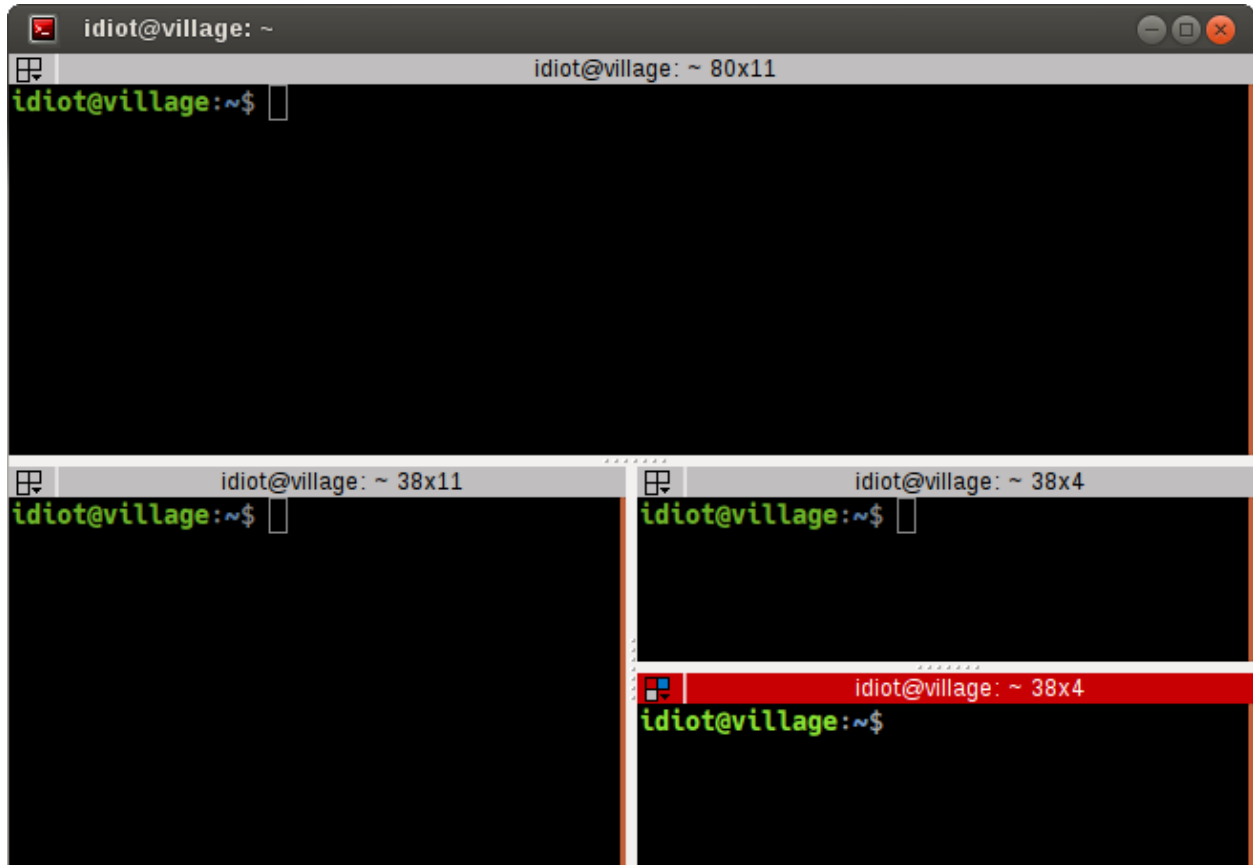
Note: By default titlebars are shown. If the titlebar has been hidden *The Grouping Menu* functions will be added as a sub-menu to *The Context Menu*.

1.3.1 The Context Menu

The context menu is split into six parts. The first part is the standard Copy and Paste for text that has been highlighted with the mouse. The shortcuts are:

Action	Default Shortcut
Copy	Ctrl+Shift+C
Paste	Ctrl+Shift+V

The second section is where the fun starts. **Split Horizontally** and **Split Vertically** are used to divide the current space for the current terminal half. Your original terminal takes the top/left in half, and a new terminal is started and placed in the right/bottom half. You can repeat this as often as you wish, sub-dividing down until the terminals are completely impractical. Here's a window that is split Horizontally, Vertically, and Horizontally again:



Note: People sometimes raise the ambiguity of the terminology used, and disagree as to which way round Horizontal and Vertical are used. It has been the way it is for a very long time. Changing it now will just confuse existing users, so I won't be changing it. Besides, I happen to agree with the way round it is. So deal with it.

Between the terminals you can see a space that is a splitter grab handle. You can grab these and drag them, and the terminals will resize. In this way Terminator acts a lot like a tiling window manger. It lets you arrange many terminals in a single view, allowing adjustments as your needs change.

The last item in this part of the menu is to **Open tab**. This will give you a tab like most other terminals do. Unlike most other terminals, in Terminator you can also split the terminals in each tab as often as you like.

Note: The same effects could have been achieved with *shortcuts*, and is the case for most actions.

The third part of the menu will **Close** the current terminal. It's on its own to prevent accidents.

The entries in the fourth part allow you to temporarily focus on one terminal. **Zoom terminal** will zoom into the current terminal hiding all other terminals and tabs, and increasing the the size of the font. This can be handy to eliminate distractions, give yourself a bit more space for the current task, or even when giving presentations or training. **Maximise terminal** is almost identical, except that it does not increase the size of the terminal font.

When you are zoomed or maximised it is not possible to split terminals, or create new tabs, so the entries for those actions disappear from the menu. So too do the zoom and maximise options, and in their place is a **Restore all terminals** entry. This will take you back to your windows original layout, and restore the font size if necessary.

Warning: An outstanding issue is that sometimes the font size selected when zooming in can be a bit extreme. You can use [Terminal zooming](#) to increase and decrease the font size if this happens. This will not affect the restored font size.

The fifth part of the menu has three items. **Show scrollbar** will toggle the scrollbar on a per terminal basis. There is also a way to define this in the Profiles. **Preferences** lets you configure and tune Terminator to better suit your needs and is further described [here](#). Lastly, **Encodings** will allow you to select a different encoding to the default of UTF-8. Finally we have **Layouts...** where for now the only option default. For more options see [here](#).

There are actually additional optional items that can be added to the menu that will only be shown if you enable those [Plugins](#) that add menu items.

1.3.2 Navigating around

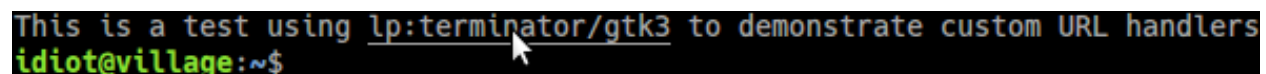
Apart from the obvious of clicking the terminal for focus, there are a number of shortcuts that will move the focus around:

Action	Options	Default Shortcut
Move focus	Up, Down, Left, Right	Alt+<Arrow>
Cycle to terminal	Next, Prev	Ctrl+Tab or Shift+Tab
Focus to terminal	Next, Prev	Ctrl+Shift+N/P
Switch to tab #	1 to 10	
Switch tab	Previous, Next	Ctrl+PgUp/PgDn
Context menu		Menu Key
Help ¹		F1

Once the Context menu is visible, it can be navigated with the arrow keys.

Click-able items

Terminator can make strings of text that match a pattern click-able:



The user can perform two additional actions on these when the mouse pointer hovers over a matched item:

- **Ctrl+click** Will try to open the item in a suitable program depending on what the type of the item is (see below).
- **right-click** Will add two entries to *The Context Menu*:
 - *Open link* - Same as Ctrl+click
 - *Copy address* - Copies the URL to the clipboard

In some types this may be converted into a different form depending on what the item represents.

¹ Although as you're reading this, I guess you figured that one out!

Here are the built-in formats understood:

URL	Note	Made up example, Don't use!
news://user@host:port/path		news://steve@news.example.org:1234/announce
telnet://user@host:port/path		telnet://steve@insecure.example.,org:1234
nnntp://user@host:port/path		nnntp://steve@news.example.org:1234/announce
file://user@host:port/path		file://steve@localhost/var/log/syslog file:///var/log/syslog
http://user@host:port/path	+ https://	http://steve@www.example.org/index.html
ftp://user@host:port/path	+ ftps://	ftp://steve@ftp.example.org/var/log/
webcal://user@host:port/path		webcal://steve@webcal.example.org/today
wwwhostname.domain:port/path		www-server.example.org/index.html www.example.org
ftphostname.domain:port/path		ftp-server.example.org/var/log/ ftp.example.org
VoIP		
callto:user:number@path		callto:steve:0123456789@not/sure/here
h323:user:number@path		h323:steve:0123456789@not/sure/here
sip:user:number@path		sip:steve:0123456789@not/sure/here
E-Mail		
mailto:name@host		mailto:steve@example.org
News		
news:name@host:port		news:steve@news.example.org:1234

These are just the ones built-in by default to Terminator. The *Plugins* can extend this further with a **URL Handler**, although strictly speaking it does not have to be a *URL* - as can be seen from some of the above - just a well defined pattern that can be matched.

1.3.3 Changing the current layout

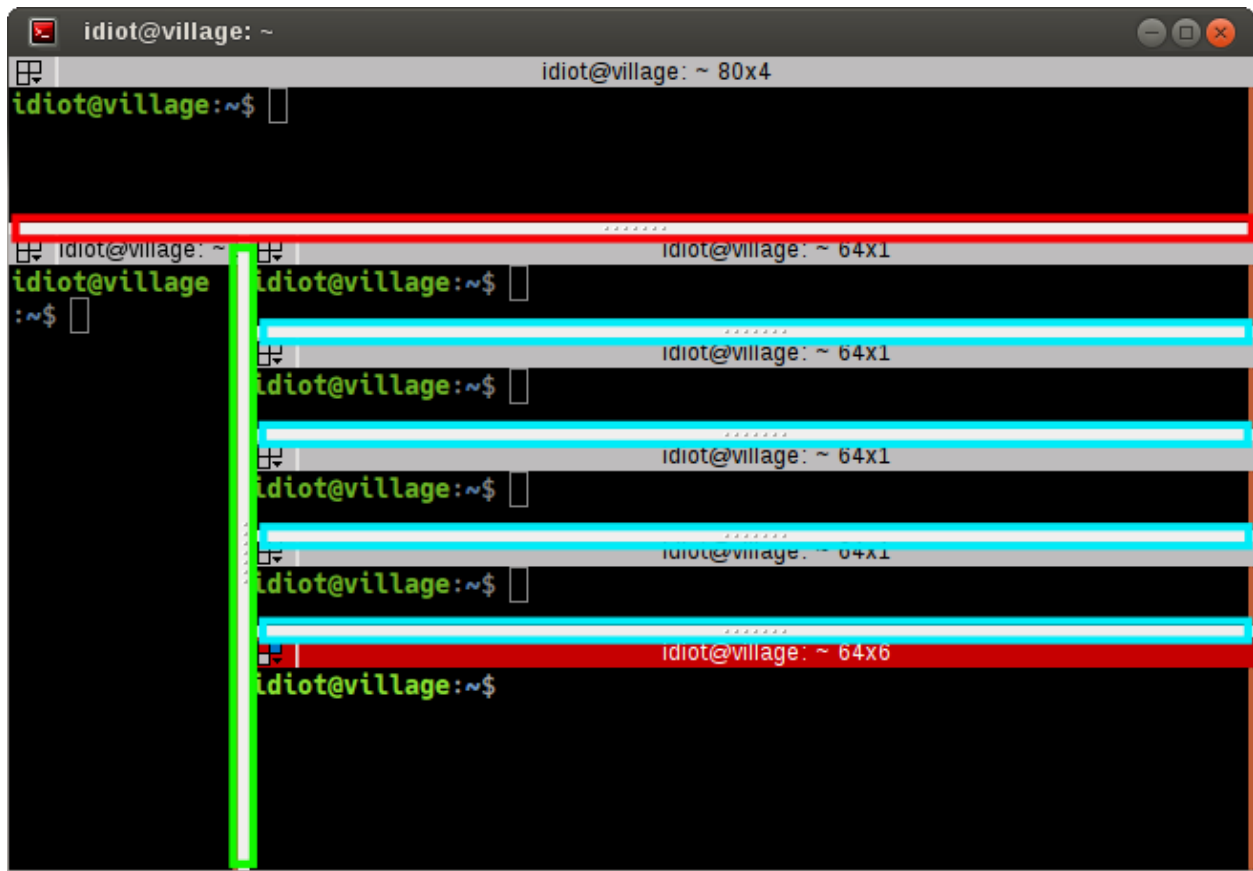
I've already used the term *layout* a few times in this page already. I should define what exactly is meant by a layout.

A layout describes the collection of windows in the current process, the tabs, and how the windows and tabs are divided up into terminals. It also includes the positions, dimensions, as well as other aspects related to how Terminator looks.

Besides the items in the *The Context Menu* there are four main methods to adjust the layout.

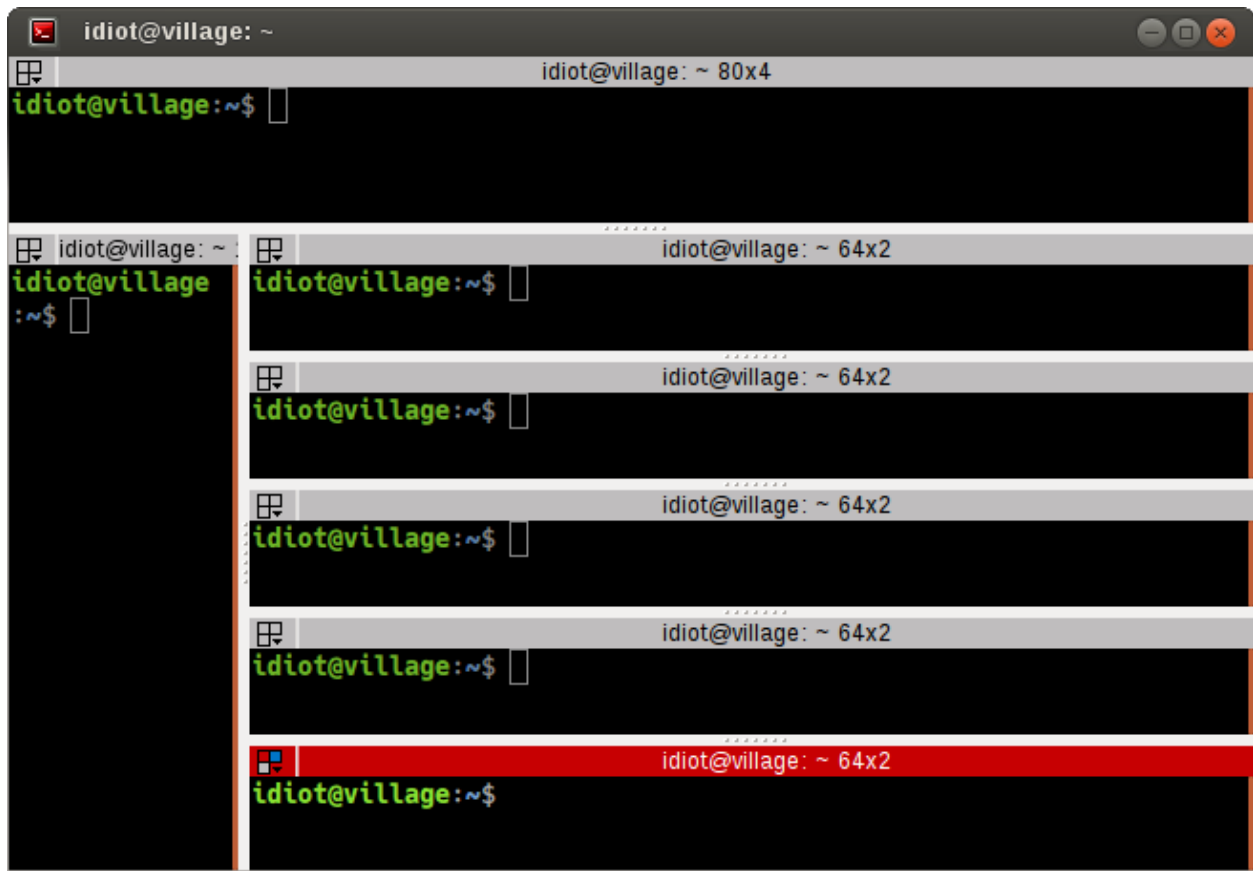
Using the splitters

So, by now you've probably made a few splits and used the mouse to drag them about, and you now have something resembling the following, minus the highlights:

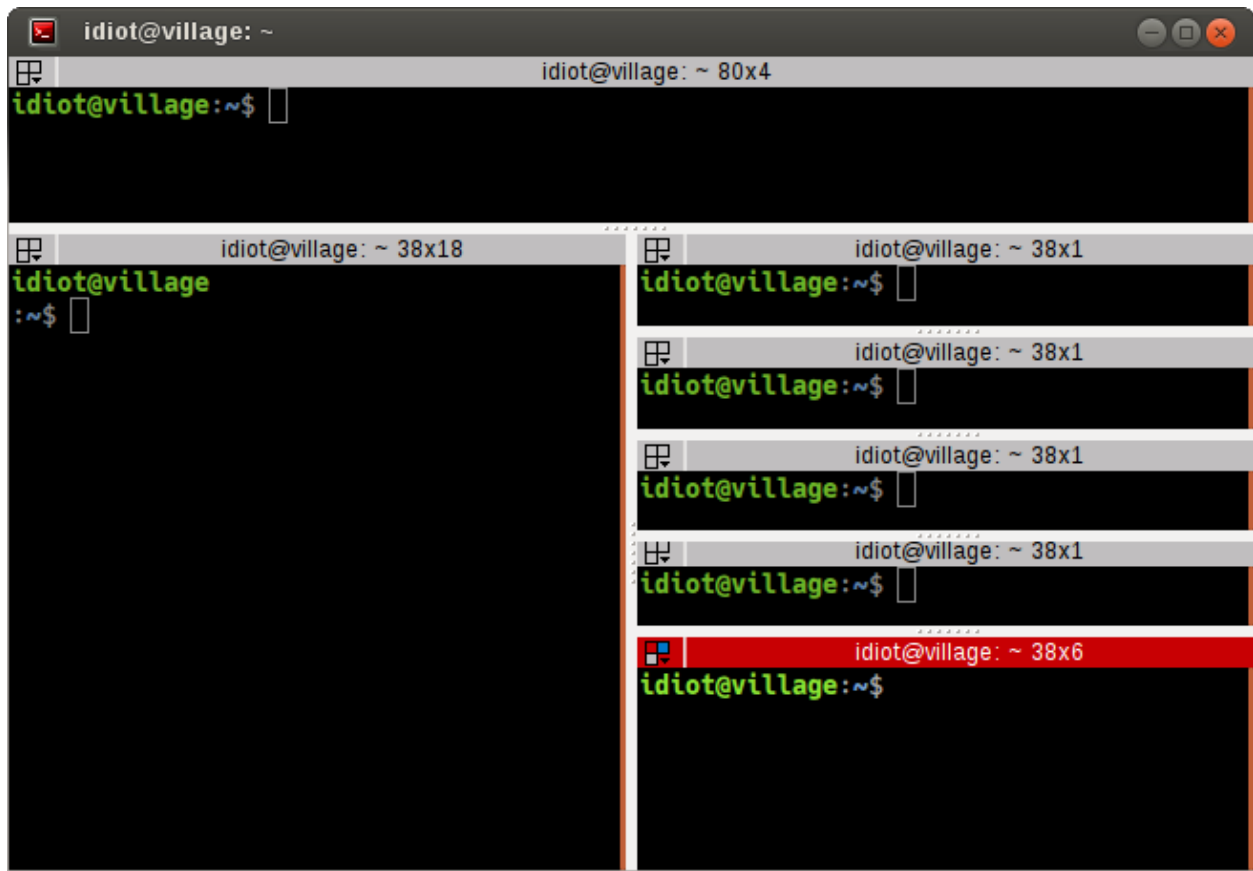


Terminator lets us *rebalance* the terminals, equally dividing the available space between the *siblings*. The different highlighting shows the siblings. The key thing to understand is that the blue splitters are considered siblings, which are *children* of the green *parent*. The green is itself a child of the red *parent*.

By double-clicking the splitter, the space will be divided evenly between the siblings. So, double-clicking any of the blue splitters will give:

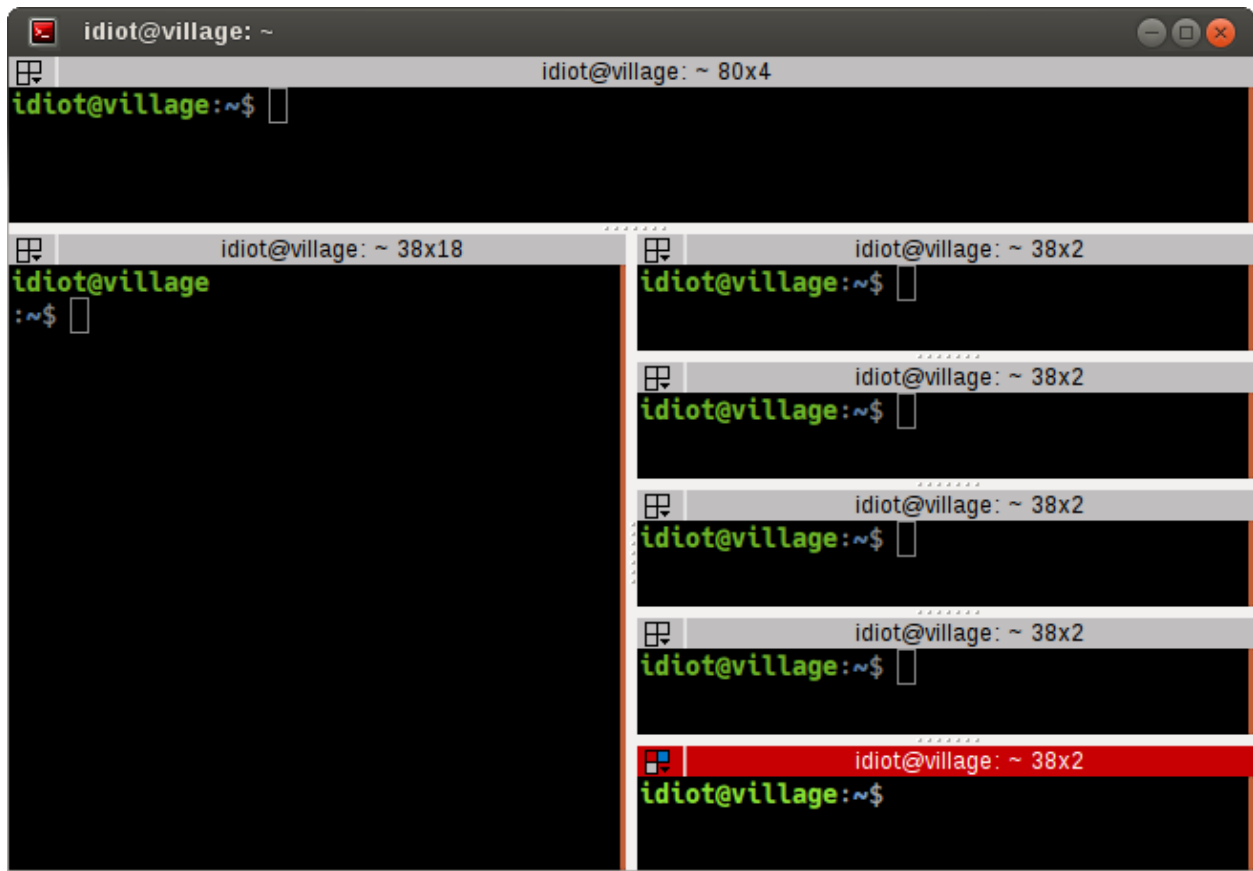


If instead we double-click on the green splitter, we get:

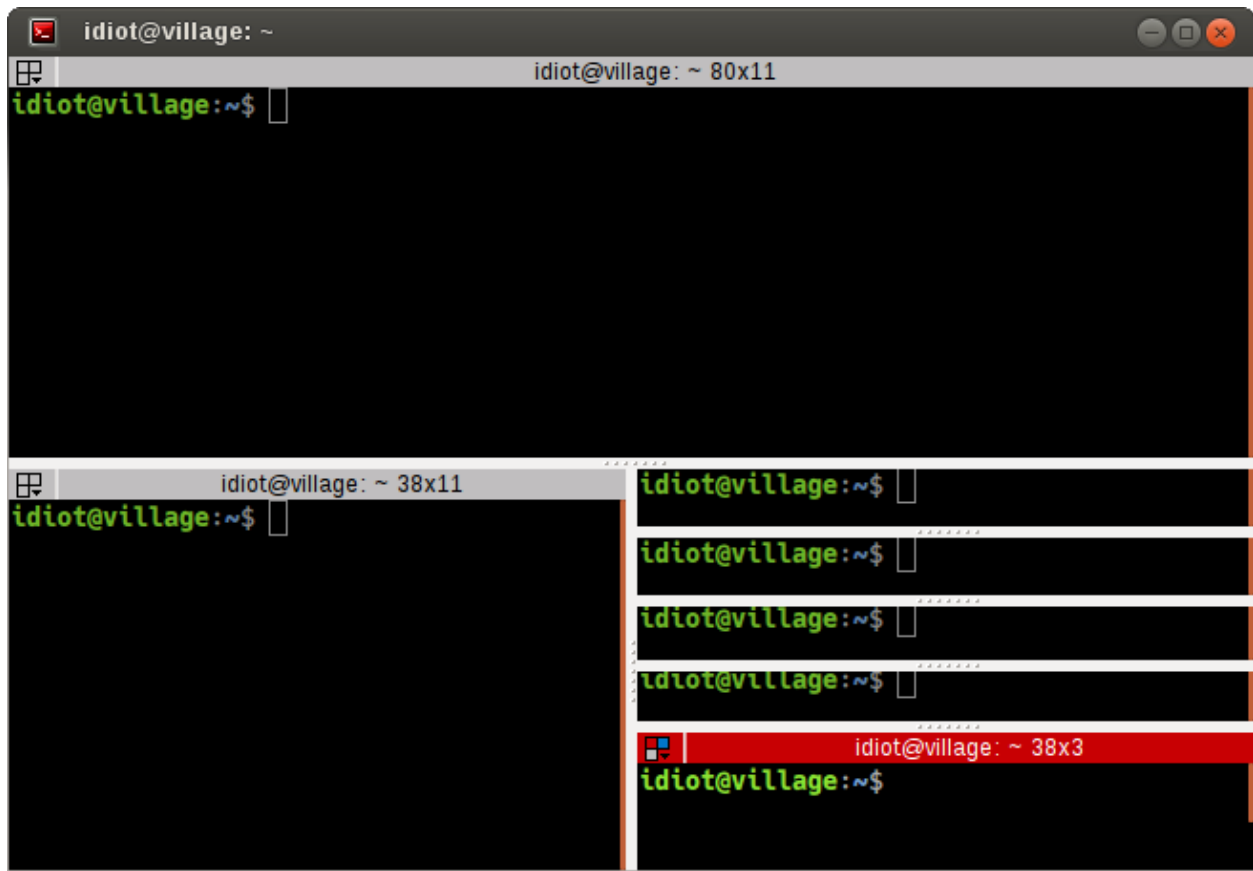


But there's more! We can use two modifier keys to rebalance more collections of siblings. Shift+double-click the splitter and all children, grandchildren, and so on, will be rebalanced. Super+double-click and all parents, grandparents, and so, on, will be re-balanced. You guessed it! Shift+Super+double-click and all visible terminals will be rebalanced. It will not affect terminals in other windows or tabs.

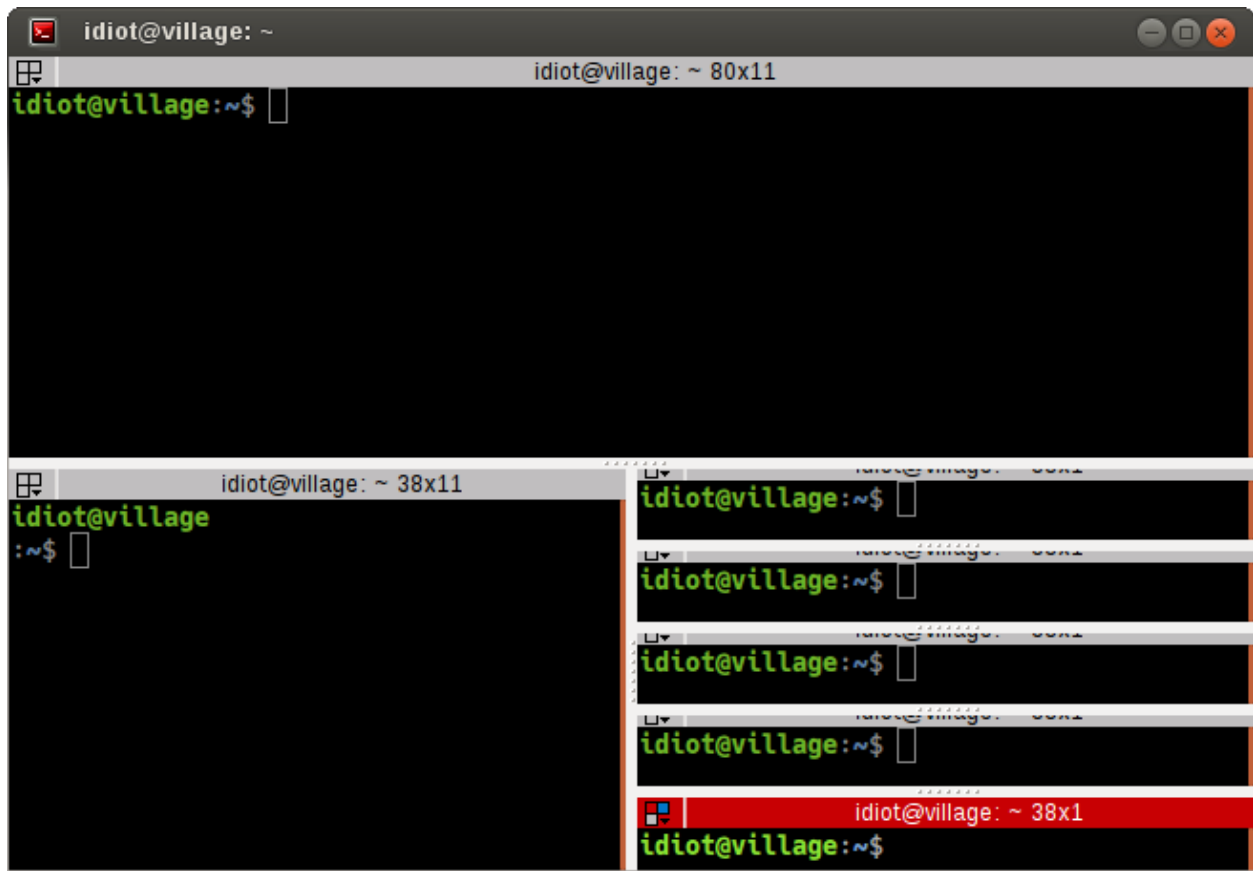
Shift+double-click on green:



Super+double-click on green:



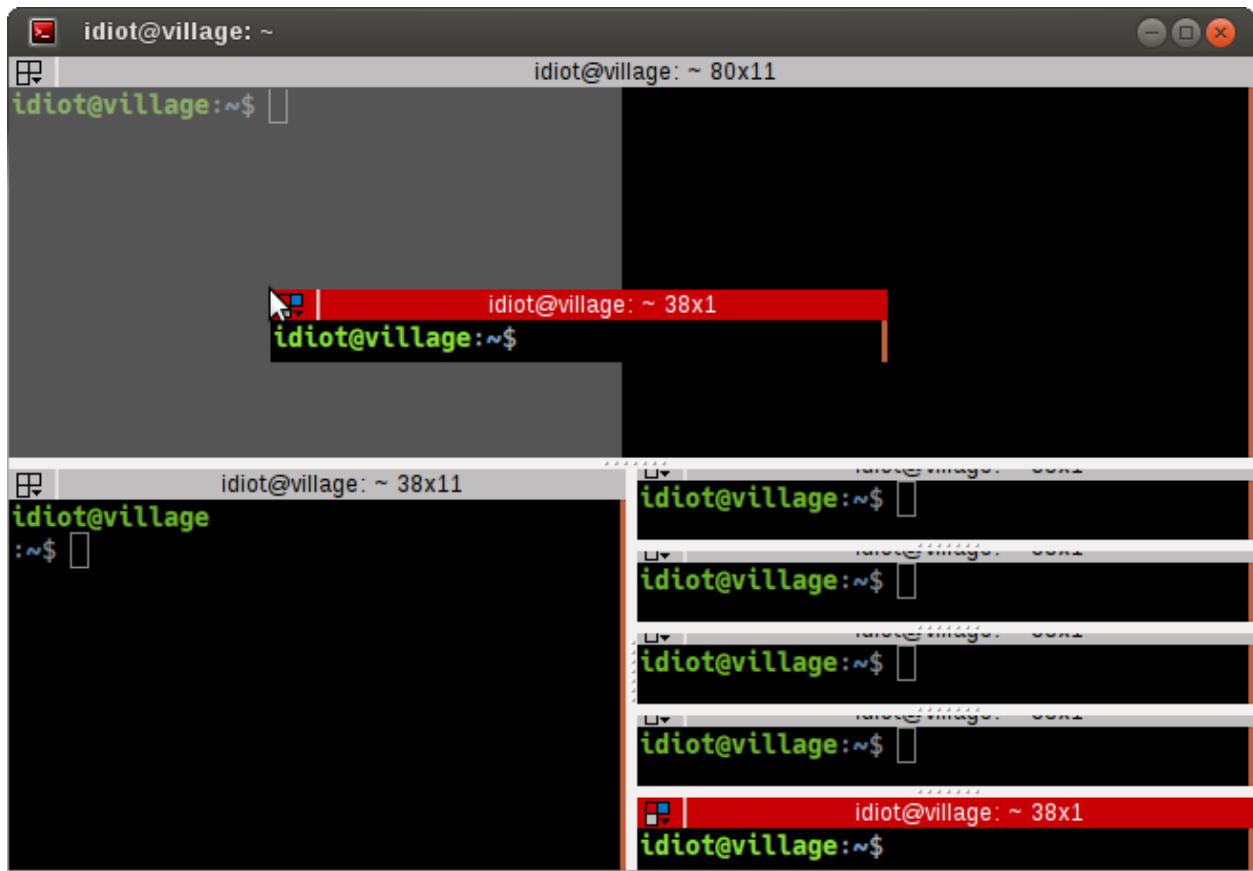
Shift+Super+double-click on green:



Note: Notice in the last two shots that you can shrink a terminal to a point where it is unusable or even completely hidden. We don't place an arbitrary minimum size. Some people want the ability to move the splitter all the way.

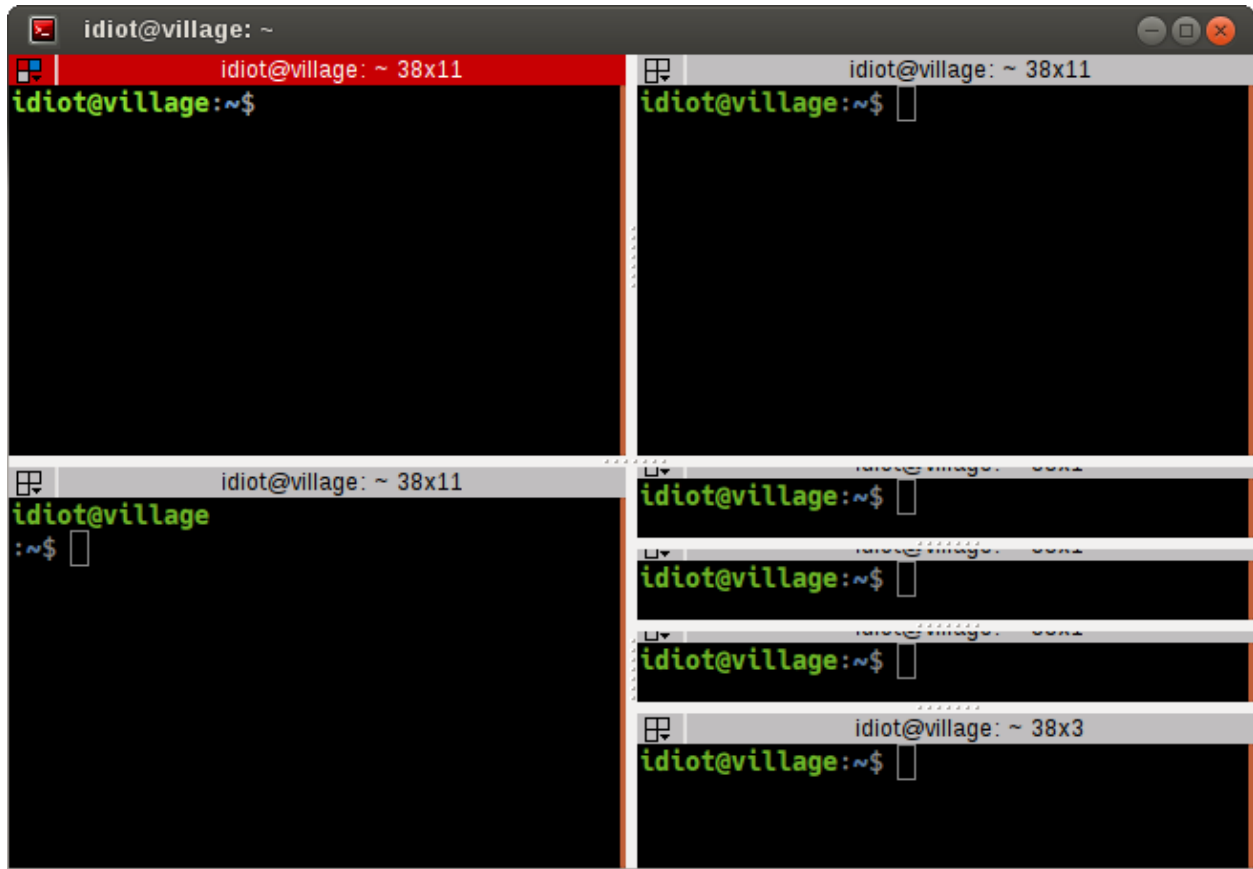
Dragging and dropping a terminal

There are two ways to drag a terminal from one location to another within the window. The simplest is to use the titlebar at the top of each terminal. Simply `click-drag`, and you will be able to hover over the other terminals and drop the dragged terminal to move it:



Here you can see a preview of the dragged terminal - scaled if large - and shading to show which area it will cover, which can be the top, bottom, left or right of an existing terminal.

The above action results in the following:



The other way to drag a terminal can be done from within the terminal with `Ctrl+right-click-drag`. With this method once you start the drag, you *must* release the `Ctrl` key *before* releasing the right-mouse-button. If you do not the drag will cancel.

You can drag between tabs by initiating a drag and hovering over the tab. Terminator will switch to the tab under the cursor, you can then drag to the desired position, and the terminal can be dropped.

You can also drag between Terminator windows *provided the windows are part of the same process*. By default all windows will be part of the same process. Windows will not be part of the same process if you deliberately turn off the [DBus](#) interface with the [Preferences](#) or the [Command line options](#) when starting Terminator up. [Layouts](#) are also currently isolated at a process level for technical reasons. - **Needs to be double checked and confirmed.** Since the work that got layouts working over DBus, this may now be wrong.

Using the keyboard

Of course, with Terminator being a terminal application, it makes sense to keep your hands on the keyboard as much as possible. So there are many shortcuts that you can tailor to your own preference. Here are the ones that will affect the layout:

Action	Options	Default Shortcut
New instance ²		Super+I
New window		Ctrl+Shift+I
New Tab		Ctrl+Shift+T
Split terminal	Horizontally, Vertically	Ctrl+Shift+O/E
Hide window ³		Ctrl+Alt+A
Close window		Ctrl+Shift+Q
Close terminal		Ctrl+Shift+W
Toggle fullscreen		F11
Resize terminal	Up, Down, Left, Right	Ctrl+Shift+<Arrow>
Rotate terminals	(Anti-)Clockwise	Super(+Shift)+R
Move Tab	Left, Right	Ctrl+Shift+PgUp/PgDn
Zoom terminal		Ctrl+Shift+Z
Maximise terminal		Ctrl+Shift+X

Dynamic layouts

Here the docs needs to be improved.

1.3.4 Resetting the terminal

There are two shortcuts available for fixing the terminal if it starts to misbehave.

Action	Default Shortcut
Reset	Ctrl+Shift+R
Reset + Clear	Ctrl+Shift+G

Note: Note that while *Reset* will only reset the current terminal state, the command *Reset + Clear* will also clear the terminal content, so be aware!

1.3.5 The scrollbar and scrollbar buffer

As already mentioned, there is a *Context Menu* item to toggle the scrollbar. There is also a shortcut listed here.

In addition there are shortcuts for moving up and down in the scrollbar buffer with more flexibility:

Action	Options	Default Shortcut
Toggle scrollbar		Ctrl+Shift+S
Page [VS]	Up, Down	Shift+PgUp/PgDn
X Lines [VS] [XL]	Up, Down	wheelup/wheel down
Page [TS]	Up, Down	
Half page [TS]	Up, Down	
Line [TS] [MS]	Up, Down	

² This is a separate process. As such, drag and drop will not work to or from this new window, or subsequent windows launched using the Ctrl+Shift+I while the focus is in the new instance.

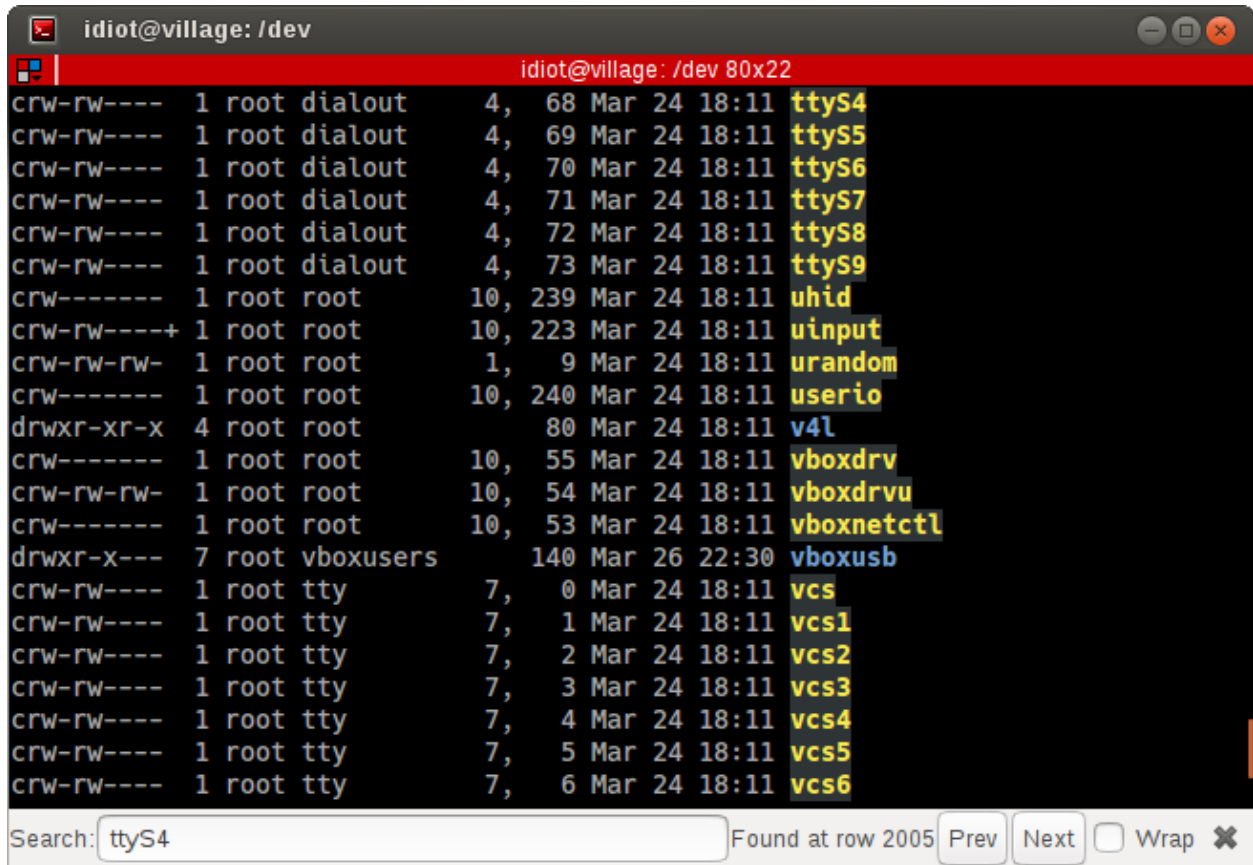
³ Hide window will currently only work on the first window of the first terminator instance that you start. That is because at present it binds the shortcut globally (it has to, or it cannot unhide) and this can only be done once. This may change in future.

1.3.6 Search the buffer

It is possible to search the buffer. The found string will be highlighted.

Action	Default Shortcut
Begin search	Ctrl+Shift+F

Resulting in a search bar at the bottom of the focused terminal:



This has buttons for moving back and forward through the results, as well as an option to wrap the search around.

1.3.7 Zooming the terminal

It is possible to zoom into and out of a terminal. There are also some modifiers to zoom more than just the current terminal.

Action	Default Shortcut
Target in ⁴	Ctrl++/wheelup
Target out	Ctrl+-/wheeldown
Target reset	Ctrl+0
+Receivers in	Ctrl+Shift+wheelup
+Receivers out	Ctrl+Shift+wheeldown
+Receivers reset	N/A (TBD, plus in/out)
All in	Ctrl+Super+wheelup
All out	Ctrl+Super+wheeldown
All reset	N/A (TBD, plus in/out)

1.3.8 Setting Titles

If you're anything like me, you've spent time clicking among the half a dozen different terminals in the taskbar, trying to find the right one. Or maybe for you it is with tabs.

In Terminator you can rename three things:

Edit	Mouse	Default Shortcut
Window title	N/A	Ctrl+Alt+W
Tab title	double-click tab	Ctrl+Alt+A
Terminal title	double-click titlebar	Ctrl+Alt+X

Additionally all three can be saved/loaded from a *layout*, or the window title can be set using a *command line option*.

1.3.9 Insert terminal number

These shortcuts let you enumerate your terminals. It can be handy if you need to login to a number of sequentially numbered machines. With multiple terminals the ordering may seem strange, but this is due to the nature of the splitting and the order in which the splits were performed.

Action	Default Shortcut
Insert terminal number	Super+1
Insert zero padded terminal number	Super+0

These actions can also be done from *The Grouping Menu*.

1.3.10 Next/Prev profile

It is possible to cycle back and forth through the available profiles that are defined in the *Profiles* tab of the *Preferences Window*, changing the behaviour and appearance of the current terminal.

Action	Default Shortcut
Next profile	
Previous profile	

⁴ Target terminal is the current terminal when using the keyboard shortcuts, or the terminal under the mouse when using the wheelup/wheeldown.

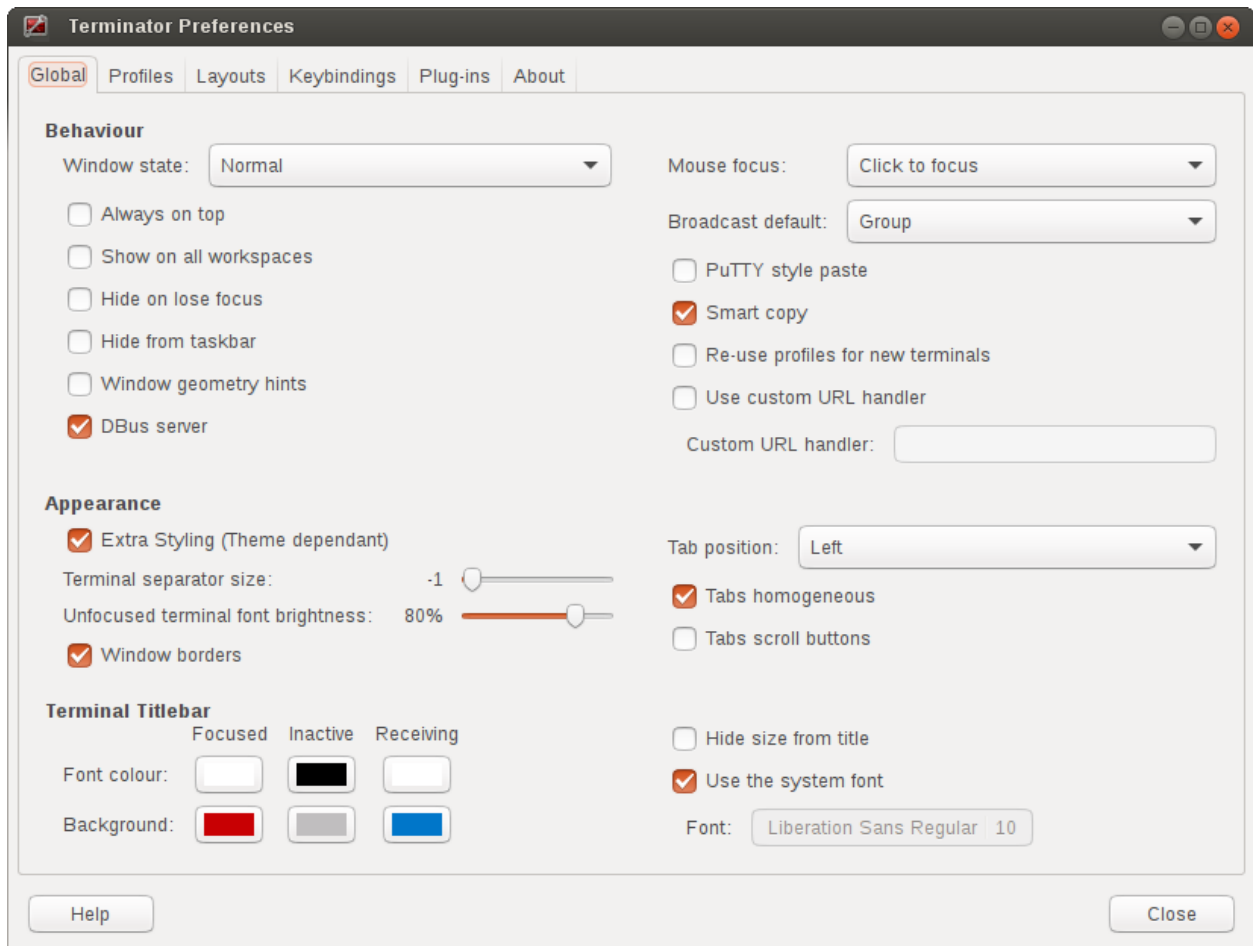
In both cases there is currently no default shortcut set. I'm not convinced they would be used often enough to warrant assigning them. For those that find it useful, the feature is there to be configured.



1.4 Preferences Window

Terminator is *highly* configurable, and automate-able, so the Preferences dialog is naturally quite extensive. It currently consists of six tabs. Let's work through them one by one.

1.4.1 Global



These settings are defaults, but some of them can be overridden by a options on the command-line, or within a layout. A number will also require a restart to take effect.

Behaviour

Window state (default: Normal)

This will determine what happens on startup normally.

- *Normal* - Window opens as normal.
- *Hidden* - Window does not open. Useful at login, so it is already available with a shortcut.
- *Maximised* - Window opens maximised in the standard window manager frame.
- *Fullscreen* - Window opens fullscreen with no window manager frame.

Always on top (default: off)

New windows attempt to remain on top, until deactivated in the window menu.

Show on all workspaces (default: off)

New windows will follow if you switch to a different virtual desktop, until deactivated in the window menu.

Hide on lose focus (default: off)

This is a quake console like feature, where the user want the window to vanish when clicking elsewhere.

Warning: This is rather buggy at the moment as it is very easy for the main window to lose focus and disappear.

Hide from taskbar (default: off)

The first window opened will not be displayed in the taskbar. Subsequent windows will show in the taskbar (bug?).

Window geometry hints (default: off)

If this is checked, then when resizing Terminator will attempt to step the sizing by the current font, and display a small box with the dimension of the window in characters.

Warning: If you have problems with Terminator windows shrinking in an uncontrollable way, then turning this option **off** will usually fix the issue. It is not clear why, but it seems Terminator and the window manager get into an argument over what size the window should be. Frankly this feature causes more trouble than it's worth. Don't be surprised if it gets removed at some point.

DBus server (default: on)

If a Terminator *DBus* server is not already on the session bus, try to start one.

Mouse focus (default: Click to focus)

By what method the mouse pointer sets the focus on a terminal.

- *GNOME Default* - Act as per the system settings.
- *Click to focus* - You must click with in a terminal to make it the focus.
- *Follow mouse pointer* - Moving the pointer over a terminal makes it the focus.

Broadcast default (default: Group)

Which broadcast mode should be selected at startup:

- *All* - All terminals receive keystrokes.

- *Group* - Only terminals in the same group as the current terminal receive keystrokes.
- *None* - Only the current terminal receives keystrokes.

PuTTY style paste (default: off)

Make the right mouse button operate like in PuTTY, so `right-click` will paste the Primary selection, and `middle-click` will open the *Context Menu*. (For ex-PuTTY users).

Smart copy (default: on)

If enabled and there is no selection, the shortcut is allowed to pass through. This is useful for overloading `Ctrl+C` to either copy a selection, or send the SIGINT to the current process if there is no selection. If not enabled the shortcut does not pass through at all, and the SIGINT does not get sent.

Note: For newbies SIGINT is the keyboard interrupt signal that will interrupt the program running in the foreground of a terminal.

Re-use profiles for new terminals (default: off)

When creating a new terminal with splitting or new tabs, if this is enabled, then the profile from the previously focussed terminal will also be used for the new one.

Use custom URL handler (default: off)

If this is enabled then `Ctrl+click` on a URL will try to use the command defined in *Custom URL handler* to open the link. If not enabled, Terminator will attempt to open the link with its internal logic. In order this attempts to open the URL using GTK, xdg-open, and lastly python's internal web browser support.

Custom URL handler (default: inactive, empty)

If active and set, then URL's will be passed as a command-line parameter to the given command.

Appearance

Extra Styling (Theme dependant) (default: on)

For themes we have the option to include some additional CSS code to make the window a bit prettier. For example under the Ubuntu Ambiance theme GNOME Terminal has custom tabs. In line with our unofficial policy of following gnome-terminal, I have replicated that customisation for Terminator. Some may prefer to use the unadulterated standard tabs, so using this option the extra styling can be turned off.

Terminator separator size (default: -1)

This is the width in pixels, and can range from -1 to 20. The value of -1 will take the default size from the system theme.

Note: Making this too small will make grabbing the splitters quite difficult, as we remove the oversized splitter handles some themes provide because it interferes with mouse selection of text.

Unfocused terminal font brightness (default: 80%)

Terminals that do not currently have the focus will can be dimmed to aid focus. The value can range from 0% (invisible) to 100% (full brightness)

Window borders (default: on)

The window manager frame is removed from your windows.

Tab position (default: Top)

Where the tabs will be located within the window

- *Top*
- *Bottom*
- *Left*
- *Right*
- *Hidden* - Tabs still work, you just can't see them.

Tabs homogeneous (default: on)

Warning: This option was removed during the port to GTK 3, and has no effect, apart from giving access to the *Tab scroll buttons* option.

It used to give the choice between tabs of uniform and non-uniform width.

Tabs scroll buttons (default: off)

When there are more tabs than can fit within the window buttons will be drawn for moving left and right.

Warning: If the tab scroll buttons are turned off and you open an extreme number of tabs in a single window, an undesirable behaviour occurs. Once the tabs reduce to the minimum possible size the window is forced wider to accomodate additional tabs. It is not immediately obvious as to what the correct response is in this situation.

Terminal Titlebar

There is a table of the colours for the titlebars on the left. These are modelled on those used in a utility I used to use called ClusTerm. The three sets (Focused, Inactive and Receiving) will make more sense after reading the section about *The Grouping Menu*.

	Focused	Inactive	Receiving
Font colour	#FFFFFF	#000000	#FFFFFF
Background	#C80003	#C0BEBF	#0076C9

Hide size from title (default: off)

At the end of the label in the titlebar the size of the terminal is given in characters, i.e. (80x24). Enabling this item will disable the size text.

Use the system font (default: on)

By default the system defined proportional font will be used for the text in the titlebar. Turning this off allows you to use a custom font.

Font (default: inactive, system proportional font)

If active and set, then the custom font to be used in the titlebar.

Title bar at bottom (default: off)

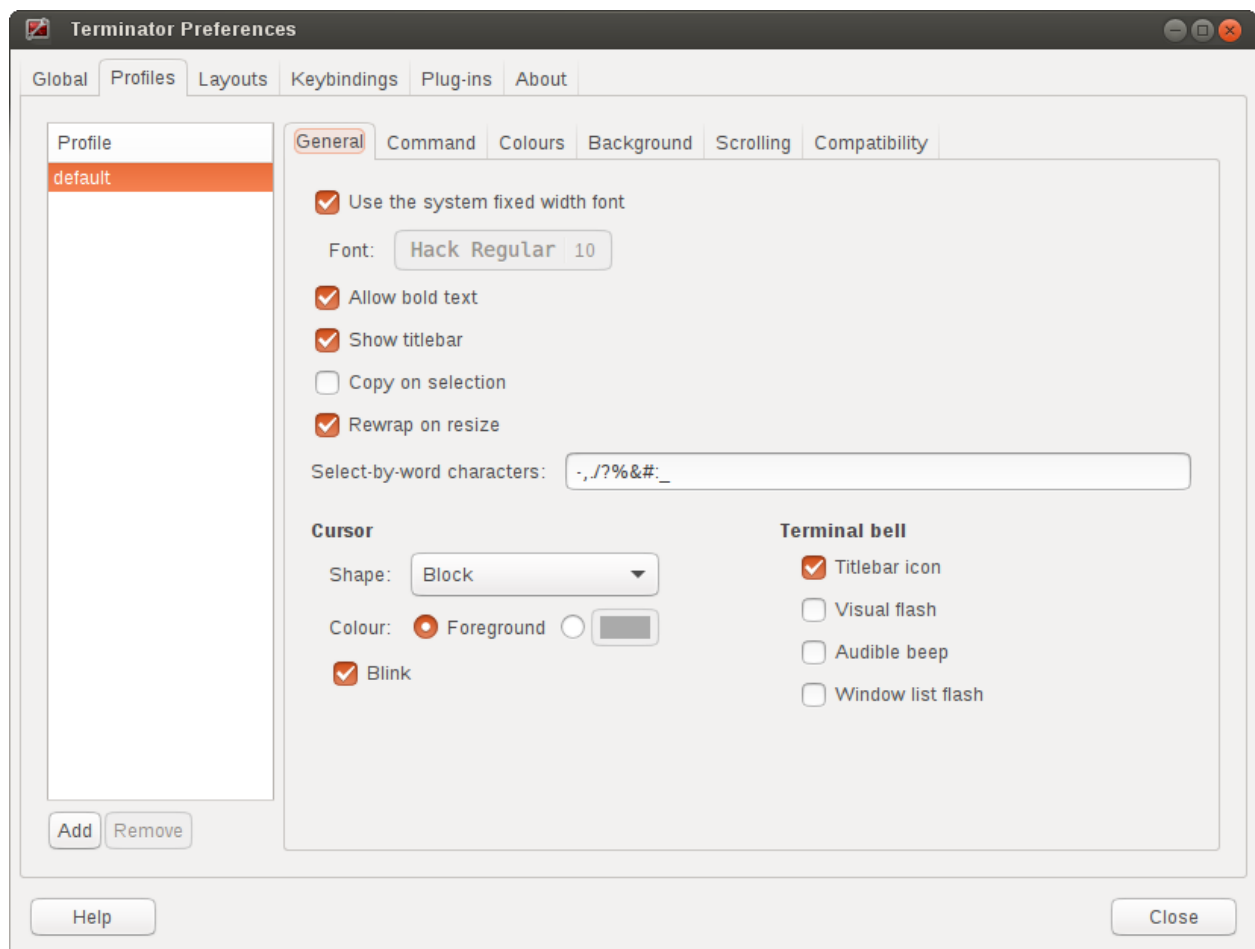
If set the terminal title bar is placed at the bottom of the terminal.

1.4.2 Profiles

You should already be familiar with the sub-tabs from GNOME Terminal, and Terminator's are modelled on those available in GNOME Terminal where it makes sense, and give much of the same functionality.

Below we will go through each pane, and highlight and explain differences between Terminator and GNOME Terminal.

General



One key difference is that we have a sidebar to the left listing the available Profiles, as opposed to GNOME Terminal, where the list is a separate window launched from the menu bar. This also means a few of the widgets, like the profile name, are not needed.

Use the system fixed width font (default: on)

By default the system defined proportional font will be used for the text in the terminal. Turning this off allows you to use a custom font.

Font (inactive, system fixed width font)

If active and set, then the custom font to be used in the terminal.

Allow bold text (default: on)

Allows you to disable the use of bold fonts in the terminal.

Show titlebar (default: on)

The titlebar strip across the top of each terminal can be turned off.

Copy on selection (default: off)

This puts the selection into the copy/paste buffer, as well as being available on middle-click.

Rewrap on resize (default: on)

This will cause longer lines to rewrap when a terminals width changes.

Note: Larger or infinite scrollbar buffers may become slow when this option is enabled.

Select-by-word characters (default: -, . / ? % & # : _)

Using `double-click` to select text will use this pattern to define what characters are considered part of the word.

Cursor

Shape (default: Block)

Set the cursor shape

- *Block* - Solid rectangle.
- *Underline* - Single pixel tall horizontal line.
- *I-Beam* - Single pixel wide vertical line.

Colour (default: Foreground)

The colour of the cursor. A radio option of Foreground will use whatever the foreground is defined as for regular text, as set in the Colours tab. Alternatively a custom colour can be chosen using the colour swatch.

Note: Foreground uses xor'ing so the text under the cursor is always clear. Xor'ing is not used with a custom colour. This means that if the colour of the character under the cursor is similar to the colour chosen, then it can be difficult to discern what that character is. The following option can help with this.

Blink (default: on)

Whether the cursor blinks on and off.

Terminal bell

Titlebar icon (default: on)

On the right side of the titlebar a small light-bulb icon will be displayed for a few seconds.

Visual flash (default: off)

The terminal area will briefly flash.

Audible beep (default: off)

The normal system beep noise as defined in system settings.

Window list flash (default: off)

This will set the urgent flag on the window in the taskbar. The actual effect will be taskbar dependant.

Not in Terminator

Profile name

Our profiles names are in the sidebar to the left.

Profile ID

Ummm... OK, I have no idea what GNOME Terminal uses this for.

Show menubar by default in new terminals

Terminator doesn't use a traditional menu bar. This has been removed in new versions of GNOME Terminal.

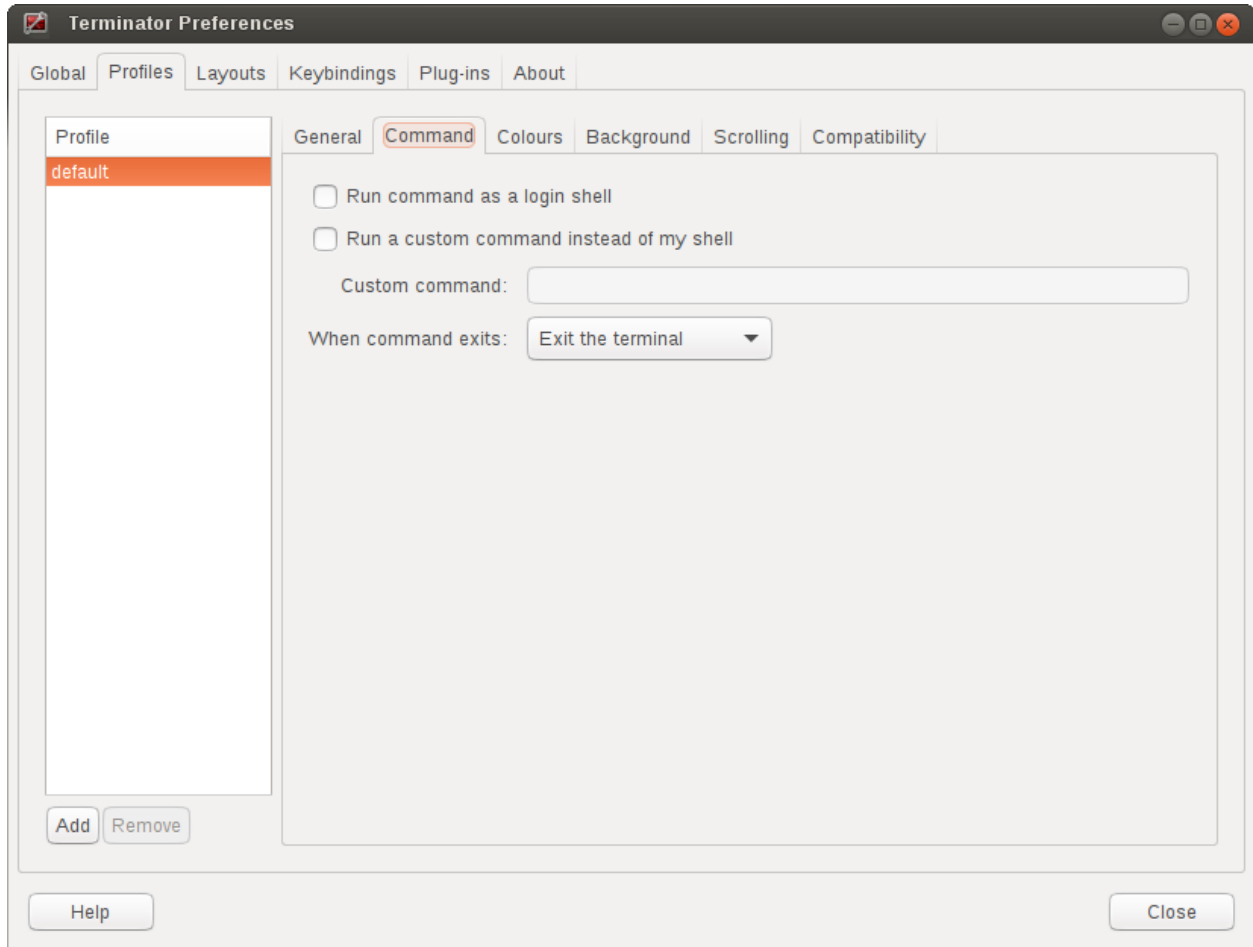
Terminal bell

Terminator has more options, so has four separate options in their own grouping. This item in GNOME Terminal is the same as *Audible beep* defined above.

Initial terminal size

Terminator handles window sizes within *Layouts*, or with *Command line options*.

Command



Run commands as a login shell (default: off)

Force the command to run as a login shell.

Run a custom command instead of my shell (default: off)

Enable the use of a custom command instead of the users default shell.

Custom command (default: inactive, empty)

If enabled and set, the users default shell will be replaced with the command specified here.

Note: If you place an entry here note that there is no `bash` or other shell underneath it. When the command ends, there is no chance to drop to a shell or other program. This can be worked around by using the shell line separator `;` and a following `bash` command.

Warning: Running a non-bash program as a command *can* lead to unexpected results. Some programs behaviour depends on having a full, interactive shell underlying the program. An example would be `mutt`. Run standalone, at startup it will begin with all threads expanded. Using:

```
bash -c mutt
```

will also not work, as this is a non-interactive session. Instead make the session interactive with:

```
bash -ic mutt
```

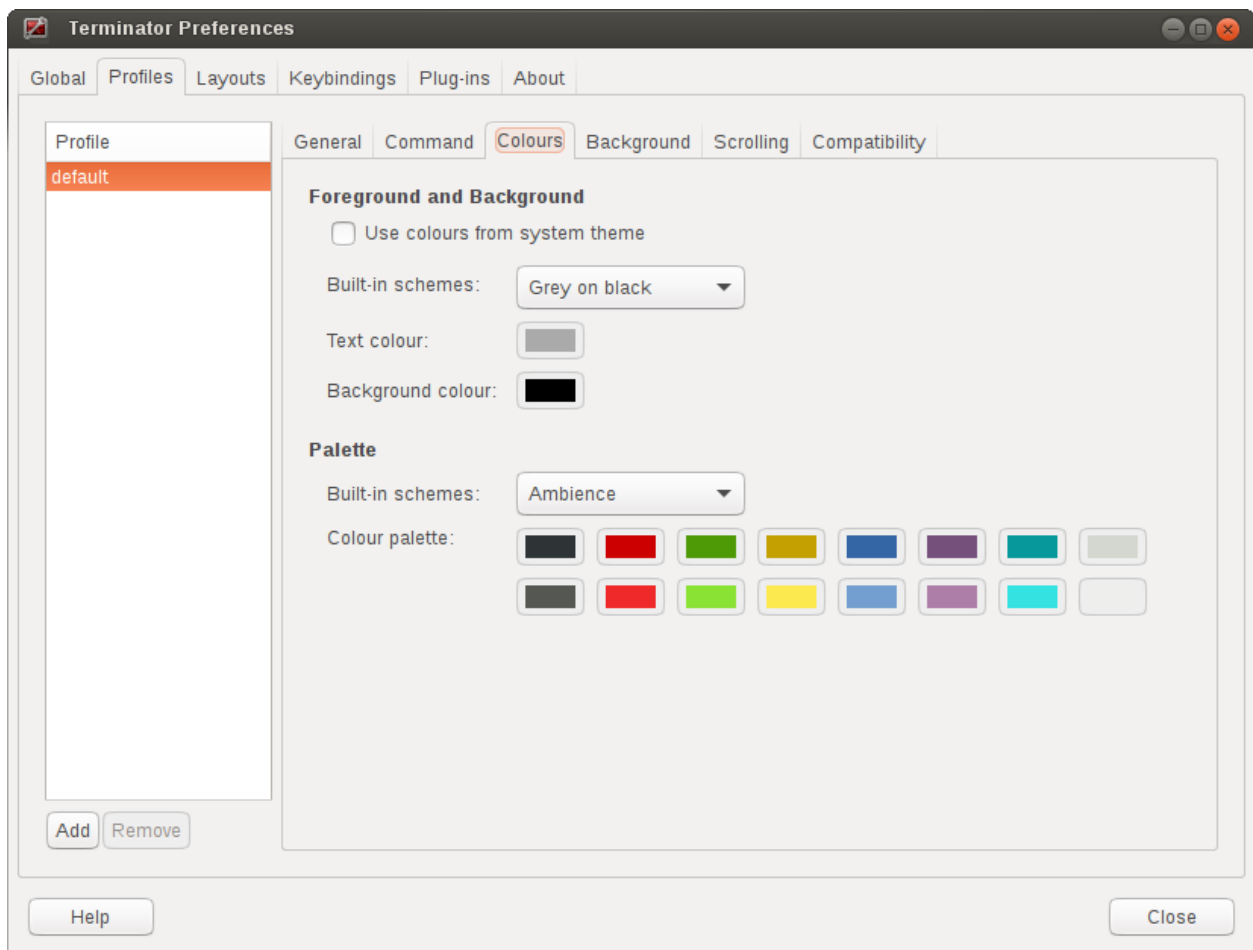
When command exits (default: Exit the terminal)

When the running command exits (default or custom) what action should be taken.

- *Exit the terminal* - Terminal closes, causing layout to adjust.
- *Restart the command* - Original command restarts immediately.
- *Hold the terminal open* - The terminal and scrollback will remain visible and accessible until the user explicitly closes the terminal, or closes the window.

Warning: If you are using *Restart the command* and your command is broken and exits immediately, then you can end up in a resource hungry loop.

Colours



There seems to be some mild quirks and differences (palettes available or selected from the system theme) between Terminator and GNOME Terminal.

Foreground and Background

Use colours from system theme (default: off)

Use colours as defined in the system theme. These are requested from the underlying VTE widget.

Built-in schemes (default: Grey on black)

Pick a primary colour combination for foreground and background. Again there are differences between Terminator and GNOME Terminal.

The list for GNOME Terminal seems to be dynamic and vary depending on the system, with the addition of *Custom* which allows setting the colours as desired. Terminator has a number of schemes hard coded. (This may see improvement at some point.)

Text colour (default: inactive, #AAAAAA)

If the *Built-in schemes* is set to *Custom* the text colour can be set here.

Background colour (default: inactive, #000000)

If the *Built-in schemes* is set to *Custom* the background colour can be set here.

Palette

Built-in schemes (default: Ambience)

A predefined colour palette can be selected. The same text applies as used for the *Built in schemes* option under *Foreground and Background*.

Colour palette (default: inactive)

If the Palette's *Built-in schemes* is set to custom, a set of colour swatches are used to configure the 16 primary colours of the shell palette.

Not in Terminator

Bold colour

In theory nothing is stopping us implementing this, it just doesn't appear to have ever been added.

Same as text colour

In truth, I'm not exactly sure what this does, but at a guess, the user can force bold to be drawn in the same colour as the foreground text.

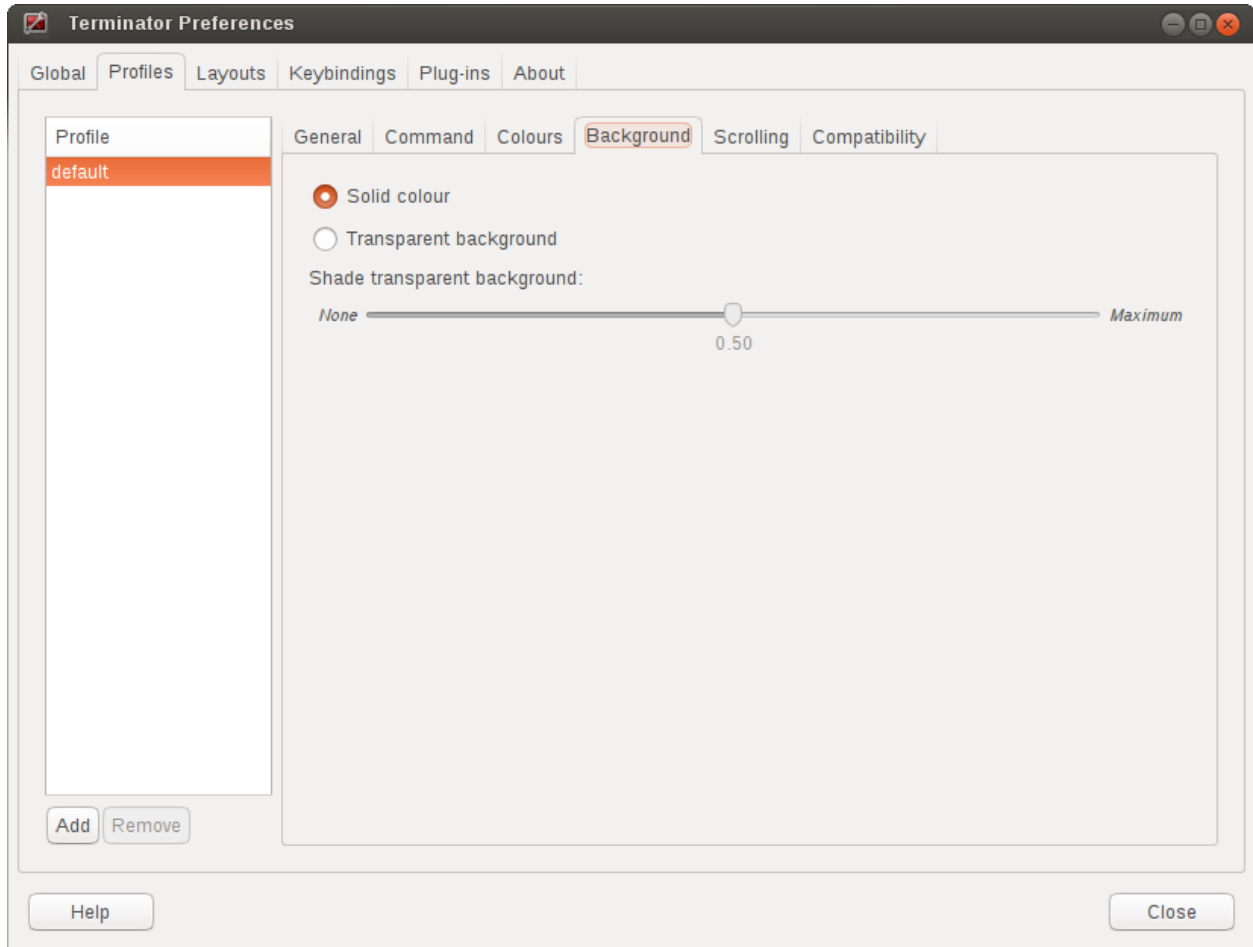
Use transparent background

Our transparency has a tab all to itself.

Use transparency from system theme

Not sure which setting GNOME Terminal gets this from.

Background



Solid colour (default: active)

Background of terminal is set to the solid colour set in previous *Colours* tab.

Transparent background (default: inactive)

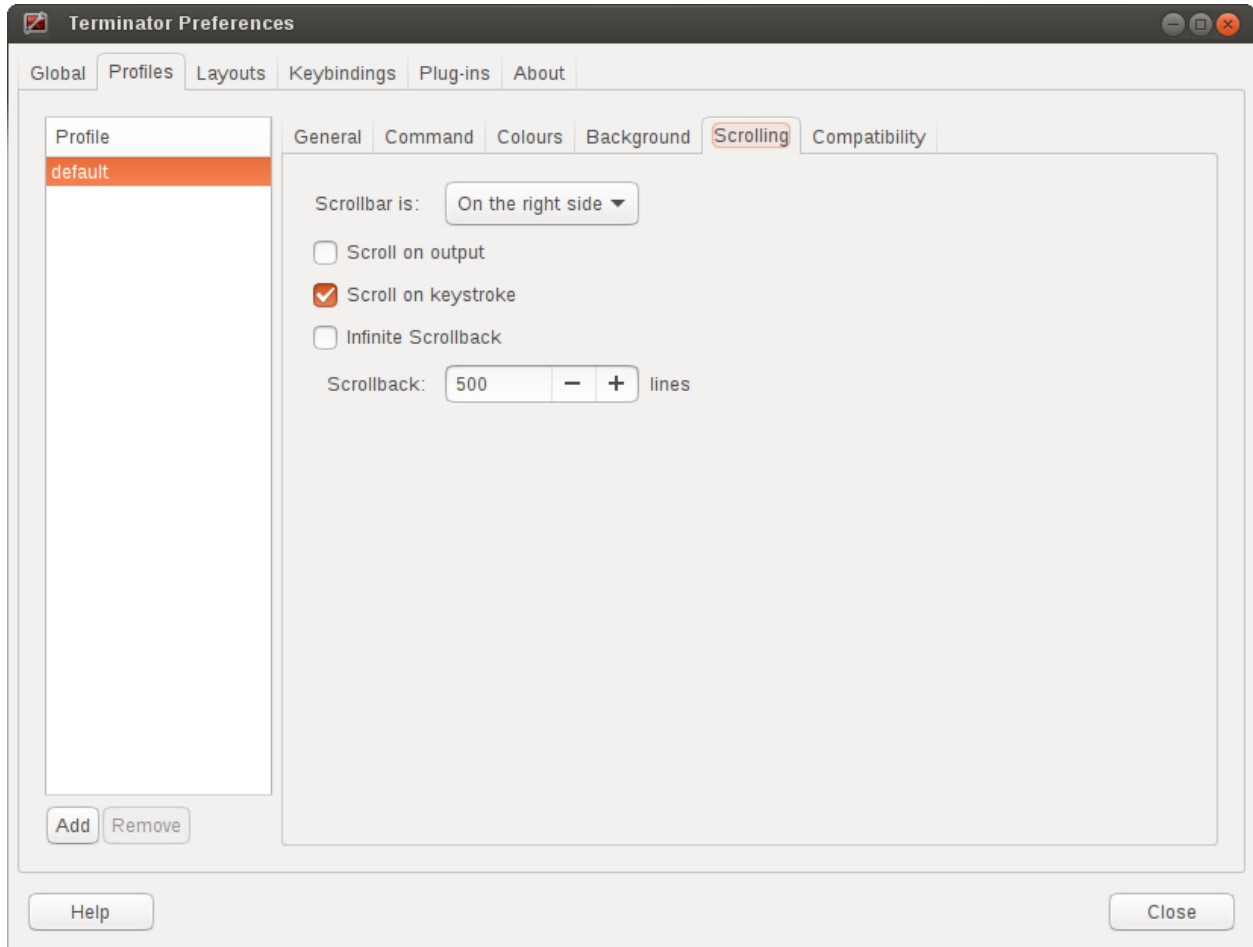
This will attempt true transparency where the windows below are partially visible through the terminal.

Note: This option requires a compositing desktop.

Shade transparent background (default: 0.5)

For *Transparent background* this is how much the solid colour should be blended in, giving a tinting effect.

Scrolling



Scrollbar is (default: On the right side)

If and where the scrollbar should appear.

- *On the left side*
- *On the right side*
- *Disabled*

Scroll on output (default: off)

Moves terminal to end of scrollbar buffer when any output occurs.

Scroll on keystroke (default: on)

Moves terminal to end of scrollbar buffer when any keypress occurs.

Infinite Scrollback (default: off)

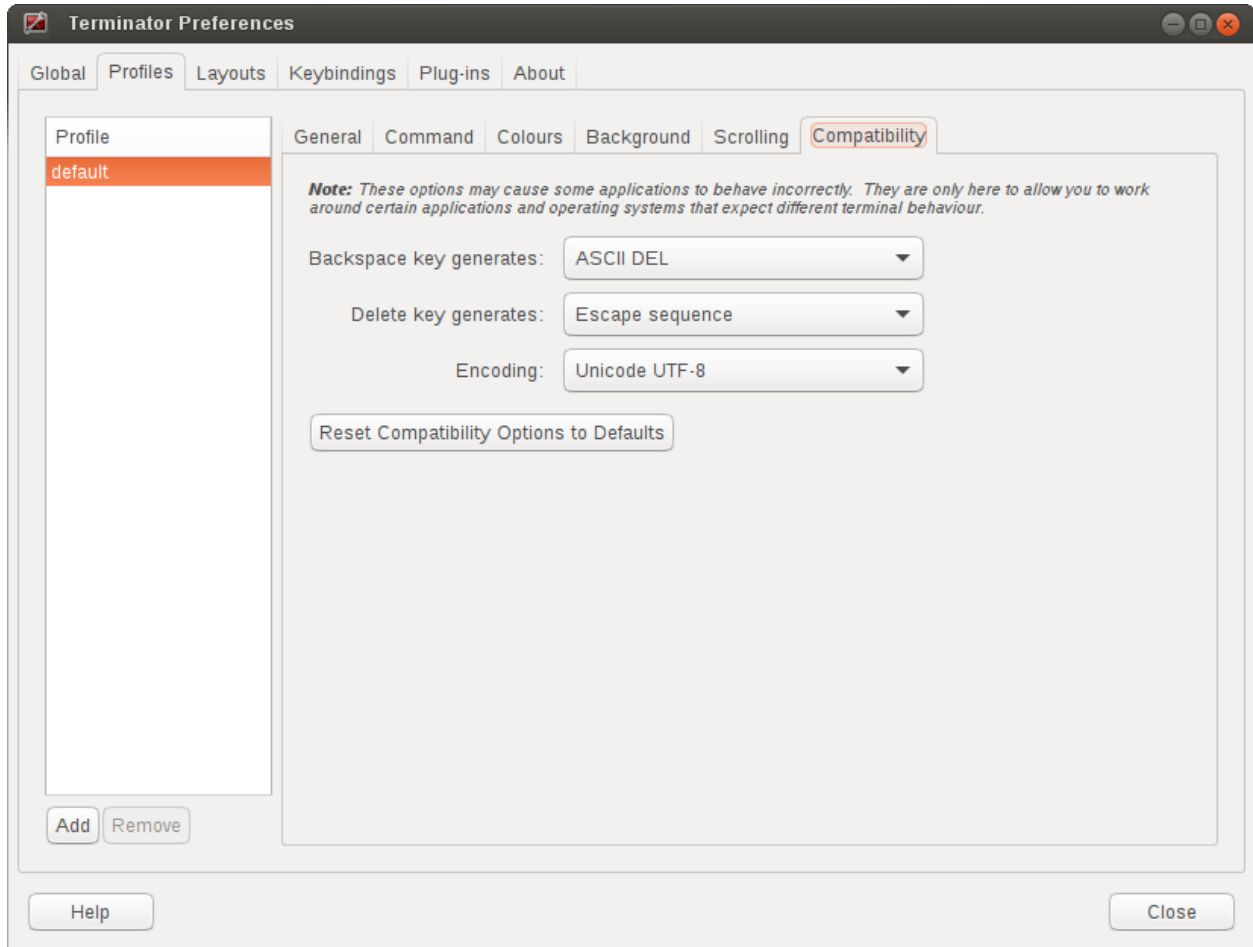
Lines are never discarded, and all lines since the session began are available.

Note: Data is placed onto the disk by the underlying VTE component, so even after a long time, the memory footprint and performance of Terminator should be OK.

Scrollback (default: 500 lines)

How many lines to keep before discarding.

Compatibility



Backspace key generates (default: ASCII DEL)

Change behaviour of the Backspace key.

- *Automatic*
- *Control-H*
- *ASCII DEL*
- *Escape sequence*

Delete key generates (default: Escape sequence)

Change behaviour of the Delete key.

- *Automatic*
- *Control-H*
- *ASCII DEL*
- *Escape sequence*

Encoding (default: Unicode UTF-8)

Choose the default encoding method used from a long list of available encodings.

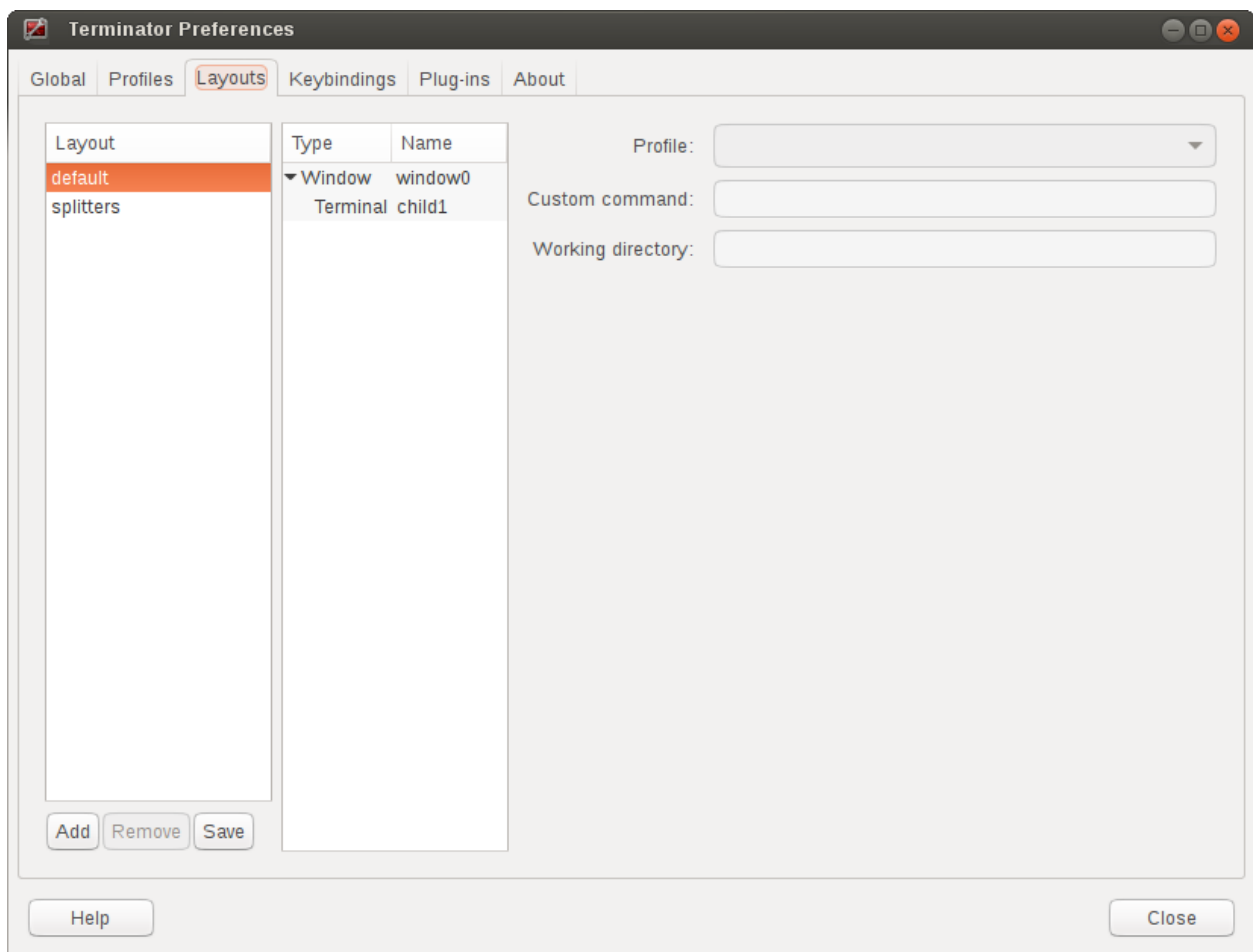
Reset Compatibility Options to Defaults

Sets the previous items back to their defaults.

Not in Terminator**Ambiguous-width characters**

Not really too sure what this does.

1.4.3 Layouts



Layouts are the primary means for saving collections of windows, tabs, and terminals. The use and flexibility of layouts is covered in [Layouts and the Layout Launcher](#). Here we will cover the bare minimum to understand the configuration options.

In the list to the left is the saved layouts, with three buttons below:

- *Add* - Creates a new layout from the current windows, tabs and terminals, and saves them with a new name.
- *Remove* - Delete the selected layout

- *Save* - Update the selected layout with the current windows, tabs, and terminals.

Warning: You do not need to use the save button when changing the options in the controls on the right.

If you do, you *will* lose the *Custom command* and *Working directory* settings for all terminals in this layout. It will also replace the *saved* layout with the *current* layout. This means your windows may now be the wrong size, or in the wrong position.

Once a layout is highlighted, its name can be changed by clicking it again.

In the central list is a tree showing the structure of the selected layout. When highlighting an entry of type Terminal, the controls on the right become enabled, and can be changed.

Profile

The profile used by the select terminal as listed in the *Profiles* tab.

Custom command

Override the command run in the terminal, same as in a profile, but this one has a higher priority. If empty, it will run the command in the profile, or the default user shell.

Note: If you place an entry here note that there is no `bash` or other shell underneath it.

If your application needs a shell (i.e. `mutt` misbehaves if run without `bash`) then run your command inside a `bash` session with:

```
bash -ic <command>
```

When the command ends, there is no chance to drop to a shell or other program. This can be worked around by using the shell line separator `;` and a following `bash` command:

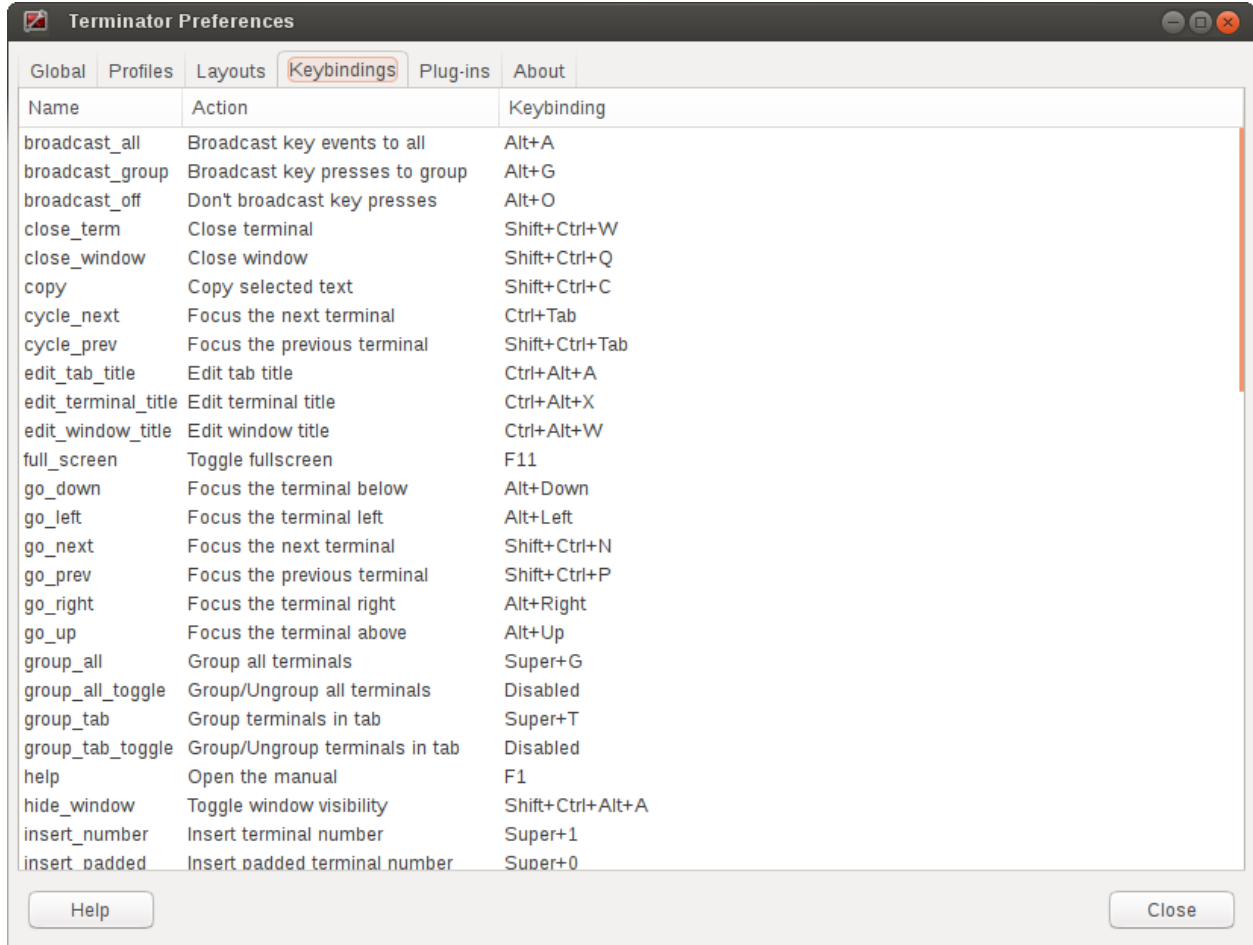
```
bash -ic <command>; bash
```

However, note that the second `bash` will have no connection to the details of the `bash` the command ran under. This means no environment variables, or return codes are carried over.

Working directory

Whatever command is run (from layout, profile, or user default) it will be executed with this entry as the working path. If empty the default working directory is used, which is either where Terminator was launched from, or the users home directory.

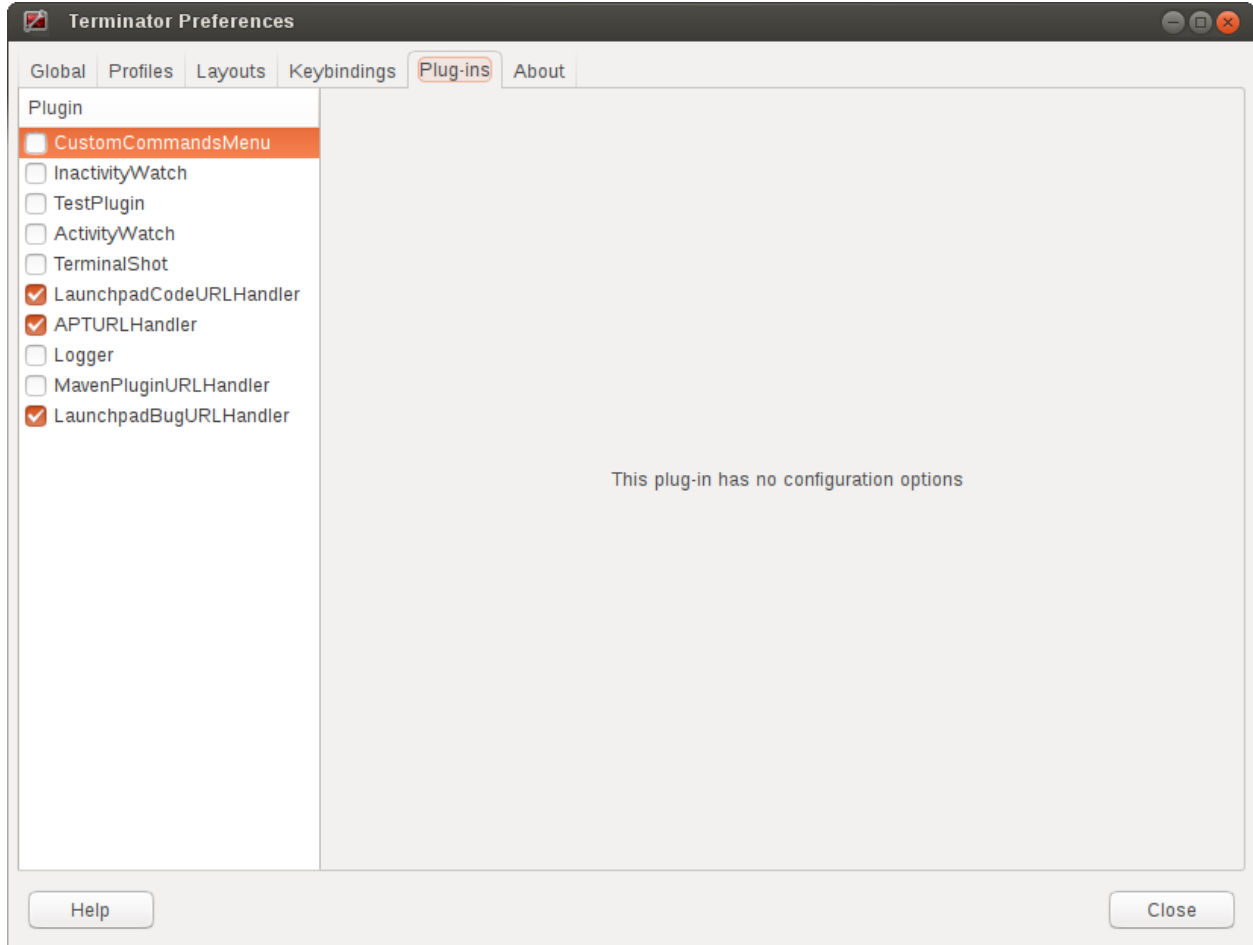
1.4.4 Keybindings



This is a list of all available keyboard shortcuts in the application.

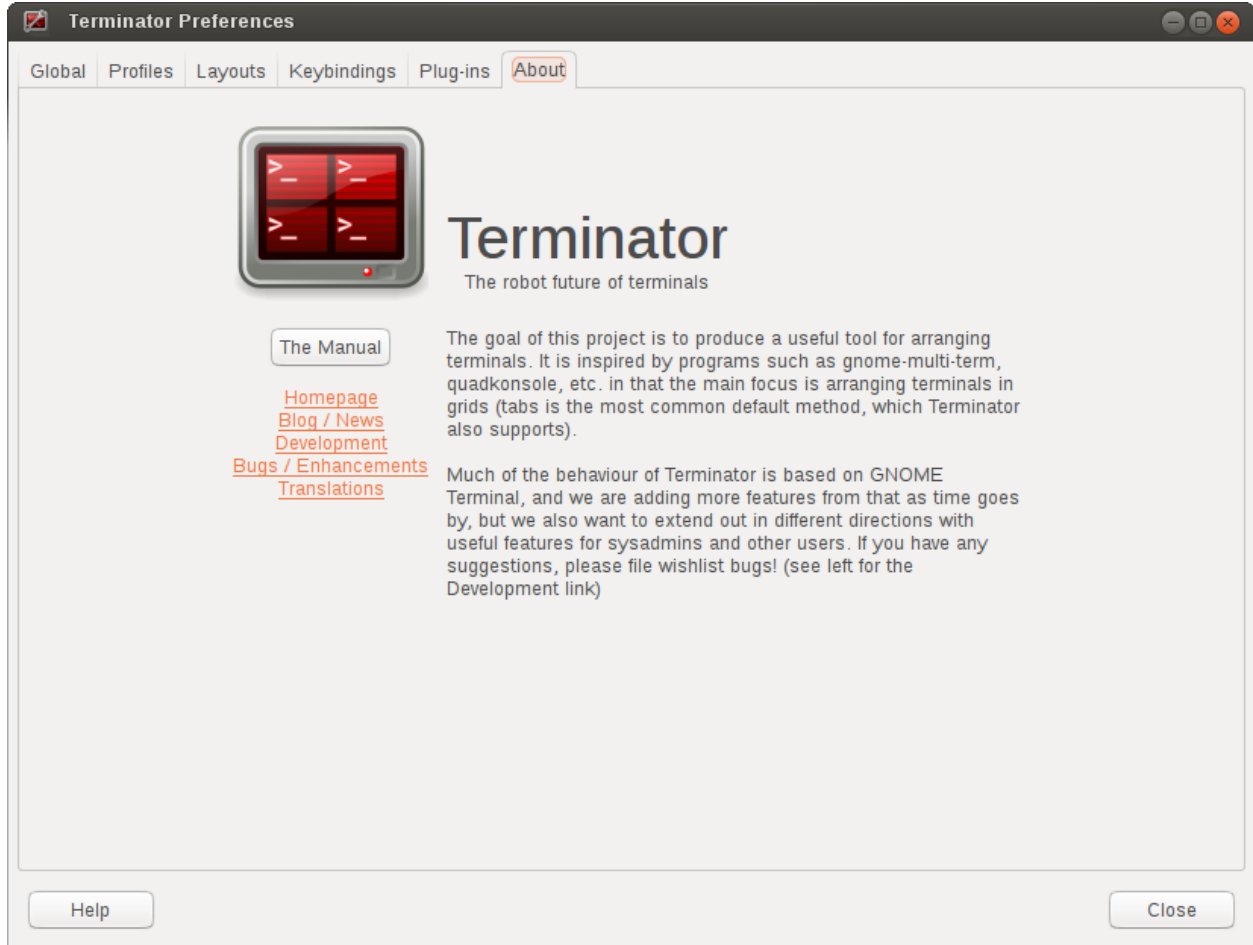
To change a keybinding, first highlight the entry you wish to change. Next click on the *Keybinding* column again. The entry should change to **New accelerator...** Simply perform the shortcut you wish to set. If you change your mind use `Esc` (Escape) key to revert back to the existing shortcut. If you wish to delete a shortcut, use the `BkSp` key (Backspace, `←`, or `⌫` depending on your keyboard).

1.4.5 Plugins



Here you will find a list of available plugins, and whether they are enabled or not. Plugins are covered in more detail in *Plugins*.

1.4.6 About



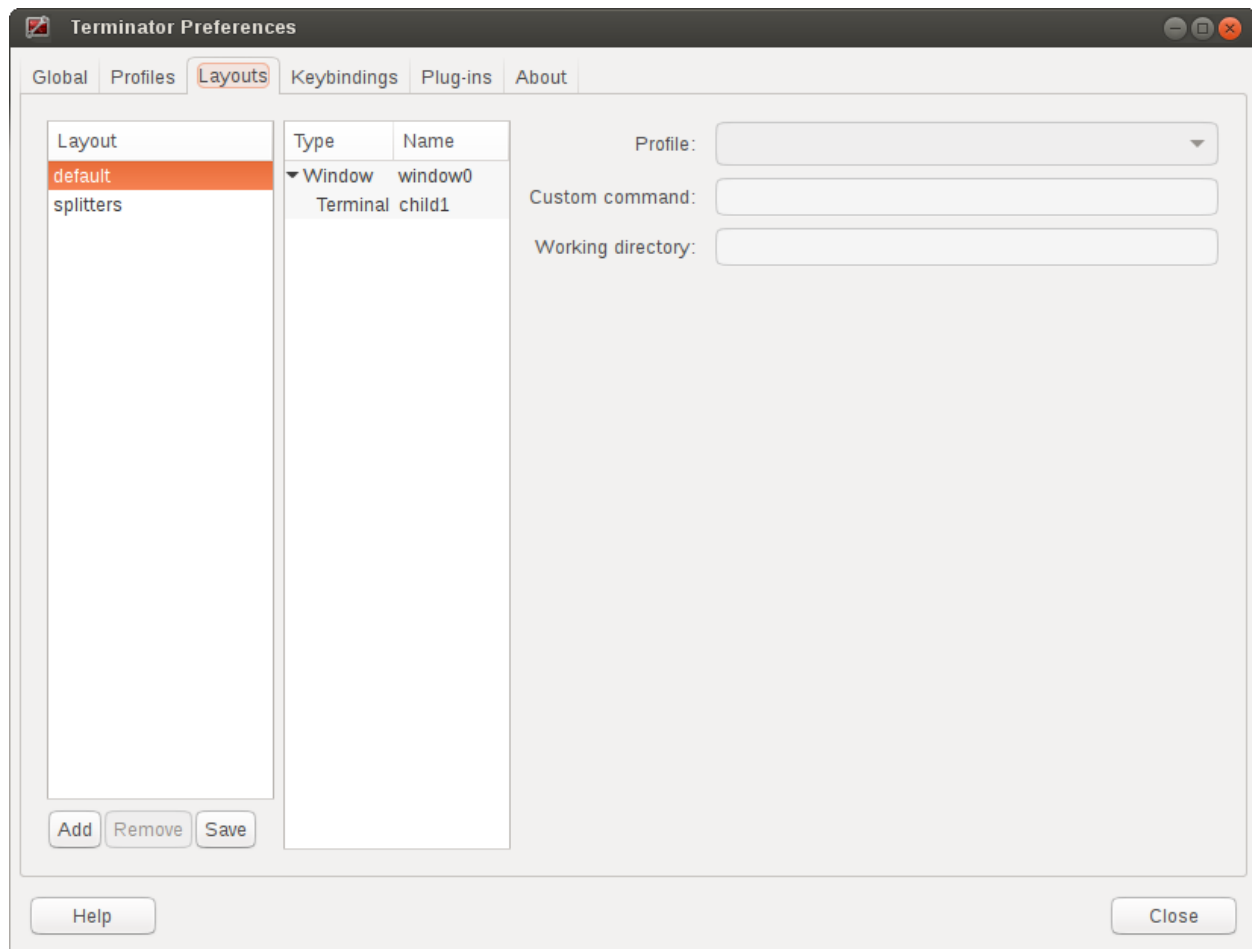
A simple panel describing a bit about the application, and a set of links that will guide users to some helpful Terminator project resources.



1.5 Layouts and the Layout Launcher

Layouts are how Terminator helps you quickly start a given set of windows with the terminals arranged just how you like, and even launching unique commands in each one.

You have already seen a glimpse of this in the *Layouts* tab of the *Preferences Window*. If you haven't already read the information there, you should probably go back and do so. Here's the *Layouts* tab again:



It's simple to create a new layout. Just launch new windows, add tabs and splits, arrange them, and customise them to your liking.

Layouts will currently directly detect and save:

- Window sizes and positions as well as the fullscreen or maximised state
- Splitter positions
- Custom window, tab and titlebar names
- The profile of each terminal
- Group setting of each terminal
- The active terminal per window or tab, and the active tab per window if applicable
- The UUID of each terminal

When done, use the [Layouts](#) section of the [Preferences Window](#) to keep this layout for future use. You save them by using the **Add** or **Save** buttons, where *Add* creates a new layout entry and prompts for a name, and *Save* updates the currently selected layout.

Warning: Currently some things are not detected by the code, and have to be configured in the [Layouts](#) tab of the [Preferences Window](#) after the layout is saved/added.

This means that if you use the *Save* button in the *Layouts* after spending time setting the items below, you *will* lose these stored values.

- Custom command
- Working directory

First get the layout right, then edit these within the *Layouts* tab of the *Preferences Window*. You do **not** need to use the *Save* button to keep these settings.

There is potential to improve this behaviour, as it *is* a little unintuitive.

1.5.1 The Layout Launcher

You can set up an application launcher with the `-l LAYOUT` option which will load the named layout, but what if you have a long list of layouts, like me? It can be annoying distinguishing between 30 items with the same icon, waiting for a tool-tip to tell you which one you're about to launch. No-one has the stamina to draw 30 distinct icons representing all these layouts either!



Enter the **Layout Launcher**, as shown on the right. This will list all of your saved layouts in alphabetical order, apart from *default*, which is always at the top. You can double-click an entry, highlight it and select **Launch**, or use the keyboard to move the highlight, pressing `Return` to launch.

The *Layout Launcher* can be opened from a running terminal using a shortcut, or by running Terminator with the `-s` option. This option could be set in an application launcher, to get to the Layout Launcher with a single click.

You can have more than one *Layout Launcher* window open, or you could launch one at the beginning, and pin it to always be on the visible workspace.

Here's a brief run-down of keyboard and mouse use:

Action	Mouse	Default Shortcut
Open the Layout Launcher	N/A	Alt+L
Move Up/Down list	click	<Up/Down Arrow>
Launch a layout	double-click	Return

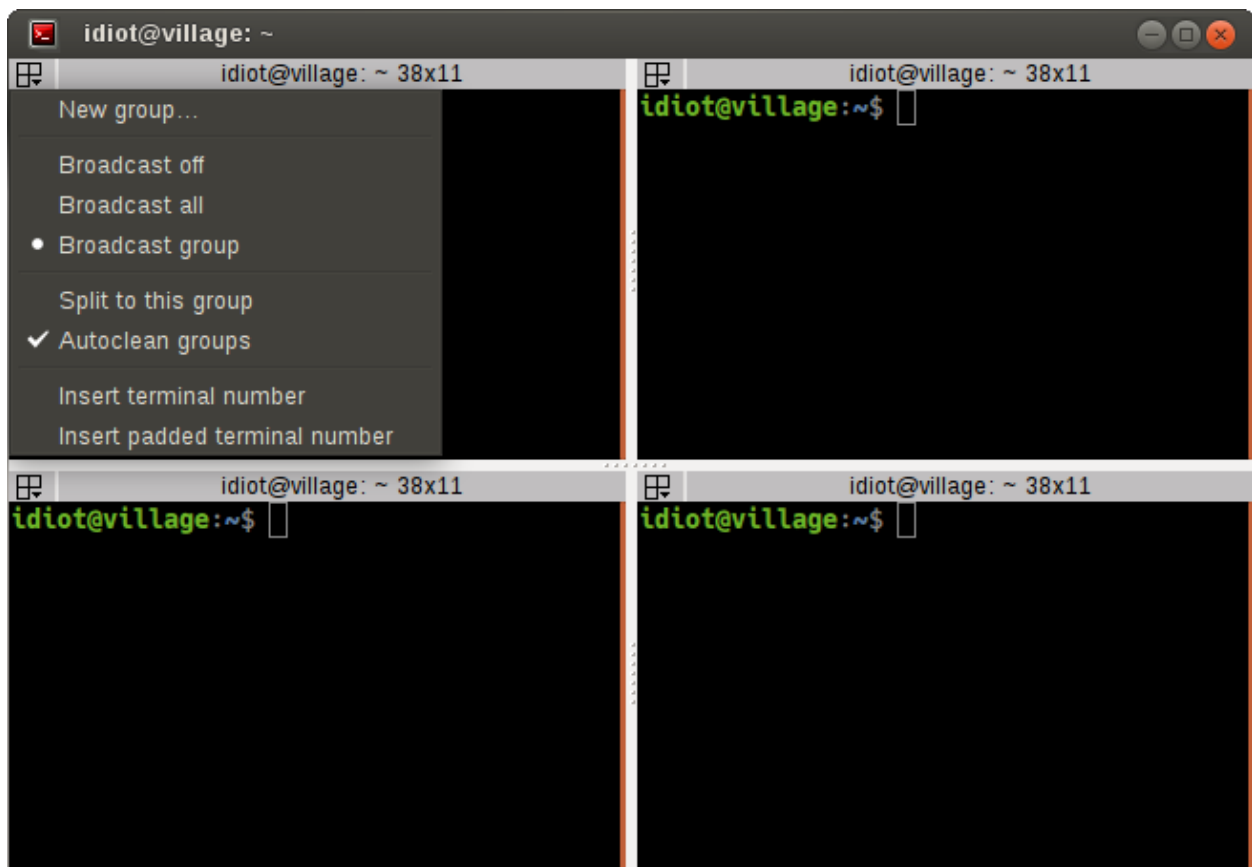


1.6 The Grouping Menu

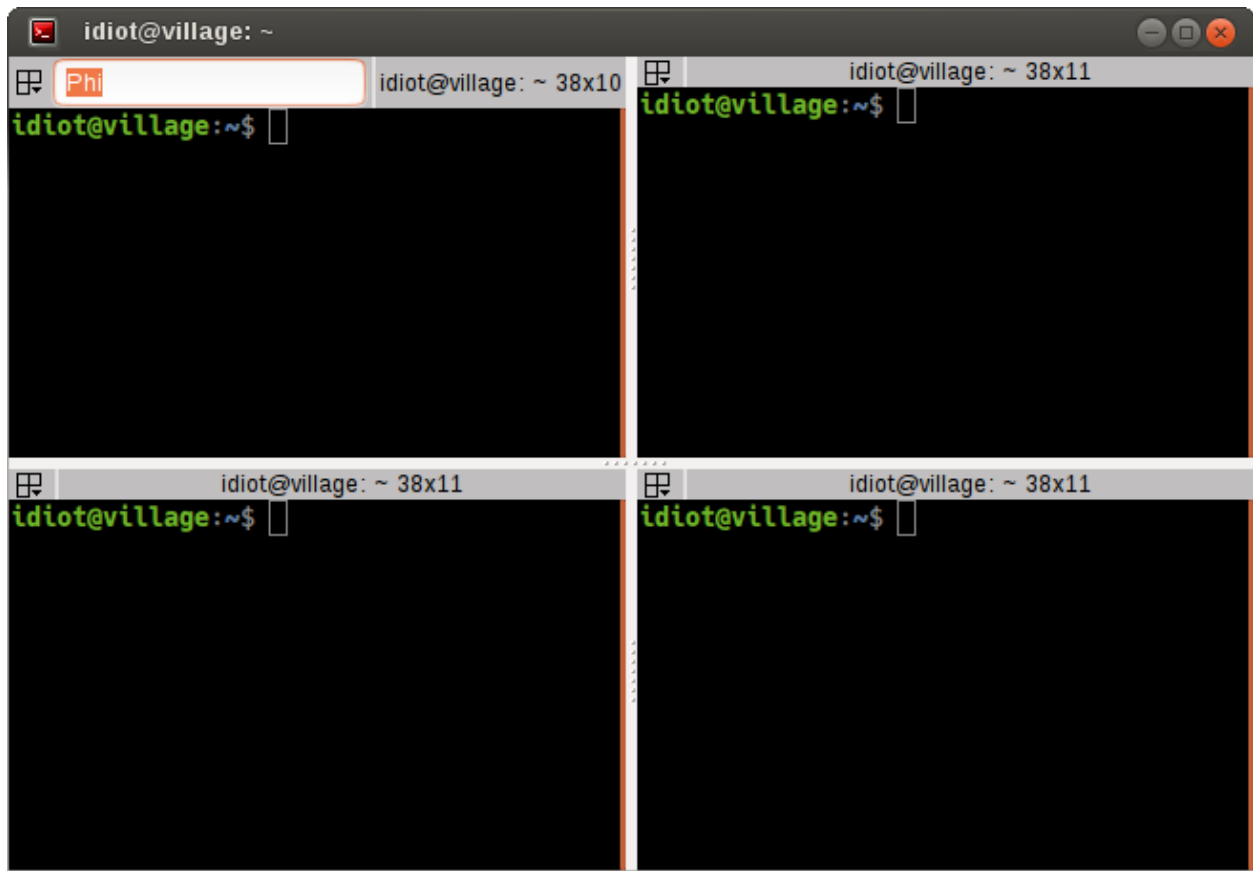
1.6.1 Manipulating terminal groups

Grouping, as the name suggests, allows grouping of terminals so that actions can be taken that affect more than one terminal. As usual, following along will help understanding, so let's start with a basic window, then split into a 2x2 grid.

Let's have another look at the grouping menu for reference, because as we proceed, it will change:

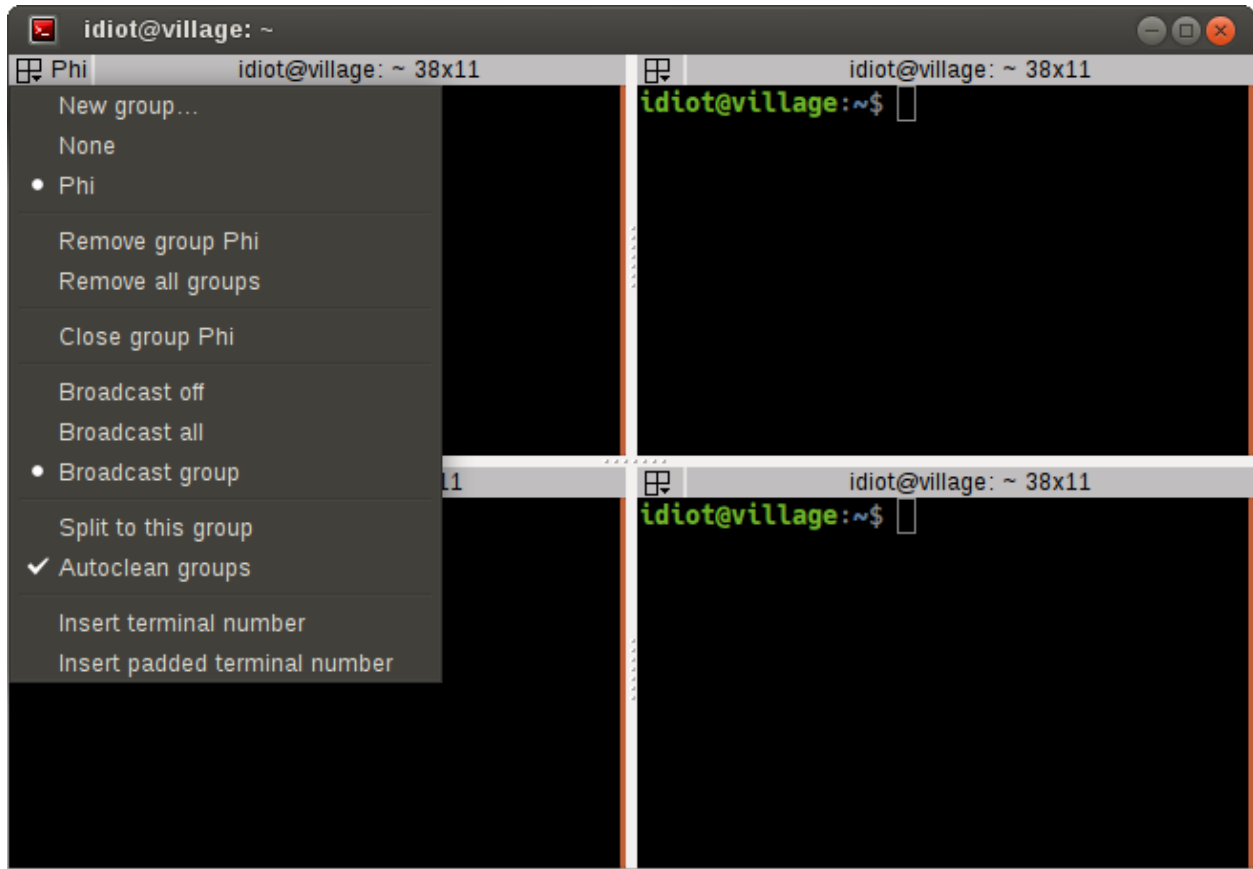


The first item **New group...** allows you to create a named group, using an editable field that will appear next to the 3-box icon. By default this will be given a randomly selected name from the names of the Greek alphabet. Here you can see **Phi** was selected:



You can either start typing to replace the provided one with something more descriptive, or you can accept the default with Return. For the purpose of this document I will just be using default names.

If you again click the 3-box/group button, you will see that several new entries have been added to the menu:



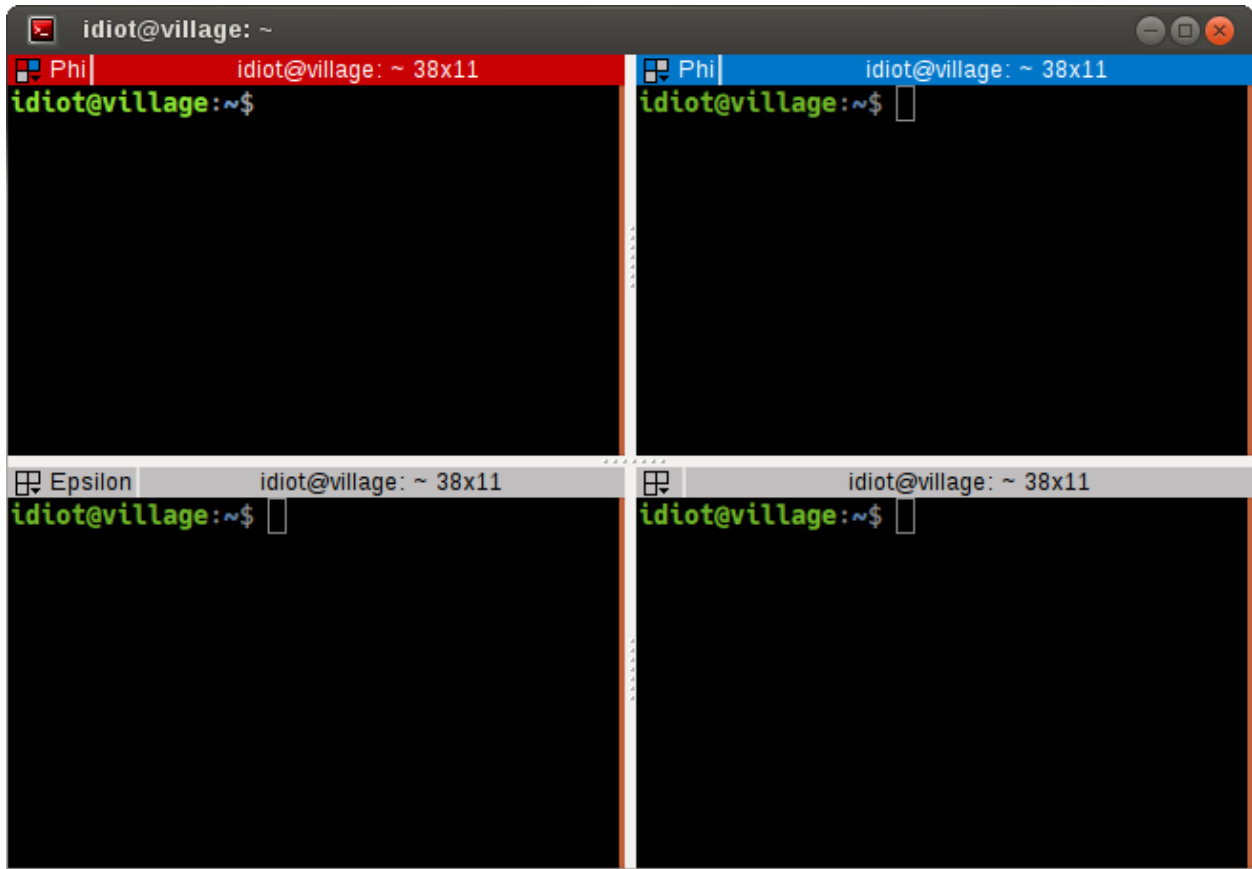
Because the terminal now has a group the first new entry is **None** which will remove the grouping for this terminal.

Following that is a list of the known groups, in this case only *Phi* so far. This list (plus the *None* entry) can be used to change the group of this terminal.

Next are two items: **Remove group Phi** and **Remove all groups**. You will only get the currently assigned group group as an option to remove, as well as an option to remove all groups. This does not close the group, but simply removes the group assignments from the terminals.

The next option is more destructive: **Close group Phi** will exit the terminals assigned to this group.

Here I've gone ahead and set the upper right terminal to the same *Phi* group, and the lower left terminal to a *New group...* of **Epsilon**.



A quick glance at the menu again will show that the only change is the addition of the *Epsilon* group to the list.

To change a group name you can either click on *New group...* again, or **Shift+click** on the 3-box/group button, and the editable field will reappear.

The current terminal is the one with focus. By clicking with key modifiers on other terminals 3-box/group button we can speed up group organisation. Here's a list of mouse actions (including some already detailed):

Action	Mouse	Default Shortcut
Group menu popup	click	(TBD)
Edit group for terminal	Shift+click	(TBD)
Edit group for all terminals in the "group"	Shift+Super+click	(TBD?)
Toggle to current terminals group ¹	Ctrl+click	
Toggle "group" to current terminals group ¹	Ctrl+Super+click	

Warning: The terminals with no named group are also considered a de facto group. If you use one of the Super shortcuts on a terminal with no group, you will also include all the other terminals with no group.

Skipping slightly ahead in the menu, there are two options that make more sense covered here. The **Split to this group** (default: off) option means that when you split the current terminal, the new one will inherit the group of the current terminal. It is off by default in which case new terminals have no group.

The second option **Autoclean groups** (default: on) will remove a group from the menus group list when the last terminal with that group is closed. If off groups will remain in the list until the application exits, or the option is

¹ These shortcuts will only work if you use them on terminals that are *not* the current terminal.

enabled.

Note: Not shown in the above screenshots, there are also menu items for grouping all terminals in a tab in the menu. They only appear once a new tab is created - **Group all in tab** and **Ungroup all in tab**.

Some final group related shortcuts are for grouping all terminals at once, or grouping terminals in the same tab.

Action	Default Shortcut
Group all	Super+G
Ungroup all	Shift+Super+G
Group tab	Super+T
Ungroup tab	Shift+Super+T
Group all toggle	
Group tab toggle	

1.6.2 Broadcasting input to multiple terminals

So first let me describe some terminology (no pun intended). **Broadcasting** is the act of sending your input to multiple terminals. The current terminal (the one with the red titlebar by default) is always the **broadcaster**. Any terminal that is in the same group as the current terminal is a potential **receiver**. I say potential because the act of broadcasting can be turned on and off independently of the grouping.

This can be an invaluable time-saver when having to do active investigation across multiple machines where you would be repeating the same commands on each of them.

Terminator's titlebar is colour-coded to help you quickly see which terminals are potential receivers, and whether they *will* receive the broadcast input.

The titlebar is split into two parts. The leftmost part is the 3-box/group button that has one of three background colours as defined in the *Global* tab of *Preferences Window*:

- *Red* - The current terminal and broadcaster.
- *Blue* - A terminal that is in the same group as the broadcaster.
- *Grey* - A terminal in a different group, or no group.

The second part consists of the title, and uses the same colouring to show the following:

- *Red* - The current terminal and broadcaster.
- *Blue* - A terminal that is acting as a receiver and will duplicate input from the broadcaster.
- *Grey* - A terminal that is not a receiver.

There are three settings for broadcasting, selected from the *Group menu*. Following are images of each of these modes, with `test` typed into the current broadcasting terminal:

- *Broadcast off*
- *Broadcast all*
- *Broadcast group* (default mode; can be changed in the *Global* tab of *Preferences Window*.)

Warning: Be careful with additional tabs, windows, or when you are zoomed or maximised on a single terminal. Just because you cannot see a terminal does not mean the terminal is not receiving. This can cause problems if you are typing a destructive command without realising that this command is going to other terminals.

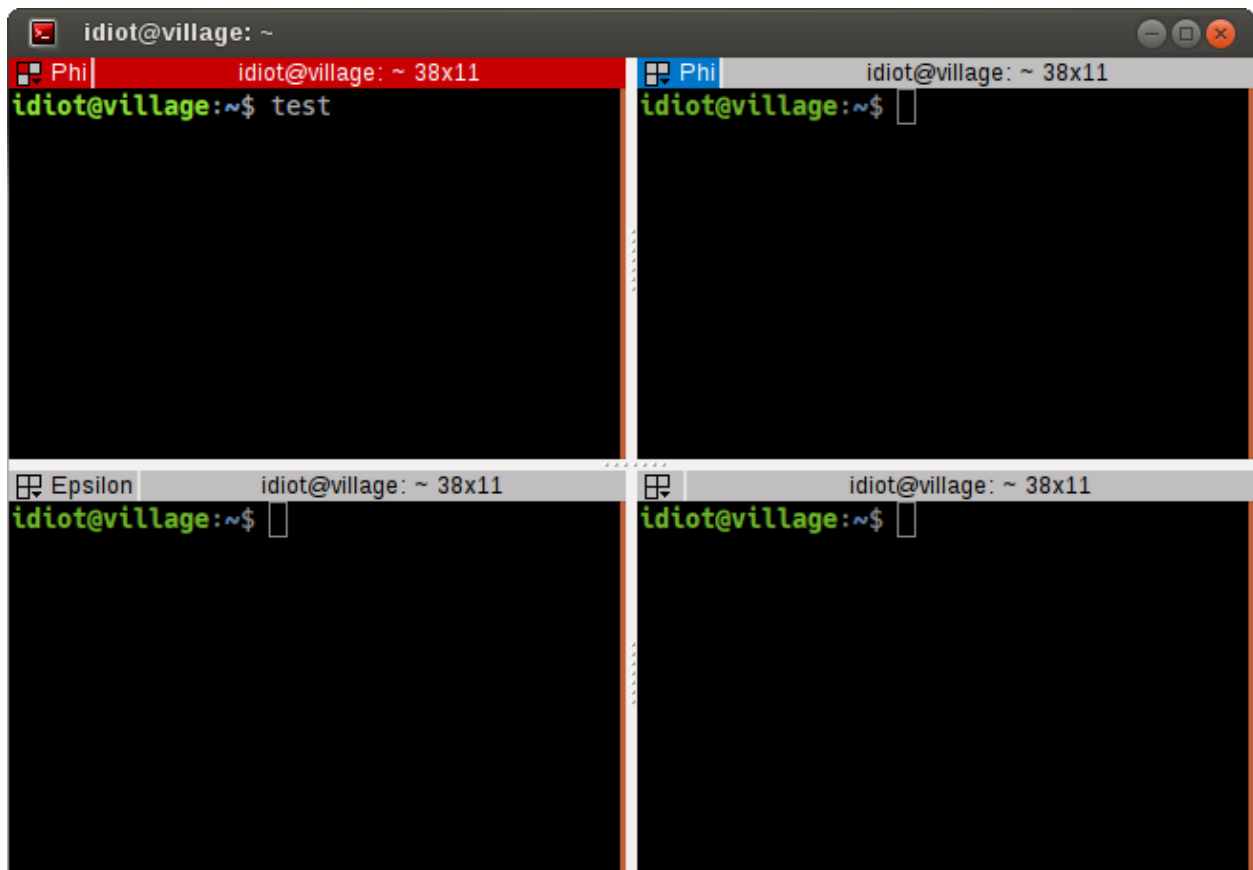


Fig. 1: Here you can see that only the current terminal receives input, even though the upper right terminal is also a part of the *Phi* group.

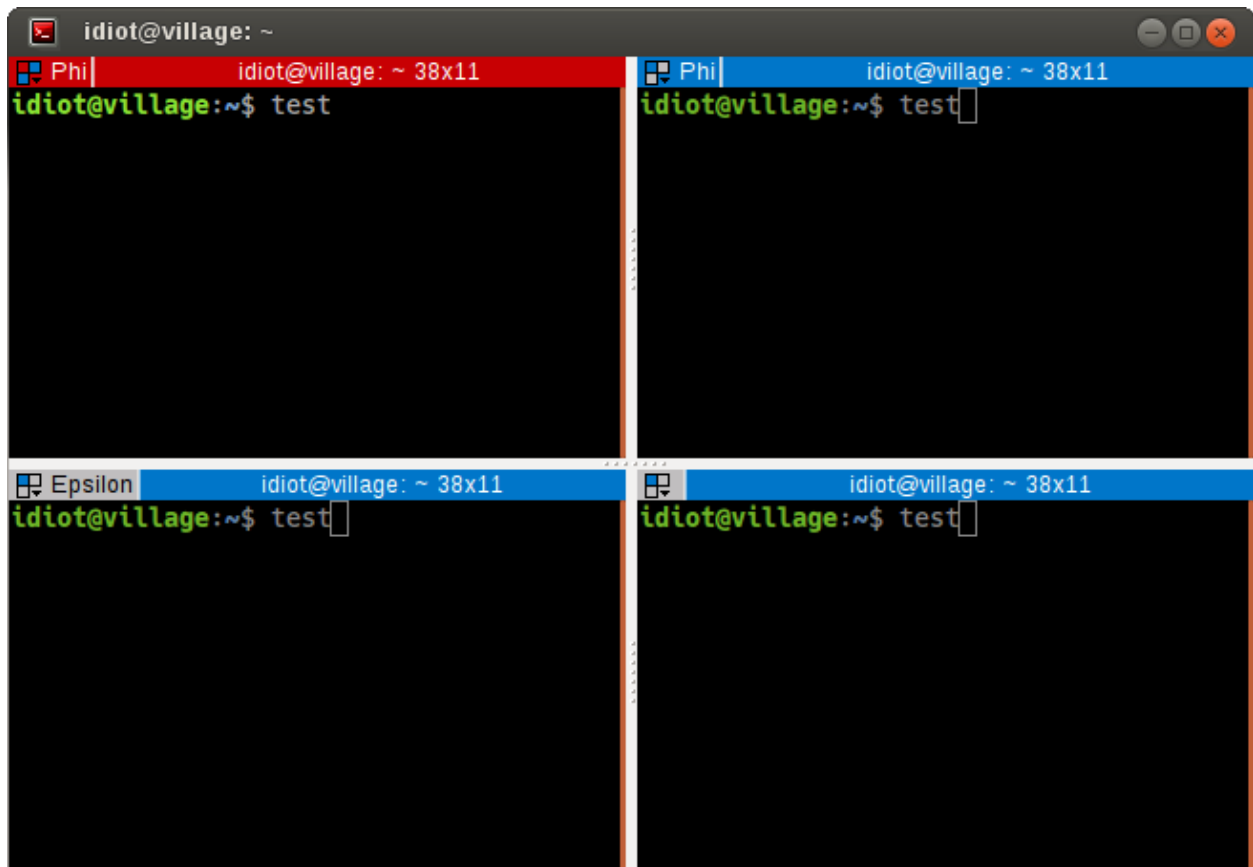


Fig. 2: Here you can see that all terminals, including those in other groups, or with no group, receive the input.

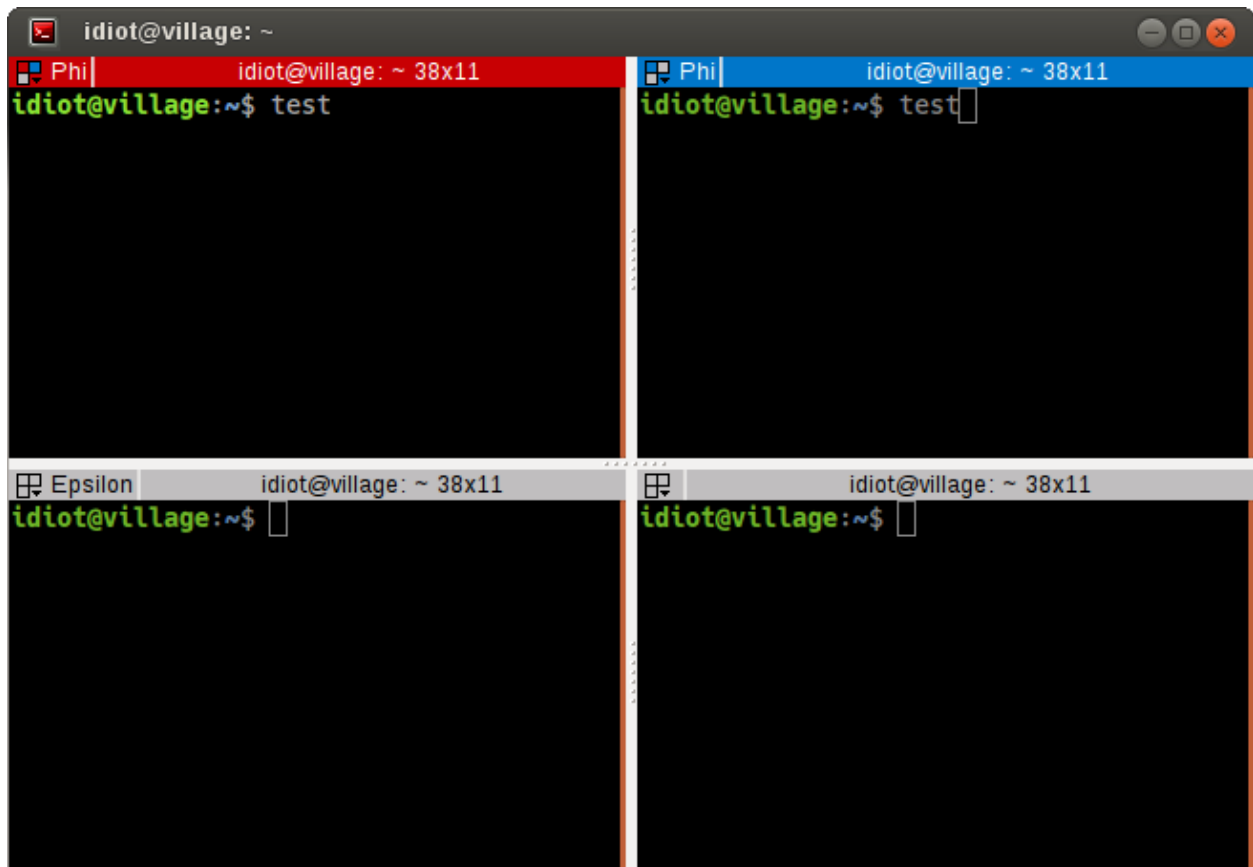


Fig. 3: Here you can see that only the terminal(s) in the same group as the broadcaster receive input.

In particular, note that when you run Terminator a second time, by default you are *not* running a completely separate process. It *is* still connected to the grouping of the initially launched Terminator process. If you need a completely separated window you need to use the `-u` option or disable the Dbus interface in your config file.

Those with good eyesight may have spotted the other visual indicator of the colours in the 3-box icon changing for the different status of the terminals too. When you are zoomed or maximised, the presence of blue in this icon might be the only visual indicator of receivers.

As with all things in Terminator, we have shortcuts to help you keep your hands on the keyboard:

Action	Default Shortcut
Broadcast off	Alt+O
Broadcast all	Alt+A
Broadcast group	Alt+G

Warning: It has become apparent that the complexity of the input systems used (IBus, IME, dead key layouts, etc) can cause problems with broadcast input.

Instead of getting your intended character in all receivers, you will only get the composed letter in the current terminal, and what goes into the receivers is a bit unpredictable. If you copy and paste the character into the current terminal then it will be input into the receivers.

You may not even realise that you are running one of these systems (I had IBus by default, although I did not experience issues with it.) Killing or disabling them should temporarily fix the problem. There is a fix for the IBus issue in newer the GTK3 version of Terminator, but we still encounter people for whom this function is not 100% reliable.

1.6.3 Insert terminal number

The last two menu items are slightly out of place here. They are the same function as the shortcuts mentioned [here](#). They were added early on when the broadcast feature was added, and the argument could be made for removing them. So far no-one has though, so for now they will stay here.



1.7 Plugins

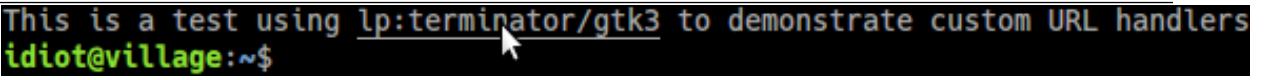
Terminator can be expanded using plugins. Additional features can be created outside of the main application, and added in at runtime.

In theory you should be able to implement fairly powerful plugins, although so far the included ones we have are fairly small in scope.

The current plugins do not have configuration options in the [Plugins](#) tab of the [Preferences Window](#). The plugin architecture was created before I (Steve Boddy) became maintainer, and so far I haven't had reason to figure out the detail. I'm not entirely sure if/how a plugin can add options to the configuration options in the [Plugins](#) tab. What plugins can definitely do, because examples are below, is to:

- add menu items to [The Context Menu](#),

- create their own windows,
- create handlers for strings that match a pattern.

Note: 

Several of the included plugins create *Click-able items* in the terminal. These are highlighted by underlining the item when the mouse hovers over it.

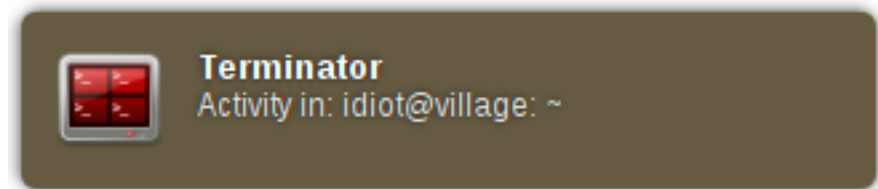
1.7.1 Included plugins

The following plugins are distributed by default with Terminator.

Note: Unless otherwise stated, the included plugins are under the *Licensing* as Terminator, GNU GPL v2.

Activity Watch

Original Author: Chris Jones



Adds a menu item, **Watch for activity**, to *The Context Menu* which will create a notification, as seen to the right, when there is output to the terminal. This is useful when you have a long running command and wish to know when it has completed, or output an update.

There is one option for this plugin:

hush_period (default: 10.0)

How long in seconds until the next notification of activity is presented.

Note: There is currently no way to edit these options in the GUI, it must be done directly in *The Config file*.

An extract of this item being set would be:

```
[plugins]
[[ActivityWatch]]
    hush_period = 30.0
```

Which would wait 30 seconds before showing another notification of activity.

Note: Bear in mind also that your notification may look very different to the image shown due to theming.

APT URL Handler

Original Author: Chris Jones

Text matching `apt:.*` will be converted into a click-able item that when triggered with `Ctrl+click` will launch the default package manager for software on a debian system.

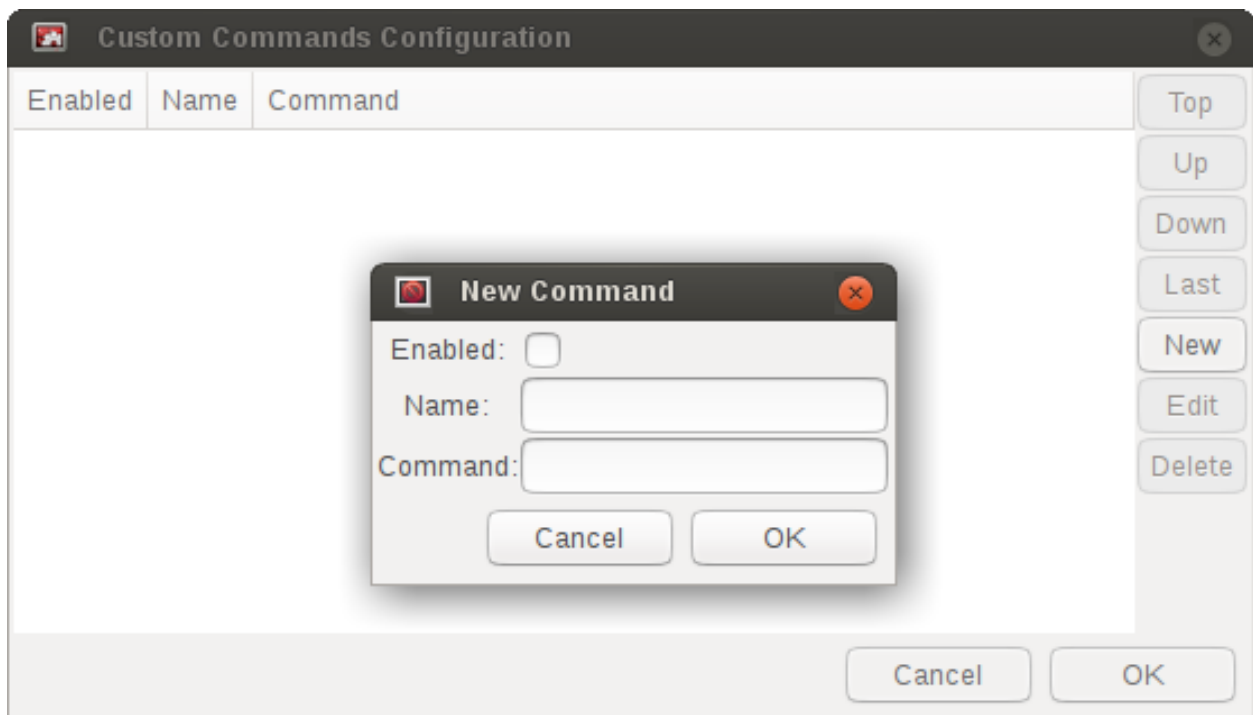
right-click over the URL will add two entries to *The Context Menu*:

- *Open software manager* - Same as `Ctrl+click`
- *Copy package URI* - Just copies the URI to the clipboard

Custom Commands Menu

Original Author: Chris Jones

Adds a menu item, **Custom Commands**, to *The Context Menu* which has a sub-menu containing its own **Preferences** item that launches the window show below. Below that is a list of user configured commands that can be chosen.



In this window you can create a **New** item, and **Edit** or **Delete** existing ones. The selected item can be repositioned in the sub-menu order using the **Top**, **Up**, **Down** and **Last** buttons.

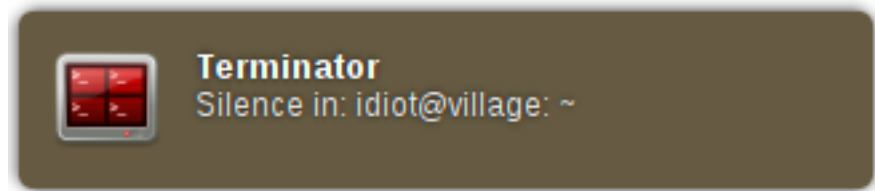
Clicking **New** or **Edit** gives the smaller window. An **Enabled** item is shown in sub-menu, and a disabled one is not. The **Name** is used for the sub-menu item text. The **Command** is the text that will be entered into the current terminal with a `Return` at the end to execute/enter it. You *do not* get a chance to edit the text first.

A rudimentary support for sub-trees is implemented. Simply add one or more `/` in the *Name* field, and the tree structure will be created. Positioning is determined by the first time a sub-tree is referenced, so a later reference will be attached to the sub-tree defined by an earlier reference.

Note: If other terminals are receiving, they too will receive and execute the *Command*.

Inactivity Watch

Original Author: Chris Jones



Adds a menu item, **Watch for silence**, to *The Context Menu* which will create a notification, as seen to the right, when a terminal has been quiet for a given period. This is useful when you have a long running process that outputs constantly (i.e. compiling a kernel) and you wish to know when it has ended. This notification will only show once, unless there is some activity in the terminal after the initial notification.

There are two options for this plugin:

inactive_period (default: 10.0)

How long in seconds until a terminal is considered inactive.

watch_interval (default: 5000)

How long in milliseconds between checks for inactivity.

Be aware that this combination will result in some uncertainty as to the exact timing of the notification. In the worst case, with the values given, the notification may take 14.9 seconds to appear.

Note: There is currently no way to edit these options in the GUI, it must be done directly in *The Config file*.

An extract of these items being set would be:

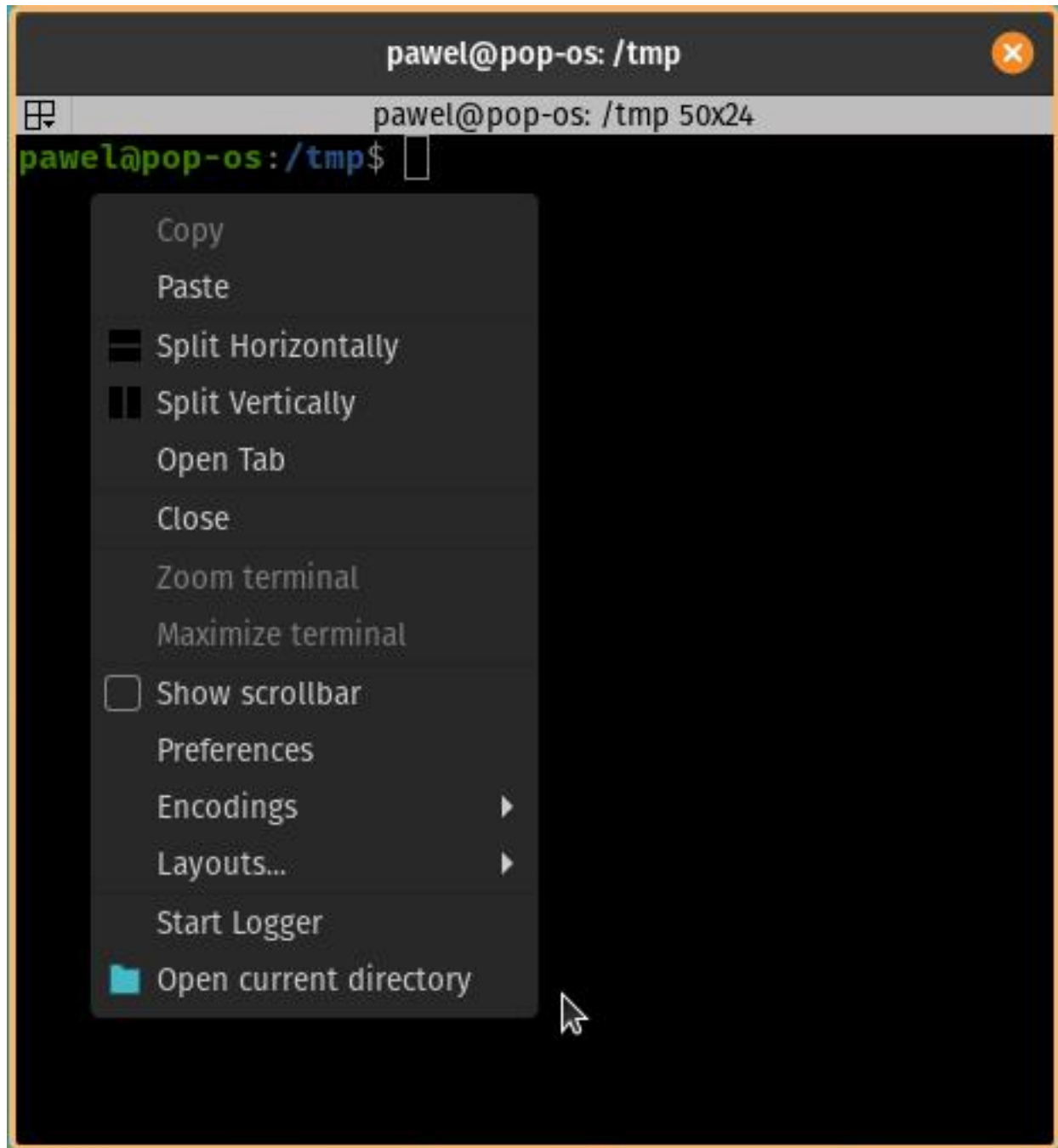
```
[plugins]
[[InactivityWatch]]
    inactive_period = 30.0
    watch_interval = 1000
```

Which would check every second if the terminal had been silent for 30 seconds.

Note: Bear in mind also that your notification may look very different to the image shown due to theming.

Current Directory Open

Original Author: Paweł Kotiuk



Adds a menu item, **Open current directory**, to *The Context Menu* which will open current directory in default file explorer.

Launchpad Bug URL Handler

Original Author: Chris Jones

Text matching `lp: #12345` where 12345 is a bug number in launchpad, will be converted into a click-able item that when triggered with `Ctrl+click` will launch a browser to the bug report in launchpad.

Additionally the plugin will accept variants where the prefix is in capitals, i.e. `LP`, and the `:`, white-space, and `#` are

optional.

The item can also be more than one bug number, and each will be opened, for example:

```
lp: #12345. #67890, 54321, #9876
```

Ctrl+click on this will open four pages; one for each bug number.

right-click over the URL will add two entries to *The Context Menu*:

- *Open Launchpad bug* - Same as Ctrl+click
- *Copy bug URL* - Just copies the URL to the clipboard

Launchpad Code URL Handler

Original Author: Chris Jones

Text matching `lp:string` will be converted into a click-able item that when triggered with Ctrl+click will launch a browser to the page in launchpad, where string is one of the following:

- *project* - i.e. `lp:terminator`
- *project/series* - i.e. `lp:terminator/gtk3`
- *group/project/branch* - i.e. `lp:~sparkstar/terminator/terminator`
- *group/+junk/branch* - i.e. `lp:~<yourname>/+junk/terminator`

Additionally the plugin will accept variants where the prefix is in capitals, i.e. `LP`.

right-click over the URL will add two entries to *The Context Menu*:

- *Open Launchpad branch* - Same as Ctrl+click
- *Copy branch URL* - Just copies the URL to the clipboard

Logger

Original Author: Sinan Nalkaya

Adds a menu item, **Start Logger**, to *The Context Menu* which will popup a window for selecting a file name to save as. Any content then written to the terminal will be written to the file too. Once started the menu item will change to **Stop Logger** which does precisely what you would expect.

Warning: There appears to be problems when applications switch to/from alternate mode (i.e. vi, mc, etc.) The obvious one is that the alternate screen is not “logged” although it is not clear how this *could* be logged. The second issue is that some of the output after the alternate screen is not logged. See [LP#1477386](#) for more info and progress.

Maven Plugin URL Handler

Original Author: Julien Nicoulaud

Ummmm. . . . I’m not entirely sure what this will do, as I don’t use Maven. Updates on a postcard, please. . .

From the source:

Maven plugin handler. If the name of a Maven plugin is detected, it is turned into a link to its documentation site. If a Maven plugin goal is detected, the link points to the particular goal page. Only Apache (org.apache.maven.plugins) and Codehaus (org.codehaus.mojo) plugins are supported.

Terminal Shot

Original Author: Chris Jones

Adds a menu item, **Terminal screenshot**, to *The Context Menu* that will take a screenshot of the underlying terminal, and present a dialog for where to save it.

Test Plugin

Original Author: Chris Jones (most likely)

An almost comically stripped down example.

1.7.2 Third party plugins

As I find (or I'm told about) plugins that are available elsewhere, I'll add links here. I've done a preliminary search, and.. Wow! I never knew there were so many out there.

If any of the authors would like to get their plugins added to the main Terminator package, or they would prefer not to be listed here for some reason, they can reach out to me through the project site on Launchpad and we can sort it out.

I'm unsure of how these plugins are perceived. They are specific to Terminator, but does that make them derivative in the eyes of GPL v2, and therefore allow me to include them? If I want to include one in the main package, do I have to hope the creator is still active? Answers on a postcard...

Warning: I have done no testing or checking of these plugins. You use at your own risk, and you are responsible for evaluating the code for bugs, issues, and security.

Warning: While we have ensured the included plugins have received the required changes to function with GTK3, the third party plugins are not under our control. Examine the change logs of the respective plugin and look for commits that mention GTK3 updates.

In absolutely no order at all...

Editor / Command

https://github.com/rail/dotfiles/blob/master/terminator_bugzilla_handler.py

- terminator_bugzilla_handler: Link "bug:12345" to the Mozilla bugzilla. (As it is for Mozilla, it seems a bit misnamed.)

<https://github.com/arnaudh/terminator-plugins>

- open_any_file_plugin: Open any file with it's default application

<https://github.com/mchelem/terminator-editor-plugin>

- editor_plugin: Click on file:line style links to launch a text editor

https://github.com/papajoker/editor_terminator

- editor_plugin: Another text editor launcher

Session / Layout

https://github.com/ilgarm/terminator_plugins

- clone_session: Split and clone ssh session

<https://github.com/camillo/TerminatorPlugins>

- LayoutManager: Saves and restores Layouts (which is built-in now, possibly redundant)
- TerminalExporter: Export contents to file

<https://github.com/abourget/abourget-terminator>

- TenscoresPlugin: Seems to be for launching set of tabs (which is built-in now, possibly redundant)

<https://github.com/alesegdia/terminator-plugins>

- Session: Save/load sessions (which is built-in now, possibly redundant)

Display / Theme

<https://github.com/Theer108/colorize>

- colorize: Colour titlebar of each terminal separately

<https://github.com/GratefulTony/TerminatorHostWatch>

- hostWatch: Attempts to figure out your current host, and apply a certain theme.

<https://bitbucket.org/pgularski/terminator-plugins>

- show_titlebar: Menu item to show/hide the titlebar.
- searchplugin: Yup, another Googler.

Logging / Dump

<https://github.com/kmoppel/dumptofile>

- dump_to_file: Dump console contents to a text file.

<https://www.snip2code.com/Snippet/58595/Terminator-plugin—log-the-output-of-t>

- my_logger: Log the output to a file with a time-stamp as the name, and prefix each line with the time. (Seems to be similar to, or derived from, the included one)

Other

<https://github.com/choffee/terminator-plugins>

- searchplugin: Search Google for the selected text in a terminal

<https://github.com/mikeadkison/terminator-google>

- google: Another google-the-text plugin

<https://github.com/iambibhas/terminator-plugins>

- `hastebin`: Uploads selected text to Hastebin and opens browser on it

https://github.com/papajoker/git_terminator

- `git_plugin`: adds commands for git when it detects a `.git` folder

<https://github.com/dr1s/terminator-plugins>

- `cluster_connect`: A way to connect to multiple machines as a cluster

<https://github.com/mariolameiras/ssh-menu-terminator>

- `ssh_menu`: I'm guessing a bit, but I think it works with SSH Menu ;-) the code is quite big to understand at a glance.

<https://github.com/ju1ius/clisnips>

- `clisnips`: Snippets for the command line.

<https://bitbucket.org/johnsanchez/terminator-applauncher>

- `applauncher`: A launcher/set-up tool (which is built-in now, possibly redundant)

<https://github.com/OlivierBoucher/terminator-k8s-plugin>

- `k8s`: NEW! Work in progress, with the ultimate goal to provide k8s specific informations in the shell title bar.

1.7.3 Installing a plugin

A plugin can be installed by adding the main python file (along with any additional files) in one of two locations:

`/usr/[local/]share/terminator/terminatorlib/plugins/` This will need root permissions to do.

The optional `local/` is usually for packages installed by hand, rather than through the package manager, and this depends on how Terminator was installed on your system.

`~/.config/terminator/plugins/` This allows you to use plugins without needing root.

1.7.4 Creating your own plugins

Note: The following guide was initially sourced from a now archived [tutorial](#) written by Chris Jones back in April 2010. I'm reproducing it here as a precaution, although I don't expect the archived original will disappear. It will get rewritten and expanded as more knowledge and information is added.

One of the features of the new 0.9x series of Terminator releases that hasn't had a huge amount of announcement/discussion yet is the plugin system. I've posted previously about the decisions that went into the design of the plugin framework, but I figured now would be a good time to look at how to actually take advantage of it.

While the plugin system is really generic, so far there are only two points in the Terminator code that actually look for plugins - the Terminal context menu and the default URL opening code. If you find you'd like to write a plugin that interacts with a different part of Terminator, please let me know, I'd love to see some clever uses of plugins and I definitely want to expand the number of points that plugins can hook into.

The basics of a plugin

A plugin is a class in a `.py` file in `terminatorlib/plugins` or `~/.config/terminator/plugins`, but not all classes are automatically treated as plugins. Terminator will examine each of the `.py` files it finds for a list called `available` and it will load each of the classes mentioned therein.

Additionally, it would be a good idea to import `terminatorlib.plugin` as that contains the base classes that other plugins should be derived from.

A quick example:

```
import terminatorlib.plugin as plugin
available = ['myfirstplugin']
class myfirstplugin(plugin.SomeBasePluginClass):
    # etc.
```

So now let's move on to the simplest type of plugin currently available in Terminator, a URL handler.

URL Handlers

This type of plugin adds new regular expressions to match text in the terminal that should be handled as URLs. We ship an example of this with Terminator, it's a handler that adds support for the commonly used format for Launchpad. Ignoring the comments and the basics above, this is ultimately all it is:

```
class LaunchpadBugURLHandler(plugin.URLHandler):
    capabilities = ['url_handler']
    handler_name = 'launchpad_bug'
    match = '\\b(lp|LP):?\\s?#?[0-9]+(,\\s*#?[0-9]+)*\\b'

    def callback(self, url):
        for item in re.findall(r'[0-9]+', url):
            return('https://bugs.launchpad.net/bugs/%s' % item)
```

That's it! Let's break it down a little to see the important things here:

- inherit from `plugin.URLHandler` if you want to handle URLs.
- include 'url_handler' in your capabilities list
- URL handlers must specify a unique handler_name (no enforcement of uniqueness is performed by Terminator, so use some common sense with the namespace)
- Terminator will call a method in your class called `callback()` and pass it the text that was matched. You must return a valid URL which will probably be based on this text.

And that's all there is to it really. Next time you start terminator you should find the pattern you added gets handled as a URL!

Context menu items

This type of plugin is a little more involved, but not a huge amount and as with `URLHandler` we ship an example in `terminatorlib/plugins/custom_commands.py` which is a plugin that allows users to add custom commands to be sent to the terminal when selected. This also brings a second aspect of making more complex plugins - storing configuration. Terminator's shiny new configuration system (based on the excellent `ConfigObj`) exposes some API for plugins to use for loading and storing their configuration. The nuts and bolts here are:

```
import terminatorlib.plugin as plugin
from terminatorlib.config import Config
available = ['CustomCommandsMenu']

class CustomCommandsMenu(plugin.MenuItem):
    capabilities = ['terminal_menu']
    config = None
```

(continues on next page)

(continued from previous page)

```
def __init__(self):
    self.config = Config()
    myconfig = self.config.plugin_get_config(self.__class__.__name__)
    # Now extract valid data from sections{}

def callback(self, menuitems, menu, terminal):
    menuitems.append(gtk.MenuItem('some jazz'))
```

This is a pretty simplified example, but it's sufficient to insert a menu item that says "some jazz". I'm not going to go into the detail of hooking up a handler to the 'activate' event of the MenuItem or other PyGTK mechanics, but this gives you the basic detail. The method that Terminator will call from your class is again "callback()" and you get passed a list you should add your menu structure to, along with references to the main menu object and the related Terminal. As the plugin system expands and matures I'd like to be more formal about the API that plugins should expect to be able to rely on, rather than having them poke around inside classes like Config and Terminal. Suggestions are welcome :)

Regarding the configuration storage API - the value returned by Config.plugin_get_config() is just a dict, it's whatever is currently configured for your plugin's name in the Terminator config file. There's no validation of this data, so you should pay attention to it containing valid data. You can then set whatever you want in this dict and pass it to Config().plugin_set_config() with the name of your class and then call Config().save() to flush this out to disk (I recommend that you be quite liberal about calling save()).

Wrap up

Right now that's all there is to it. Please get in touch if you have any suggestions or questions - I'd love to ship more plugins with Terminator itself, and I can think of some great ideas. Probably the most useful thing would be something to help customise Terminator for heavy ssh users (see the earlier fork of Terminator called 'ssherminator')



1.8 Advanced Usage

This is a grab-bag of topics that cover the bits you probably wouldn't use in day-to-day activities.

1.8.1 Command line options

Various options can be passed to Terminator at startup time to change numerous aspects and behaviour.

The following option sub-sections can also be seen in the manual page for Terminator:

```
man terminator
```

Note: I've rearranged and grouped the options compared to how they would appear using the -h option just to aid clarity.

General options

-h, --help Show a help message and exit
-v, --version Display program version
-g CONFIG, --config=CONFIG Specify a config file
--new-tab If Terminator is already running, just open a new tab
-p PROFILE, --profile=PROFILE Use a different profile as the default
-u, --no-dbus Disable *DBus*

Window options

-m, --maximise Maximise the window
-f, --fullscreen Make the window fill the screen
-b, --borderless Disable window borders
-H, --hidden Hide the window at startup
--geometry=GEOMETRY Set the preferred size and position of the window (see X man page)
-T FORCEDTITLE, --title=FORCEDTITLE Specify a title for the window
-i FORCEDICON, --icon=FORCEDICON Set a custom icon for the window (by file or name)

Shell options

-e COMMAND, --command=COMMAND Specify a command to execute inside the terminal
-x, --execute Use the rest of the command line as a command to execute inside the terminal, and its arguments
--working-directory=DIR Set the working directory

Layout options

-l LAYOUT, --layout=LAYOUT Launch with the given layout
-s, --select-layout Select a layout from a list

Custom Window Manager options

These settings are for people with heavy customisations to their window manager. Some window managers allow various rules to be applied, or actions to be taken, depending on how the window system perceives the window. These settings facilitate that.

-r ROLE, --role=ROLE Set a custom WM_WINDOW_ROLE property on the window

Note: In case you're looking for the previously supported `classname` setting, it has been removed as the gtk libraries deprecated the function call that allowed forcing the window class in this way.

Debugging options

See *Debugging* for more explanation of these options.

-d, --debug Enable debugging information (twice for debug server)

--debug-classes=DEBUG_CLASSES Comma separated list of classes to limit debugging to

--debug-methods=DEBUG_METHODS Comma separated list of methods to limit debugging to

1.8.2 The Config file

The default configuration file for Terminator is stored in the standard path for configuration files. It can be found at:

```
${HOME}/.config/terminator/config
```

It is human readable, and can be edited if you are careful. This is not generally recommended though, and you are, of course, strongly advised to make a backup before making manual changes.

There are many more specific details in the manual page:

```
man terminator_config
```

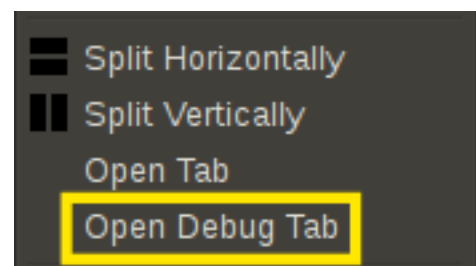
Warning: If you place items in the wrong location within the config file it can cause unintended results. In the worst case Terminator will fail to load. In the best case it will have no effect and you will simply be confused as to why your change has made no difference.

1.8.3 Debugging

There are inbuilt debugging features in Terminator. The simplest is to start Terminator from another terminal application (i.e. `gnome-terminal`) with the option `-d`. This will dump many debug statements to the launching terminal.

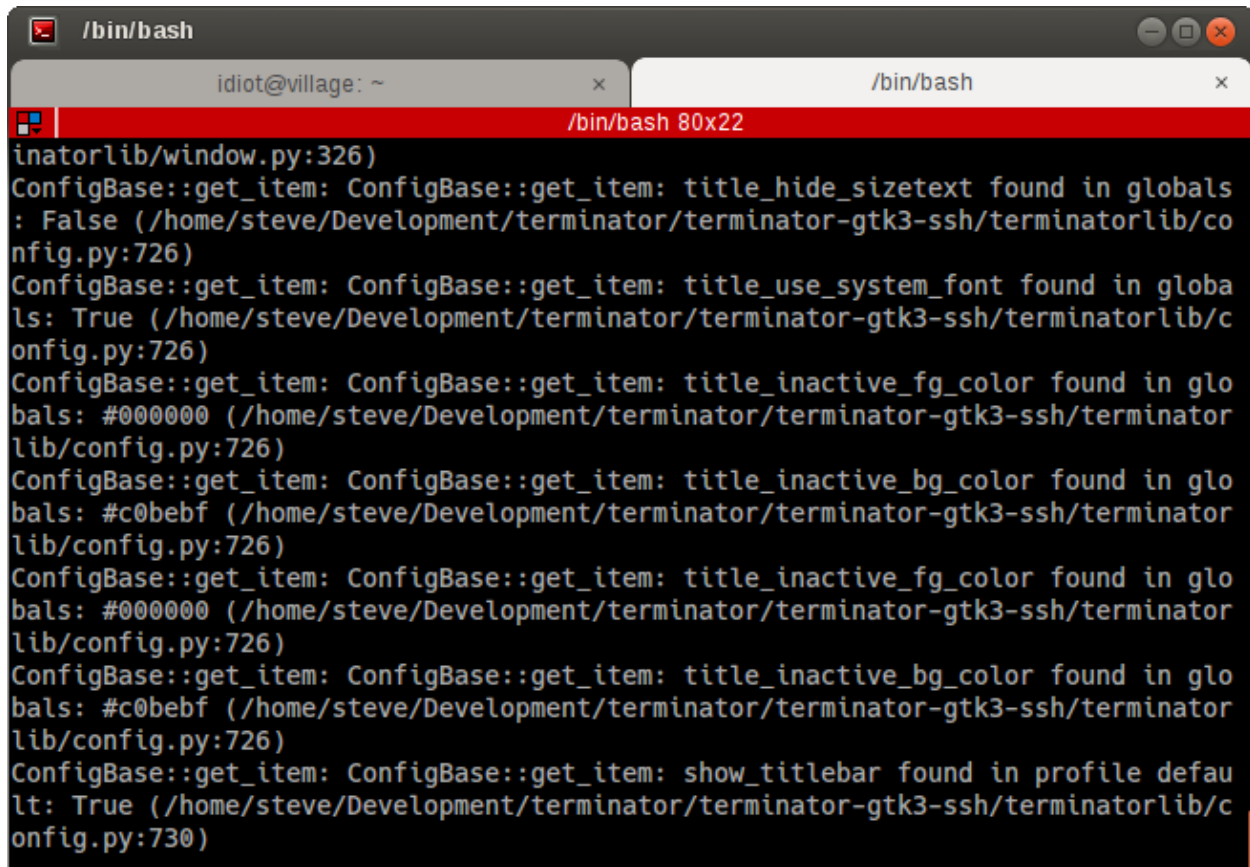
Note: If the Dbus is active in any other Terminator, then by default your attempt to launch with debug will launch a new window under the already running process. To prevent this you can use the `-u` option which will disable the Dbus interaction for the debugged instance of Terminator.

There is a lot of output, and a great deal of it will typically not be related to the area you are looking into. There are two more options that can be passed that limit the amount of debug lines to classes or methods of interest. See *Debugging options* for the detail.



The final facility is to start a debug server by passing `-dd` (this is the same as `-d -d`) which will start a debug server. With this setting a fourth item, **Open Debug Tab**, also appears in the second part of the *The Context Menu*, as highlighted in the image to the right.

Selecting it will give the following new tab with dedicated debug terminal:



```
/bin/bash
idiot@village: ~
/bin/bash 80x22
inatorlib/window.py:326)
ConfigBase::get_item: ConfigBase::get_item: title_hide_sizetext found in globals: False (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: title_use_system_font found in globals: True (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: title_inactive_fg_color found in globals: #000000 (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: title_inactive_bg_color found in globals: #c0bebf (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: title_inactive_fg_color found in globals: #000000 (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: title_inactive_bg_color found in globals: #c0bebf (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:726)
ConfigBase::get_item: ConfigBase::get_item: show_titlebar found in profile default: True (/home/steve/Development/terminator/terminator-gtk3-ssh/terminatorlib/config.py:730)
```

This prompt is a standard Python interactive prompt, but this is connected to the Terminator instance. You can explore the applications data structures, classes, etc. and can even call functions and methods.

Alternatively, instead of launching this tab, you could connect to the debug server from a different window. This requires that you search back through the debug output for the line containing “listening on”. Here you will see the port number, and you can simply use:

```
$ telnet localhost <port>
```

Warning: Whichever method you use the debug output is also dumped into this terminal, even though it is already there in the launching terminal. This can get rather annoying, and seems counter-productive to me, so a way to turn off the output in the debug console may be added. In the meantime you can use:

```
>>> from terminatorlib import util
... util.DEBUG=False
```

This should turn off the output, and let you explore the internal structure more easily.

Warning: Using the `-dd` option will make the *DBus* interface temperamental. Any attempt to use *Remotinator* will hang the main application.

The debug options and their usage are detailed [here](#).

1.8.4 DBus

DBus is a standardized form of IPC, or Inter-Process Communication. More detail about the internals of DBus can be found at the freedesktop.org for DBus.

In Terminator we currently use DBus for two tasks:

- Only run one instance of Terminator

The first instance will create the server. The second instance will fail to create the server, so it will request the first instance to create a new window (or new tab with `--new-tab`).

- Enable *Remotinator*

Warning: Running a single instance of Terminator can cause behaviour that is unexpected by the user when a terminator instance is already running. They do not have separate processes, and currently some features (in particular broadcast, and grouping keys) may include more terminals than you expect. You can work around this by using the `-u` option that will disable the DBus for that secondary instance.

The Layout Launcher already does this for you, and as a result any layout launched this way is running without DBus, and cannot be controlled with DBus. If you use the command line option `-l <LAYOUT_NAME>` to open a new layout, this will **not** disable the DBus unless you explicitly add the `-u` option too.

Note: There is quite some scope for improving this. I have a vague notion of a single master server and multiple instance servers, to improve the interaction between DBus and layouts.

1.8.5 Remotinator

Remotinator is a minimal wrapper around making DBus calls, and is typically run from *within* a Terminator terminal. This is not strictly necessary but, if you do not, you will have to do some extra work to determine the valid UUID of a current terminal and pass it as the `TERMINATOR_UUID` environment variable, or as the value to the `-u/--uuid` option. Remotinator is called within Terminator with:

```
$ remotinator <command>
```

or with one of the following:

```
$ remotinator --uuid <UUID> <command>
$ TERMINATOR_UUID=<UUID> remotinator <command>
$ export TERMINATOR_UUID=<UUID>; remotinator <command>
```

to force the UUID, or call it from outside Terminator.

There are a couple of commands that do not require a UUID. Please see the table below for details.

The following commands are currently enabled:

Command	Action
get_tab	Get the UUID of a parent tab
get_tab_title	Get the title of a parent tab
get_terminals ¹	Get a list of all terminals
get_window	Get the UUID of a parent window
get_window_title	Get the title of a parent window
hsplit	Split the current terminal horizontally
new_tab	Open a new tab
new_window ¹	Open a new window
vsplit	Split the current terminal vertically

Calling Remotinator without a command or with the `-h` will print the options and available commands to the terminal.

Note: If a layout has been launched using the *The Layout Launcher* or using the `-u` option Remotinator will not work with that layout as it is not connected to the DBus session.

As mentioned in the *DBus* section, this has the potential to be improved upon.

There is a lot of scope for expanding the available commands, and it is relatively simple to do, so is an ideal task for dipping ones toes.



1.9 Frequently Asked Questions

Here I'll try to list some common questions that get asked.

1.9.1 Why...

...is there another terminal program called Terminator?

There is *another terminal* project programmed in Java. It was begun a bit before this project, but when this projects creator searched the name I guess the other project did not come up. I don't know the details, but this project was always Terminator to me. I haven't received complaints from the other project, although they do get some people asking in their Groups for support on this project. Please don't do that folks.

I have contemplated a name change, although this project has a lot of visibility with it's current name, and it is hard to come up with a decent *alternative*.

...write in Python? It's slow/bloated/bad?

Performance

Profiles were configured with command `bash -c exit`, and the commands run a couple of times to get the caches loaded up.

¹ These commands **do not** require the UUID. If not marked as such then the command **does** require the UUID.

GNOME-Terminal:

```
idiot@village:~$ time for i in {1..30} ; do gnome-terminal --profile=Quickexit; done
real    0m10.606s
```

Terminator:

```
idiot@village:~$ time for i in {1..30} ; do terminator -g deletemeconfig -p Quickexit;
↪ done

GTK2: real    0m11.928s A smidgen slower.
GTK3: real    0m10.885s Yeah, basically identical!
```

Cold start, using `sync && echo 3 > /proc/sys/vm/drop_caches` before each run, then launching a single timed instance.

Gnome-Terminal:

```
idiot@village:~$ time gnome-terminal --profile=Quickexit
real    0m7.628s (approx median, there was a strange variance for GT, between 5 and 9
↪secs)
```

Terminator:

```
idiot@village:~$ time terminator -g deletemeconfig -p Quickexit

GTK2: real    0m11.390s (median of 3x)
GTK3: real    0m11.264s (median of 3x)
```

OK, so this is the once place you would notice an appreciable difference. How often do you start these things with completely empty caches/buffers?

In GTK2 there is a known issue which slows the cat'ing of large files quite a bit. The VTE in GTK3 Terminator is the *exact* same widget GNOME-Terminal uses, so this will get better, as and when we move fully to the in-progress GTK3 port. I should point out that this performance deficit is not due to the Python interpreter, or the Terminator Python code, but is solely down to the compiled C code VTE widget.

Memory use - The dumb way

GNOME-Terminal:

```
idiot@village:~$ for i in {1..100} ; do gnome-terminal --disable-factory & done
```

```
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free                # Before
↪startup
              total        used        free      shared    buffers     cached
Mem:          3102404      1388776      1713628         4052         164       45340
-/+ buffers/cache:      1343272      1759132
Swap:          3121996       788704      2333292
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free                # After
↪startup
              total        used        free      shared    buffers     cached
Mem:          3102404      2439524       662880        57196        1240       99212
-/+ buffers/cache:      2339072       763332
```

(continues on next page)

(continued from previous page)

```

Swap:      3121996      751440      2370556
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free           # After_
↪kill

```

	total	used	free	shared	buffers	cached
Mem:	3102404	1466536	1635868	4796	160	45912
-/+ buffers/cache:		1420464	1681940			
Swap:	3121996	751020	2370976			

Used (used mem -buffers/cache + swap)
 Before start: 2131976
 After start : 3090512 = 958536 kbytes, 936 Mbytes / 9.36 MBytes/instance
 After kill : 2171484 = 39508 kbytes, 38 Mbytes **not** recovered

Terminator GTK2:

```
idiot@village:~$ for i in {1..100} ; do terminator -g deletemeconfig -u & done
```

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	1313456	1788948	4284	152	43844
-/+ buffers/cache:		1269460	1832944			
Swap:	3121996	736844	2385152			

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	2866552	235852	19484	1084	65408
-/+ buffers/cache:		2800060	302344			
Swap:	3121996	736340	2385656			

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	1317724	1784680	4284	152	43464
-/+ buffers/cache:		1274108	1828296			
Swap:	3121996	736304	2385692			

Used (used mem -buffers/cache + swap)
 before start: 2006304
 after start : 3536400 = 1530096 kbytes, 1494 Mbytes / 14.94 MBytes/instance
 after kill : 2010412 = 4108 kbytes, 4 Mbytes **not** recovered

Terminator GTK3:

```
idiot@village:~$ for i in {1..100} ; do terminator -g deletemeconfig -u & done
```

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	1467204	1635200	4816	120	46132
-/+ buffers/cache:		1420952	1681452			
Swap:	3121996	751000	2370996			

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	2848372	254032	7216	960	52652
-/+ buffers/cache:		2794760	307644			
Swap:	3121996	750016	2371980			

```

root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free

```

	total	used	free	shared	buffers	cached
Mem:	3102404	1483388	1619016	4820	148	46084
-/+ buffers/cache:		1437156	1665248			

(continues on next page)

(continued from previous page)

```
Swap:      3121996      749828      2372168

Used (used mem -buffers/cache + swap)
  before start: 2171952
  after start  : 3544776 = 1372824 kbytes, 1340 Mbytes / 13.41 MBytes/instance
  after kill   : 2186984 = 15032 kbytes, 15 Mbytes not recovered
```

OK, so yes, there is more overhead. We did just start 100 Python interpreters! As titled, this is dumb, and even if you use this dumb method, are you really going to have a hundred of them?...

Memory use - The sensible way

GNOME-Terminal:

```
idiot@village:~$ gnome-terminal &
idiot@village:~$ for i in {1..100} ; do gnome-terminal & done
```

```
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free          # Before
↪100
              total        used        free      shared    buffers     cached
Mem:          3102404      1490996      1611408         5344         172       47580
-/+ buffers/cache:      1443244      1659160
Swap:         3121996       749776      2372220
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free          # After
↪100
              total        used        free      shared    buffers     cached
Mem:          3102404      1878228      1224176         5344         172       47388
-/+ buffers/cache:      1830668      1271736
Swap:         3121996       733396      2388600
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free          # After
↪kill
              total        used        free      shared    buffers     cached
Mem:          3102404      1491888      1610516         4840         272       46088
-/+ buffers/cache:      1445528      1656876
Swap:         3121996       733240      2388756

Used (used mem -buffers/cache + swap)
  Before start: 2193020
  After start  : 2564064 = 371044 kbytes, 362 Mbytes / 3.59 MBytes/instance
  After kill   : 2178768 = 14252 kbytes, -13.92 Mbytes recovered (first process)
```

Terminator GTK2:

```
idiot@village:~$ terminator -g deletemeconfig &
idiot@village:~$ for i in {1..100} ; do terminator -g deletemeconfig -u & done
```

```
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free
              total        used        free      shared    buffers     cached
Mem:          3102404      1324492      1777912         4388         152       49688
-/+ buffers/cache:      1274652      1827752
Swap:         3121996       744528      2377468
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free
              total        used        free      shared    buffers     cached
Mem:          3102404      1652112      1450292         4756         860       49968
```

(continues on next page)

(continued from previous page)

```
-/+ buffers/cache: 1601284 1501120
Swap: 3121996 744224 2377772
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free
      total        used        free      shared    buffers     cached
Mem:   3102404    1305376    1797028        4236        124     42836
-/+ buffers/cache: 1262416    1839988
Swap:   3121996     744116    2377880

Used (used mem -buffers/cache + swap)
  before start: 2019180
  after start  : 2345508 = 326328 kbytes, 319 Mbytes / 3.16 MBytes/instance
  after kill   : 2006532 = 12648 kbytes, -12.35 Mbytes recovered (first process)
```

Terminator GTK3:

Not possible at the moment because the DBus interface still needs fixing.

So that one surprised me a bit. The fact that when using the single process we are **more** memory efficient. Python + 100 terminals is using <90% of the GNOME-Terminal + 100 terminals.

Some may think that this is something to do with the different version of the VTE widget, but hang on a second. In the dumb method GTK2 Terminator used **more** memory than GTK3. Once the DBus is fixed for GTK3 there could potentially be more savings.

“Python sucks!”

Yeah, whatever. The fact is that I’m a helluva lot more productive in Python than I ever was, am, or will be, in C. In my totally biased and uninformed opinion, I also think certain things are *much* easier to get working in Python because you can iterate faster. With the *Debugging* option to run an interactive terminal you even have the ability to try out ideas and explore the running instance directly. Results don’t get more immediate than that!

In summary

It’s a bit slower on startup, it takes a bit more memory, but that’s when you use the dumb method. In normal use, where you’re likely to be using the existing process to open a new window, it is for all practical purposes as fast as the compiled GNOME-Terminal. It may even (according to the last memory section) be a little lighter memory wise, and more obliging about giving it back!

I didn’t compare to things like xterm, because frankly we’re not aimed at the same people. Personally I’d rather have the more extensive features saving me *lots* of time over the course of the day when using it, than save a handful of seconds every few days when I restart it, or worrying about an extra 5 or 10 MBytes.

1.9.2 How do I...

... make Terminator work like Quake style terminals?

You can get close, but it isn’t a perfect recreation, as Terminator was not designed with this in mind, but improvements are always welcome!

- Window state: Hidden
- Always on top: On
- Show on all workspaces: On

- Hide on lose focus: On
- Hide from taskbar
- Window borders: Off (use Alt+click-drag and Alt+middle-click-drag to position and size window.)
- Set the Toggle window visibility shortcut to your preference

Note: It must be the first Terminator instance started, because at present only the first instance can bind to the Window toggle.

This will give you a terminal hidden at startup that appears with a keypress and disappears, either with another keypress, or losing focus. It will stay on top, and appear on whichever workspace you are on.

Something that we don't have is the slide in action of a true Quake style terminal. The terminal will simply flick into view, and flick out of view.

Warning: The Hide on lose focus option is problematic at this time. You will probably find it very frustrating.

... make the tabs shorter?

You can decrease the vertical padding using the following:

```
.terminator-terminal-window notebook tab {  
    padding: 0.1em;  
}
```

... make the active tab more distinctive?

You can set the color of both the inactive and active tabs

```
.terminator-terminal-window notebook tab {  
    background-color: #222;  
}  
  
.terminator-terminal-window notebook tab:checked {  
    background-color: #000;  
}
```

You can also set a border around the active tab to make it stand out

```
notebook tab {  
    /* background-color: #222; */  
    padding: 0.4em;  
    border: 0;  
    border-color: #444;  
    border-style: solid;  
    border-width: 1px;  
}  
notebook tab:checked {  
    /* background-color: #000; */  
    border-color: #76C802;  
}
```

You can also change the text color of the tab label

```
notebook tab:checked label {  
    color: #76C802;  
    font-weight: 500;  
}
```

For more ideas, you can look at: <https://www.preining.info/blog/2020/03/de-uglify-gtk3-tabs-of-terminals/>

or

http://blog.nabam.net/workstation/2017/09/15/terminator_tabs/



1.10 Getting involved

There are many ways to help out, and they don't all involve coding.

1.10.1 Translations

Sprechen Sie Deutsch?

Awesome! I've been getting my head around the whole translation bit (English monoglot I'm afraid), and as a result there has been a lot of churn in the translations. So what are you waiting for?

Speak some other language? Take a look at [the translation page](#) because you might just be the <insert language here> speaker that we're looking for.

1.10.2 Improve icons/artwork

OK, so while the main icon contributed by Cory Kontros is really good, my hacks of it are... not so good. I'm no artist, but I do appreciate them. So if you think you could apply some polish and a cohesive design to this manual's page header images, please, give it a go. It may only be to take the existing icon and to make it suck less.

The only thing I would ask is that you maintain the main icon as a base like I have done.

1.10.3 Terminator action shots

This one's just for "PR" purposes. I want to see famous/awesome people kicking ass *and* chewing bubble-gum with Terminator in the mix.

If you spot it in a TV show, movie, or a news article I want to know. Maybe you're even the famous/awesome person, in which case drop me a note.

It will warm the cockles of my heart to know that Terminator made life easier for people who do the really important stuff like discovering new particles (CERN? Hello?), boldly going (NASA? Come in Houston), or wrangle 2 more frames per second from Half-Life 3 (Valve? Confirmed?)

Here's the ones I've spotted and noted (I've seen quite a few others previously, but never thought to note them)

- **MindMaze - VR / mind-reading.** Visible in the background of the video, and in an image lower down the page. (The Verge)

- **Dual Universe - Sci-Fi MMORPG** Visible at 17:40 of the pitch video. (KickStarter)

1.10.4 Manual updates

This manual is a new endeavour to fully document all the nooks and crannies of Terminator. As such, there may be things that are missing, incorrect, not explained clearly, or need expanding.

Suggestions, or updates are welcome.

I had a little exposure at work to Sphinx, so I thought I'd dig in a bit deeper and learn a bit about it. So far I'm happy enough, so till further notice this manual will remain in this format.

If you're feeling like a loquacious polyglot you could attempt to translate the whole manual. So far I haven't tested it, but in principle, just do an export of the manual-gtk3 branch in Launchpad to a folder `manual-gtk3-<LANG>`, where `<LANG>` is the i18n language code. This is usually just the two or three letters of the language code, but sometimes has the region too... Or something else entirely in a couple of cases. A couple of examples:

```
pt           - Portugese
pt_BR        - Brazilian Portugese
ca           - Catalan
ca@valencia  - Catalan (Dialect specific to Valencia?)
```

Then just translate away, and take new screen grabs to replace the British English ones I've done. If someone was to make a serious effort to translate the manual, I'm sure we can get it included.

Note: If there are any Americans offended by correct spelling, they are more than welcome to create an Americanised version, and I'll relegate it to the `en_US` folder. The default will remain British English.

As there is only one language available, the Help shortcut will by default open:

```
http://terminator-gtk3.readthedocs.io/en/latest/index.html
```

The specifics of how readthedocs.io handle multiple languages are still a little hazy, but as I understand it uses the http headers passed by your browser, and directs you to the appropriate URL, for example:

```
http://terminator-gtk3.readthedocs.io/de/latest/index.html
```

In order to build the html for the manual, you must have sphinx and the sphinx_rtd_theme package installed. Ideally you will be using a distro with these packages available. An example would be Ubuntu 16.04 LTS:

```
sudo apt-get install python-sphinx python-sphinx-rtd-theme
```

This will take care of installing sphinx, the theme and it's dependencies.

Once a manual has a reasonable amount of translation we can look into adding it to the readthedocs.io website so it integrates properly.

Warning: This section may need updating if we do reach the point of adding another manual. It is at this point we will have to figure out the details of adding the translated manuals, and getting the user to the correct document.

1.10.5 Testing

Just use it, explore the features, and complain when they don't work.

We actually have quite a lots of outstanding issues, and in many cases I can't reproduce due to either lack of info, differences in environment, lack of information, or because the bug is so old the original raiser has moved on and not available for questions.

I'm particularly interested in cases where I can't even see that something is an issue, such as:

- *Right-to-Left* - I can force Terminator to Arabic, and everything flips around, but I have no idea if it looks "right" to a native speaker. Frankly it just looks *weird!*
- *HighContrast* - Again, I can switch to it, but perhaps I'm not appreciating the needs of that group.
- *Accessibility* - People using only a keyboard, or only a mouse, on-screen keyboards, text-to-speech, speech-to-text, and so on.

1.10.6 Bugs

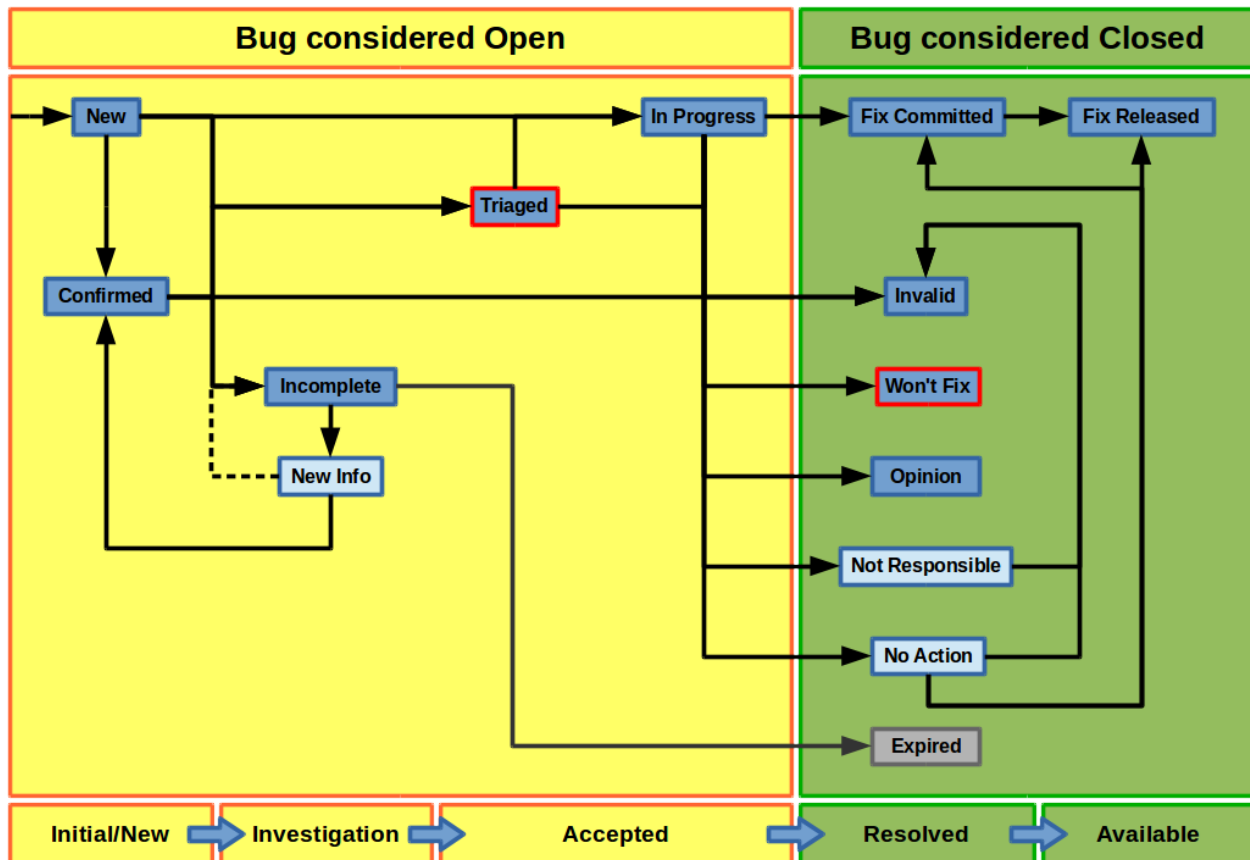
Bugs (and feature requests) are raised and dealt with in the Launchpad [bugs](#) page.

- **Fixing** - OK, so yeah, this is coding.
- **Reproduce and improving** - Sometimes bugs are lacking info to reproduce, or my system is too different. Or perhaps the original poster has moved on because we haven't fixed their pet peeve fast enough.
- **Triaging** - It's one of the less glamorous jobs, but someone's gotta do it. Shepherd bugs to the point where it has a priority, a milestone, reproduction steps, confirmation, submitted patches validated, and so on.
- **Raising** - If you have searched and cannot find your bug, you can raise a new one.

Feature requests are initially raised as bugs, and if it passes the rather undefined criteria, it will be marked as a **wishlist** item.

Bug handling

I have had one person (possibly others) who are hesitant to use the status' because they've been "told off" by the developers of other projects, and people/projects are often different in how they want to handle bugs. So, with that in mind, let me present my idea of how a bug should be handled. First a pretty picture:



So, the darker blue states are the ones available in Launchpad that can be manually set. The two marked with a red outline require bug supervisor role to set, which means a member of the Terminator team. The pale blue states are ones that I personally feel should be there, but are missing. I'll explain my intention with those in the appropriate sections below. The grey state is set automatically only, and cannot be set by anyone.

Initial/New

When you the user create a bug it goes into **New**. If another user clicks the *This bug affects you* link, this gets moved to **Confirmed**.

Investigation

If I (or indeed someone else) go to a *New* or *Confirmed* bug, and are unable to reproduce it then it will be marked **Incomplete**, and someone (preferably the original raiser, but it can be someone else affected) needs to revisit and provide the requested additional info. Ideally when that is added there would be a *New Info* (or similar) state that the user would set the bug to, and then the dashed line would be taken.

Because we don't have this state, we "skip" straight through and abuse the **Confirmed** state. Set the bug (back) to **Confirmed**, and assign the official tag *new-info*. Once the ticket is reviewed the tag will be removed, and a new state assigned, possibly even *Incomplete* again.

Note that I am aware of the two *Incomplete* options for with and without response, but the way it works is unclear, and I can't switch between the two myself, and it is not clear when Launchpad switches it. So, I'll be ignoring them and treating *Incomplete* as a single state.

Acceptance

At this point the bug should provide enough information to be reproducible. Only a supervisor can set an issue to **Triaged**. This state says, “Yes, the information provided either permits me to reproduce myself, or see what went wrong from provided logs, config, etc.” Typically they go here when I don’t have the time to start working on an immediate fix.

Alternatively I (or anyone) could start working on a bug. Ideally the issue should be set to **In Progress**, and assigned to the person picking it up. That way, two people don’t work on the same issue.

Sometimes, for trivial or interesting bugs, they might get looked at and fixed so fast that they skip all *Acceptance* categories, and go straight to one of the *Resolved* states.

Resolved

Fix Committed is for when a fix is pushed to the main Launchpad bazaar repository and typically I do this. If you create a contribution via a branch, and commit to your branch, do not set to this yourself. Instead associate the bug with the branch, and request a merge. When I do the merge I will also set the bug to *Fix Committed*.

An **Invalid** bug is usually because the user didn’t understand something, or it is in fact a support request.

Only a bug supervisor can set an issue to **Won’t Fix**. It is the supervisors way of ending the discussion when it is felt that a bug does not fit the projects plans, but someone can’t *let it go*.

Opinion is typically when the user and I have a different expectation about behaviour or a new feature, or I think that something being proposed would actually be a negative for Terminator. Unlike *Won’t Fix*, this can still be discussed within the ticket.

Not Responsible is our second missing virtual state. For me this is when, for example, an issue actually resides in *libvte*, or *GTK*. Again, there is a new official tag *not-responsible*, and the bug will actually end up set to *Invalid*.

The final virtual state is **No Action**, which is for various reasons. Sometimes other work has resolved an issue already, or the user was using an old version, and the fix is already in trunk or released. Again there is a new official tag *no-action*. These will then be put in one of the following: *Invalid*, *Fix Committed*, or *Fix Released*, depending on circumstance.

Our last Resolved state is the automatically set **Expired** one. This can only be set by Launchpad when a bug is set to *Incomplete*, and has been idle for 60 days. This is actually an on/off feature that is set by the project, and applies project-wide. Currently this is not active for Terminator bugs, but one day (when I get caught up, ha!) I might choose to turn this on.

Available

The last state is **Fix Released**, indicating that there has been a release containing a fix to the issue.

Of course this flow and states are not set in stone. A bug can be brought out of *Expired* if necessary. Or back from *In Progress* to *Confirmed* or *Triaged* if the assignee decides to stop working on the bug for some reason.

1.10.7 Plugins

Ahem... Yeah... More coding...

Some *Plugins* may have room for improvement, or perhaps you have an idea for a neat plugin no-one else has done.

1.10.8 Main Application Development

Oh come on... Coding? Again!

I see lots of people say how Terminator is really good, and it is, but like anything, it could be better!

To give an idea, as of March 2017, revision 1760, there are around 100 [wishlist items](#).

Note: Just because an item is marked as wishlist, it doesn't mean that a great deal of thought has been put into the appropriateness of the idea on my side. It may be impossible, or not a good fit, or just plain bat-sh!t crazy. If you want to pick up a wishlist item that looks like a lot of work (especially if it makes fundamental changes to the Terminator ethos) it's probably best to check first that your approach is good, and has a realistic chance of being merged.

Some of these wishlist items are also in my own text file of "Things to do" / "Big bag of crazy", which as of March 2017, revision 1760, looks like this:

```
Enhancements which may or may not have a wishlist item
=====
Completely new features
  Add libunity quicklist of saved layouts
    https://wiki.ubuntu.com/Unity/LauncherAPI#Python_Example
    http://www.techques.com/question/24-64436/Refreshing-of-Dynamic-Quicklist-
↳ doesn't work after initialization
    http://people.canonical.com/~dpm/api/devel/GIR/python/Unity-3.0.html
    Possibly use the progress bar and or counter for something too.
  Add an appindicator menu for launching sessions.
  If we can figure out how to do arbitrary highlighting, perhaps we can get a
↳ "highlight differences" mode like used to exist in ClusTerm.
    This could also be limited to highlighting diffs between those in the same
↳ group.
  Synchronised scroll based on groups
  Triggers (actions) based on regex for received text
  A "swap" mode for drag and drop
  Encrypted dumping/logging to disk
  Remotinator commands to modify debug level / class / funcs, and switch trace on/
↳ off
  Allow custom commands to only show on particular profiles

Search
  Might be able to misuse the ClusTerm method of overwriting to "highlight" (gtk2
↳ only)

Layouts
  Layout Launcher
    Could bind the shortcut as a global toggle to hide/show
    Could save
      window position/size
      hidden status
      always on top
      pin to visible workspace
  Layout needs to save/load more settings
    Per layout?
      Group mode status (all, group, off)
      Split to this group
      Autoclean groups
    Per window
      always on top
```

(continues on next page)

(continued from previous page)

```

        pin to visible workspace
    Per tab
    Per terminal
        Store the custom command and working directory when we load a layout, so
        ↪making small changes and saving doesn't lose everything.
        It could be possible to detect the current command and working directory
        ↪with psutil, but could be tricky. (i.e. do we ignore bash?)
        A per layout "save on exit" option to always remember last setup/positions etc.
        ↪Probably requires above to be done first.
        A per layout shortcut launch hotkey

Missing shortcuts:
    Just shortcut:
        Context menu (in addition to Windows menu button - not always available on
        ↪all keyboards)
        Group menu
        Open preferences
        Change group name
        Toggle titlebar visibility
        Equalise the splitters (siblings/siblings+children/siblings+parents,all)
        Zoom +receiver in/out/reset
        Zoom all in/out/reset
    New code:
        Open a shortcut help overlay (Ctrl-F1?)
        Insert tab text, titlebar text, group name value into terminal(s)
        Last terminal / tab / window (again to jump back to original) #1440049
        Limit broadcast group/all to current tab / window (toggle)
        Broadcast temporarily off when maximised or zoomed to single term (toggle)

Titlebar
    Add large action/status icons for when titlebar is bigger and/or HiPDI
    Improve the look/spacing of the titlebar, i.e. the spacing around/between elements

Tabs
    right-click menu replicating GNOME-Terminals (move left/right, close, rename)

Menus
    Add accelerators (i.e. "Shift+Ctrl+O") might look too cluttered.

Preferences
    Profiles
        Add preselection to the profile tab
        Add filter to font selector to only show fixed width fonts
    Layouts
        Have changing widgets depending on what is selected in the tree
        Terminal title editable
        Button in prefs to duplicate a layout
        Ordering in list
        Working directory - add dialog too, see http://stackoverflow.com/questions/10868167/make-filechooserdialog-allow-user-to-select-a-folder-directory
        ↪10868167/make-filechooserdialog-allow-user-to-select-a-folder-directory
    Keybindings
        Add a list of the default keybindings to the Preferences -> Keybindings
        ↪window?
        Option for close_button_on_tab in prefs. (needs tab right-click menu first
        Option to rebalance siblings on a split (don't think children or ancestors make
        ↪sense)
        Figure out how to get the tree view to jump to selected row for prefseditor

```

(continues on next page)

(continued from previous page)

Plugins

Give plugins ability to register shortcuts
 Custom Commands is blocking, perhaps make non-blocking

Drag and Drop

LP#0768520: Terminal without target opens new window
 LP#1471009: Tab to different/new window depending on target

Major architectural

Improve Dbus interface, add coordination between sessions, i.e.:
 multiple Dbus ports? register them with a master Dbus session, be able to
 ↪query these, etc
 be able to drive them more with command line commands, and not just from
 ↪within own shell
 Remotinator improvements
 Abstract out the session/layout allowing multiple logical layouts in the same
 ↪process to reduce resource used
 This is a big piece of work, as a lot of the Terminator class would need
 ↪seperating out.
 Hide window should find the last focussed window and hide that. Second hit
 ↪unhides and focusses it
 Add a power hide to hide all of shortcut bound instances windows
 Use the dbus if available to hide the current active window, then unhide it
 ↪on second shortcut press
 If the dbus is available:
 The hide will go to the focussed instance, instead of the first to grab
 ↪the shortcut
 Add a super power hide to hide all Terminator windows
 In both cases a second shortcut unhides whatever was hidden

Split with command / Inherit command/workdir/groups etc

Somehow make Layout Launcher, Preferences, & poss. Custom Commands singleton/borg
 ↪(possibly use dbus)

When in zoomed/maximised mode

Perhaps the menu could contain a quick switch sub menu, rather than having to
 ↪Restore, right-click, maximise
 Shortcuts for next/prev,up/down/left/right, etc. How should they behave

All non main windows to be changed to glade files

For me the two different sets of next/prev shortcuts are a bit of a mystery.

Let window title = terminal titlebar - perhaps other combos. Some kind of %T %G %W
 ↪substitution?

So as you can see, still lots of room for improvements, and plenty of ideas if you are trying to find small starter tasks.

1.10.9 GTK2 Maintenance

The GTK2 version of Terminator has gone into deprecated mode as far as I'm concerned. If someone wants to pick up the back-porting of fixes they can contact me, and I'll give them commit access on the GTK2 branch. It is better that any focus I can spare is spent on the GTK3 version.

1.10.10 GTK3 Port

Last coding one, I promise!

After some sterling work by Egmont Koblinger, one of the VTE developers, he came up with a very large patch for rudimentary GTK3 support. A number of things were incomplete or broken, but it got it far enough along that it was no longer an insurmountable cliff face.

After that I resolved to port fixes and features between the two versions. For a time I managed this, but it got to the point where the GTK3 port was better and more stable than the old GTK2 code, due to VTE and GTK improvements that added features, and seems to have fixed many (if not all) of the segfault crashes that would happen within the GTK2 libraries.

The port is pretty much complete. I *hope* we've fixed any regressions and critical issues. There are a few minor tasks that don't seem to be urgent as far as I can see listed below. Feel free to look into these. For the record, as of March 2017, with the [gtk3 branch](#) at revision around 1760, these are the outstanding items:

```
Outstanding GTK3 port tasks/items/reviews/reimplementations etc.
=====
[    ] Need to go through all the Gtk.STOCK_* items and remove. Deprecated in 3.10.
      Very low priority as won't be problem till GTK 4.0 (hopefully!)
[    ] Homogeneous_tabbar removed? Why?
[    ] terminal.py:on_vte_size_allocate, check for self.vte.window missing.
↳Consequences?
[    ] terminal.py:understand diff in args between old fork and new spawn of bash.
↳Consequences?
[    ] VERIFY(9)/FIXME(6) FOR GTK3 items to be dealt with
[    ] Get the debian build stuff up to date and aligned with the GTK2 where
↳appropriate
[    ] LP#1521280 - Reimplement utmp option (for turning off somehow)
```

Now the GTK3 port is done there is also a long overdue port to Python3, especially in light of some distributions trying to eliminate Python2 from the base installs. Yes, Python2 will be with us for a long time yet, but this should serve as a warning.

I also have some new items specifically for the GTK3 branch which I'm still thinking about, but I'm not ready to declare. I suspect I might get a bit of unwanted pressure if I were to mention these, so for now they are under NDA.

1.10.11 Docs for Devs

Here is a list of some useful sets of documentation collected together for convenience:

General	
Python	https://docs.python.org/release/2.7/index.html
GNOME Dev. Center	https://developer.gnome.org/
Bazaar DVCS	http://doc.bazaar.canonical.com/en/
Launchpad Help	https://help.launchpad.net/
GTK 3	
GObject Introspection	https://wiki.gnome.org/Projects/GObjectIntrospection
GObject	https://developer.gnome.org/gobject/stable/
PyGObject Introspection	https://wiki.gnome.org/Projects/PyGObject
PyGObject	https://developer.gnome.org/pygobject/stable/
Many PIGO autodocs	http://lazka.github.io/pgi-docs/
GDK3 Ref. Manual	https://developer.gnome.org/gdk3/stable/
GTK3 Ref. Manual	https://developer.gnome.org/gtk3/stable/index.html
Python GTK+ 3 Tutorial	http://python-gtk-3-tutorial.readthedocs.org/en/latest/index.html
VTE for GTK 3	https://developer.gnome.org/vte/0.38/

Bibliography

[VS] **VTE Shortcuts:** Default actions from VTE that are not configurable.

[XL] **X Lines:** Where X may vary depending on distribution. On mine it is 4.

[TS] **Terminator Shortcuts:** Additional movement options from Terminator that are configurable.

[MS] **Masked Shortcuts:** VTE provides default shortcuts for line up/down, on `Shift+Ctrl+Arrow Up/Dn`, but they are masked by shortcuts for resizing terminals. You can disable or reassign the resizing shortcuts to regain access to the VTE default.