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

A quick word on style. I have none. I have an casual way of talking and writing, and a strange sense of humour. I also tend to use random pop-culture references a lot, but often in ways that make no sense. If this is a problem you are entitled to a refund of the money you paid to me to write this...
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:

- Arrange terminals in a grid
- Tabs
- Drag and drop re-ordering of terminals
- Lots of keyboard shortcuts
Terminator Documentation, Release 1.0 alpha

- Save multiple layouts and profiles via GUI preferences editor
- Simultaneous typing to arbitrary groups of terminals

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.

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.

Chapter 1. What is Terminator
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.

## Document history

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

All contributions and improvements are welcome.

<table>
<thead>
<tr>
<th>Updated for</th>
<th>Date</th>
<th>Author / Editor</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.97, r1598</td>
<td>2015-08-07</td>
<td>Stephen Boddy</td>
<td>Initial creation</td>
</tr>
<tr>
<td>0.97, r1621</td>
<td>2015-08-21</td>
<td>Stephen Boddy</td>
<td>Minor changes/corrections Added links to dev docs</td>
</tr>
<tr>
<td>0.98</td>
<td>2015-08-26</td>
<td>Stephen Boddy</td>
<td>Stick a fork in it, it’s done</td>
</tr>
<tr>
<td>0.98, r1663</td>
<td>2015-09-30</td>
<td>Stephen Boddy</td>
<td>Add the new PuTTY paste mode Add new Remotinator commands FAQ for other Terminator Add Bug handling flow Minor changes/corrections</td>
</tr>
<tr>
<td>0.98, r1667</td>
<td>2015-10-01</td>
<td>Stephen Boddy</td>
<td>Add the new Smart copy mode</td>
</tr>
</tbody>
</table>

**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.

## 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 first start Terminator you will get a default, minimal window, looking something 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) can be turned off, although you do lose ease-of-access to some of Terminators more powerful features if you do.

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 over a terminal, and will let you access all the settings, profiles, shortcuts and configurations. It is however kept brief to avoid the mega-menus that sometimes grow unchecked.

2. The **Grouping Menu** - This is reached with a 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**.

The **Context Menu**

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>Shift+Ctrl+C</td>
</tr>
<tr>
<td>Paste</td>
<td>Shift+Ctrl+V</td>
</tr>
</tbody>
</table>
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 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:

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 manager. 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 terminal programs. 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 Ctrl+wheelup/wheeldown 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.

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.

Navigating around

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Options</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move focus</td>
<td>Up, Down, Left, Right</td>
<td>Alt+&lt;Arrow&gt;</td>
</tr>
<tr>
<td>Cycle to terminal</td>
<td>Next, Prev</td>
<td>(Shift+)Ctrl+Tab</td>
</tr>
<tr>
<td>Focus to terminal</td>
<td>Next, Prev</td>
<td>Shift+Ctrl+1/9</td>
</tr>
<tr>
<td>Switch to tab #</td>
<td>1 to 10</td>
<td>Ctrl+PgUp/PgDn</td>
</tr>
<tr>
<td>Switch tab</td>
<td>Previous, Next</td>
<td>Ctrl+PgUp/PgDn</td>
</tr>
<tr>
<td>Context menu</td>
<td></td>
<td>Menu Key</td>
</tr>
<tr>
<td>Help¹</td>
<td></td>
<td>F1</td>
</tr>
</tbody>
</table>

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

Note: For me the two different sets of next/prev shortcuts are a bit of a mystery. Something to look into.

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 the 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
      
      The description might be different depending on the type of the item (see below).
    
    - *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:

<table>
<thead>
<tr>
<th>URL</th>
<th>Note</th>
<th>Made up example, Don’t use!</th>
</tr>
</thead>
<tbody>
<tr>
<td>news://user@host:port/path</td>
<td>news://steve@news.example.org:1234/announce</td>
<td></td>
</tr>
<tr>
<td>telnet://user@host:port/path</td>
<td>telnet://steve@insecure.example.org:1234</td>
<td></td>
</tr>
<tr>
<td>nntp://user@host:port/path</td>
<td>nntp://steve@news.example.org:1234/announce</td>
<td></td>
</tr>
<tr>
<td>file://user@host:port/path</td>
<td>file://steve@localhost/var/log/syslog file:///var/log/syslog</td>
<td></td>
</tr>
<tr>
<td><a href="http://user@host:port/path">http://user@host:port/path</a></td>
<td>+ <a href="https://http://steve@www.example.org/index.html">https://http://steve@www.example.org/index.html</a></td>
<td></td>
</tr>
<tr>
<td>ftp://user@host:port/path</td>
<td>+ ftps://ftp://steve@ftp.example.org/var/log/</td>
<td></td>
</tr>
<tr>
<td>webcal://user@host:port/path</td>
<td>webcal://steve@webcal.example.org/today</td>
<td></td>
</tr>
<tr>
<td>wwwhostname.domain:port/path</td>
<td>www-server.example.org/index.html <a href="http://www.example.org">www.example.org</a></td>
<td></td>
</tr>
<tr>
<td>ftphostname.domain:port/path</td>
<td>ftp-server.example.org/var/log/ ftp.example.org</td>
<td></td>
</tr>
</tbody>
</table>

### VoIP

<table>
<thead>
<tr>
<th>callto:user:number@path</th>
<th>callto:steve:0123456789@not/sure/here</th>
</tr>
</thead>
<tbody>
<tr>
<td>h323:user:number@path</td>
<td>h323:steve:0123456789@not/sure/here</td>
</tr>
<tr>
<td>sip:user:number@path</td>
<td>sip:steve:0123456789@not/sure/here</td>
</tr>
</tbody>
</table>

### E-Mail

<table>
<thead>
<tr>
<th>mailto:name@host</th>
<th><a href="mailto:steve@example.org">mailto:steve@example.org</a></th>
</tr>
</thead>
</table>

### 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.

---

**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 three 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:
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.

**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:
Terminator Documentation, Release 1.0 alpha

### Terminator Documentation, Release 1.0 alpha

<table>
<thead>
<tr>
<th>Action</th>
<th>Options</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>New instance²</td>
<td></td>
<td>Super+I</td>
</tr>
<tr>
<td>New window</td>
<td></td>
<td>Shift+Ctrl+I</td>
</tr>
<tr>
<td>New Tab</td>
<td></td>
<td>Shift+Ctrl+T</td>
</tr>
<tr>
<td>Split terminal</td>
<td>Horizontally, Vertically</td>
<td>Shift+Ctrl+C/E</td>
</tr>
<tr>
<td>Hide window³</td>
<td></td>
<td>Shift+Ctrl+Alt+A</td>
</tr>
<tr>
<td>Close window</td>
<td></td>
<td>Shift+Ctrl+Q</td>
</tr>
<tr>
<td>Close terminal</td>
<td></td>
<td>Shift+Ctrl+W</td>
</tr>
<tr>
<td>Toggle fullscreen</td>
<td></td>
<td>F11</td>
</tr>
<tr>
<td>Resize terminal</td>
<td>Up, Down, Left, Right</td>
<td>Shift+Ctrl+&lt;Arrow&gt;</td>
</tr>
<tr>
<td>Rotate terminals</td>
<td></td>
<td>Shift+Ctrl+&lt;Arrow&gt;</td>
</tr>
<tr>
<td>Move Tab</td>
<td>Left, Right</td>
<td>Shift+Ctrl+PgUp/PgDn</td>
</tr>
<tr>
<td>Zoom terminal</td>
<td></td>
<td>Shift+Ctrl+Z</td>
</tr>
<tr>
<td>Maximise terminal</td>
<td></td>
<td>Shift+Ctrl+X</td>
</tr>
</tbody>
</table>

### Resetting the terminal

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Shift+Ctrl+R</td>
</tr>
<tr>
<td>Reset + Clear</td>
<td>Shift+Ctrl+G</td>
</tr>
</tbody>
</table>

### The scrollbar and scrollback 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 scrollback buffer with more flexibility:

<table>
<thead>
<tr>
<th>Action</th>
<th>Options</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle scrollbar</td>
<td></td>
<td>Shift+Ctrl+S</td>
</tr>
<tr>
<td>Page [VS]</td>
<td>Up, Down</td>
<td>Shift+PgUp/PgDn</td>
</tr>
<tr>
<td>X Lines [VS] [XL]</td>
<td>Up, Down</td>
<td>wheelup/wheeldown</td>
</tr>
<tr>
<td>Page [TS]</td>
<td>Up, Down</td>
<td></td>
</tr>
<tr>
<td>Half page [TS]</td>
<td>Up, Down</td>
<td></td>
</tr>
<tr>
<td>Line [TS] [MS]</td>
<td>Up, Down</td>
<td></td>
</tr>
</tbody>
</table>

### Search the buffer

It is possible to search the buffer, although at this time there is a limitation that the found string is not highlighted.

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin search</td>
<td>Super+Ctrl+F</td>
</tr>
</tbody>
</table>

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

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

### Zooming the terminal

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target in</td>
<td>Ctrl++/wheelup</td>
</tr>
<tr>
<td>Target out</td>
<td>Ctrl+-/wheeldown</td>
</tr>
<tr>
<td>Target reset</td>
<td>Ctrl+0</td>
</tr>
<tr>
<td>+Receivers in</td>
<td>Shift+Ctrl+wheeldown</td>
</tr>
<tr>
<td>+Receivers out</td>
<td>Shift+Ctrl+wheeldown</td>
</tr>
<tr>
<td>+Receivers reset</td>
<td>N/A (TBD, plus in/out)</td>
</tr>
<tr>
<td>All in</td>
<td>Super+Ctrl+wheeldown</td>
</tr>
<tr>
<td>All out</td>
<td>Super+Ctrl+wheeldown</td>
</tr>
<tr>
<td>All reset</td>
<td>N/A (TBD, plus in/out)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Mouse</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window title</td>
<td>N/A</td>
<td>Ctrl+Alt+W</td>
</tr>
<tr>
<td>Tab title</td>
<td>double-click tab</td>
<td>Ctrl+Alt+A</td>
</tr>
<tr>
<td>Terminal title</td>
<td>double-click titlebar</td>
<td>Ctrl+Alt+X</td>
</tr>
</tbody>
</table>

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

Insert terminal number

These shortcuts let you enumerate your terminals. It is 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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert terminal number</td>
<td>Super+1</td>
</tr>
<tr>
<td>Insert zero padded terminal number</td>
<td>Super+0</td>
</tr>
</tbody>
</table>

These actions can also be done from The Grouping Menu.

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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next profile</td>
<td></td>
</tr>
<tr>
<td>Previous profile</td>
<td></td>
</tr>
</tbody>
</table>

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.

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.
Global

These settings are defaults, but some of them can be overridden by 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)

Window attempts to remain on top.
Show on all workspaces (default: off)

The focused window will follow if you switch to a different virtual desktop.

Hide on lose focus (default: off)

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

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.

DBus server (default: on)

If a TerminatorDBus 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 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.

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.

**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

**Terminator seperator size** (default: -1)

This is the width in pixels, and can range from -1 to 5. The value of -1 will take the default size from the system theme.

**Unfocused terminal font brightness** (default: 0.8)

Terminals that do not currently have the focus will can be dimmed to aid focus. The value can range from 0 (invisible) to 1 (full brightness)

**Window borders** (default: on)

The window manager frame can be 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)

Tabs will have equal widths

**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.

### 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*.

<table>
<thead>
<tr>
<th></th>
<th>Focused</th>
<th>Inactive</th>
<th>Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font colour</td>
<td>#FFFFFF</td>
<td>#000000</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>Background</td>
<td>#C80003</td>
<td>#C0BEF</td>
<td>#0076C9</td>
</tr>
</tbody>
</table>

**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.

**Warning:** With newer versions of some of the underlying libraries, using a bitmap font can cause the Preference window to crash. If you find this happening, you will need to either remove the configuration from the Terminator configuration file or you will need to change your system settings. Please see LP#1294699 for more information.

### 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. Bear in mind that some of these differences are due to changes in the underlying VTE widget between GTK2 and GTK3.

Below we will go through each pane, and highlight and explain differences between Terminator with GTK2 and GNOME Terminal with GTK3.
One key difference is that we have a sidebar to the left listing the available Profiles, as opposed to GNOME Terminator, where the list is a separate window launched from the menu bar.

**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.

---

**Warning**: With newer versions of some of the underlying libraries, using a bitmap font can cause the Preference window to crash. If you find this happening, you will need to either remove the configuration from the Terminator configuration file or you will need to change your system settings. Please see LP#1294699 for more information.

**Allow bold text** (default: on)

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

**Anti-alias text** (default: on) †Not in GNOME Terminal
In Terminator you can turn the font smoothing off. This is no longer possible in GNOME Terminator.

**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.

**Select-by-word characters** (default: -A-Za-z0-9,./%&#:_)

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: #AAAAAA)

The colour of the cursor.

**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.

**Show menubar by default in new terminals**

Terminator doesn’t use a traditional menu bar.
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.

Use custom default terminal size

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

Command

![Terminator Preferences](image)

Run commands as a login shell (default: off)

Force the command to run as a login shell.

Update login records when command is launched (default: on)

Updates login records when a new shell is opened.

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.

### 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.

### Not in Terminator

**Initial title**

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

**When terminal commands set their own titles**

Terminator doesn’t have this setting.
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. Not clear at this time where exactly these come from. Differences in the GTK2, GTK3 and GNOME Terminal.

**Built-in schemes** (default: Grey on black)

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

The list seems to be dynamic and vary depending on the system, with the addition of *Custom* which allows setting the colours as desired.

**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. Again there are unexplained differences between Terminator and GNOME Terminal.

The default here may be system dependant, with Ambience being an Ubuntu colour scheme.

**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.
Background

Solid colour (default: active)

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

Background image (default: inactive)

Background will be an image. There is no scaling done.

Image file (default: inactive, None)

If Background image is set, then the image to use can be selected here.

Background image scrolls (default: inactive, on)

If the Background image is set, then setting this to on will cause the background image to change as the window moves. This is a form of fake transparency.

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 or image background (default: 0.5)

For Background image and 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

Scrollbar (default: 500 lines)

How many lines to keep before discarding.

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.

**Scroll on output** (default: on)

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

**Scroll on keystroke** (default: on)

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

**Use keystrokes to scroll on alternate screen** (default: on)

Ummmm... I don’t know. Alternate screens are a bit of a mystery to me.

**Compatibility**

**Backspace key generates** (default: ASCII DEL)

Change behaviour of the Backspace key.

- **Automatic**

![Terminator Preferences](image)

**Encoding**

Default: **Unicode UTF-8**
• \textit{Control-H}
• \textit{ASCII DEL}
• \textit{Escape sequence}

\textbf{Delete key generates} (default: Escape sequence)

Change behaviour of the Delete key.
• \textit{Automatic}
• \textit{Control-H}
• \textit{ASCII DEL}
• \textit{Escape sequence}

\textbf{Reset Compatibility Options to Defaults}

Sets the two previous items back to their defaults.

\textbf{Encoding}

\textbf{Default} (default: Unicode UTF-8)

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

Once a layout is highlighted, it’s 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.

**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.
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. 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.

---

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.

Keybindings

![Terminator Preferences](image)

1.4. Preferences Window
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).

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.

Warning: For some reason clicking on the text label of a plugin does not just select the item, but actually toggles the active/inactive status. This does not happen in the experimental GTK3 version of Terminator, and is a bit of a mystery.
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. There’s also a mysterious button... I wonder what happens when I press it?

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. Here it is 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.

---

**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:

<table>
<thead>
<tr>
<th>Action</th>
<th>Mouse</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open the Layout Launcher</td>
<td>N/A</td>
<td>Alt+L</td>
</tr>
<tr>
<td>Move Up/Down list</td>
<td>click</td>
<td>〈Up/Down Arrow〉</td>
</tr>
<tr>
<td>Launch a layout</td>
<td>double-click</td>
<td>Return</td>
</tr>
</tbody>
</table>

### The Grouping Menu

#### Manipulating terminal groups

Grouping, as the name suggests, allows grouping of terminals allowing actions to 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:

![Screen Shot]

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:

![Screen Shot]

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):

<table>
<thead>
<tr>
<th>Action</th>
<th>Mouse</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group menu popup</td>
<td>click</td>
<td>(TBD)</td>
</tr>
<tr>
<td>Edit group for terminal</td>
<td><em>Shift+click</em></td>
<td>(TBD)</td>
</tr>
<tr>
<td>Edit group for all terminals in the “group”</td>
<td><em>Shift+Super+click</em></td>
<td>(TBD?)</td>
</tr>
<tr>
<td>Toggle to current terminals group¹</td>
<td><em>Ctrl+click</em></td>
<td></td>
</tr>
<tr>
<td>Toggle “group” to current terminals group¹</td>
<td><em>Ctrl+Super+click</em></td>
<td></td>
</tr>
</tbody>
</table>

**Warning:** The terminals with no named group are also considered a 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

¹ These shortcuts will only work if you use them on terminals that are *not* the current terminal.
terminal with that group is closed. If off groups will remain in the list until the application exits, or the option is 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.

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group all</td>
<td>Super+G</td>
</tr>
<tr>
<td>Ungroup all</td>
<td>Shift+Super+G</td>
</tr>
<tr>
<td>Group tab</td>
<td>Super+T</td>
</tr>
<tr>
<td>Ungroup tab</td>
<td>Shift+Super+T</td>
</tr>
<tr>
<td>Group all toggle</td>
<td></td>
</tr>
<tr>
<td>Group tab toggle</td>
<td></td>
</tr>
</tbody>
</table>

**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 an image of each of these modes, with test typed into the current broadcasting terminal:

- **Broadcast off**
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.

- **Broadcast all**
Here you can see that all terminals, including those in other groups, or with no group, receive the input.

- **Broadcast group** (default mode; can be changed in the *Global* tab of *Preferences Window*.)
Here you can see that only the terminal(s) in the same group in the same group as the broadcaster receive input. 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.

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:

<table>
<thead>
<tr>
<th>Action</th>
<th>Default Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast off</td>
<td>Alt+0</td>
</tr>
<tr>
<td>Broadcast all</td>
<td>Alt+A</td>
</tr>
<tr>
<td>Broadcast group</td>
<td>Alt+G</td>
</tr>
</tbody>
</table>

**Warning:** It has become apparent that IBus causes problems with broadcast input. When multiple keystrokes are used to *compose* a letter, such as the `AltGr`+`[` followed by `a`, `o`, `u` keys to get ä, ö, ü characters for the German language.

Instead of getting your composed letter in all receivers, you will only get the composed letter in the current terminal. 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 IBus. Killing it will temporarily fix the problem. There is a fix for IBus and VTE, but only in the newer builds of VTE. For that we would need the experimental GTK3 version of Terminator.
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.

_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 made apparent by underlining the item when the mouse hovers over it.

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.

---

**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.

**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:

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

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

---

**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:

```plaintext
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.

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.
In absolutely no order at all...

**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/ilgarm/terminator_plugins**
- clone_session: Split and clone ssh session

**https://github.com/arnaudh/terminator-plugins**
- open_any_file_plugin: Open any file with it's default application

**https://github.com/dr1s/terminator-plugins**
- cluster_connect: A way to connect to multiple machines as a cluster

**https://github.com/mchelem/terminator-editor-plugin**
- editor_plugin: Click on file:line style links to launch a text editor

**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/choffee/terminator-plugins**
- searchplugin: Search Google for the selected text in a terminal

**https://github.com/papajoker/editor_terminator**
- editor_plugin: Another text editor launcher

**https://github.com/papajoker/git_terminator**
- git_plugin: adds commands for git when it detects a .git folder

**https://github.com/iambibhas/terminator-plugins**
- hastebin: Uploads selected text to Hastebin and opens browser on it

**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/mikeadkison/terminator-google**
- google: Another google-the-text plugin

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

**https://github.com/Theer108/colorize**
- colorize: Colour titlebar of each terminal separately

**https://github.com/julius/clisnips**
- clisnips: Snippets for the command line.

**https://github.com/GratefulTony/TerminatorHostWatch**
• hostWatch: Attempts to figure out your current host, and apply a certain theme.

https://github.com/kmoppel/dumptofile
  • dump_to_file: Dump console contents to a text file.

https://bitbucket.org/pgularski/terminator-plugins
  • show_titlebar: Menu item to show/hide the titlebar.
  • searchplugin: Yup, another Googler.

https://bitbucket.org/johnsanchezc/terminator-applauncher
  • applauncher: A launcher/set-up tool (which is built-in now, possibly redundant)

  • 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)

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.

Creating your own plugins

Note: The following guide is initially sourced from a tutorial written by Chris Jones back in April 2010. I’m
reproducing it here as a precaution, although I don’t expect the 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 announce-
ment/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:

```python
class LaunchpadBugURLHandler(plugin.URLHandler):
capabilities = ['url_handler']
handler_name = 'launchpad_bug'
match = r'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:

```python
import terminatorlib.plugin as plugin
from terminatorlib.config import Config
available = ['CustomCommandsMenu']
class CustomCommandsMenu(plugin.MenuItem):
capabilities = ['terminal_menu']
config = None

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

Advanced Usage

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

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 --help option just to aid clarity.

General options

- `--help` Show a help message and exit
- `--version` Display program version
- `--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.

-c CLASSNAME, --classname=CLASSNAME Set a custom name (WM_CLASS) property on the window
-r ROLE, --role=ROLE Set a custom WM_WINDOW_ROLE property on the window

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
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
```

Debugging

There is inbuilt debugging features in Terminator. The simplest is to start Terminator from another terminal with the option `-d`. This will dump many debug statements to the launch 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:
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 problematic. Any attempt to use `Remotinator` will hang the main application.

The debug options and their usage are detailed [here](#).
**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](http://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

Running a single instance of Terminator will cause problems if you are trying to launch a layout, when an instance is already running, so when configuring a program, script or menu item to launch a layout, you need to remember to include the `-u` option that will disable the DBus for that instance. *The Layout Launcher* does this for you, and as a result any launched layout is running without DBus, and cannot be controlled with DBus.

**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.

**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 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.

The following commands are currently enabled:

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_tab&quot;</td>
<td>Get the UUID of a parent tab</td>
</tr>
<tr>
<td>get_tab_title&quot;</td>
<td>Get the title of a parent tab</td>
</tr>
<tr>
<td>get_terminals</td>
<td>Get a list of all terminals</td>
</tr>
<tr>
<td>get_window&quot;</td>
<td>Get the UUID of a parent window</td>
</tr>
<tr>
<td>get_window_title&quot;</td>
<td>Get the title of a parent window</td>
</tr>
<tr>
<td>hsplit&quot;</td>
<td>Split the current terminal horizontally</td>
</tr>
<tr>
<td>new_tab&quot;</td>
<td>Open a new tab</td>
</tr>
<tr>
<td>new_window</td>
<td>Open a new window</td>
</tr>
<tr>
<td>vsplit&quot;</td>
<td>Split the current terminal vertically</td>
</tr>
</tbody>
</table>

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

1 These entries require either `TERMINATOR_UUID` environment var, or the `-uuid` option must be used.
Note: Because a layout (unless launched from the command line as as the first instance) is normally launched as a separate instance requiring the `-u`, Remotinator will not work within layouts. As mentioned in the `DBus` section, this has the potential to be improved upon.

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

Frequently Asked Questions

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

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.

GNOME-Terminal:

```
for i in {1..30} ; do gnome-terminal --profile=Quickexit; done
```

```
time
real 0m10.606s
```

Terminator:

```
for i in {1..30} ; do terminator -g deletemeconfig -p Quickexit; done
```

```
time
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:
Terminator Documentation, Release 1.0 alpha

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

Terminator:

```
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:**

```
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: 2399072 763332
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:**

```
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
```

1.9. Frequently Asked Questions
Terminator Documentation, Release 1.0 alpha

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:

```bash
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
```

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

```bash
gnome-terminal &
for i in {1..100}; do gnome-terminal & done
```

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

```
total used free shared buffers cached
Mem: 3102404 1372824 1619016 4820 148 46084
-/+ buffers/cache: 1437156 1665248
Swap: 3121996 749828 2372168
```

Used (used mem -buffers/cache + swap)
before start: 2186984
after start : 3544776 = 15032 kbytes, 15 Mbytes not recovered

Chapter 1. What is Terminator
### Terminator GTK2:

```
terminator -g deletemeconfig &
for i in {1..100} ; do terminator -g deletemeconfig -u & done
```

```
root@pinpoint:~# sync && echo 3 > /proc/sys/vm/drop_caches && free # After
-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
-kill
    total    used    free    shared    buffers    cached
Mem:       3102404    1491888   1610516       4840       272     46088
-/+ buffers/cache:       1445528   1656876
Swap:        3121996    733240   2388756
```

### 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.

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.
Getting involved

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

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 https://translations.launchpad.net/terminator because you might just be the <insert language here> speaker that we’re looking for.

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 manuals 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.

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)

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 a checkout of the trunk, and do a full copy of the `doc/manual` folder to `doc/manual_XXXX` where `XXXX` 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>pt_BR</td>
<td>Brazilian Portuguese</td>
</tr>
<tr>
<td>ca</td>
<td>Catalan</td>
</tr>
<tr>
<td>ca@valencia</td>
<td>Catalan (Dialect specific to Valencia?)</td>
</tr>
</tbody>
</table>

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.

The Help shortcut checks the `LANGUAGE` environment variable, and tries those folders in order, before falling back gracefully to the default manual, which is British English anyway:

```
LANGUAGE=en_GB:en
```

So this is going to try:

- `html_en_GB` - the non-existant British English folder
- `html_en` - the non-existant generic English folder
- `html` - the default document that happens to be in British English

**Note:** Although the source is in a folder beginning with `manual`, that gets replaced with `html` for installation.

**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.

In order to create the html for the manual, you must have the `sphinx_rtd_theme` package installed. This does not appear to be packaged for Ubuntu 14.04 LTS as far as I can tell. This means you must install it using the `pip` tool. This may need installing on your system too with:

```
sudo apt-get install python-pip
```

Once that is installed you can install the theme with:

```
sudo pip install sphinx-rtd-theme
```

This will take care of installing the theme and it’s dependencies.

**Warning:** On Ubuntu this also installed a newer version of `Sphinx` under the `/usr/local` folder. This caused a bit of confusion at one point, so be aware.

## Testing

Just use it, explore the features, and complain when they don’t work.
We actually have quite a lot 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.

## 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.

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.

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.

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 October 2015, revision 1663, there are 86 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 October 2015, revision 1663, looks like this:

<table>
<thead>
<tr>
<th>Enhancements which may or may not have a wishlist item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completely new features</td>
</tr>
<tr>
<td>Add libunity quicklist of saved layouts</td>
</tr>
<tr>
<td><a href="https://wiki.ubuntu.com/Unity/LauncherAPI#Python_Example">https://wiki.ubuntu.com/Unity/LauncherAPI#Python_Example</a></td>
</tr>
<tr>
<td>Possibly use the progress bar and or counter for something too.</td>
</tr>
<tr>
<td>Add an appindicator menu for launching sessions.</td>
</tr>
<tr>
<td>If we can figure out how to do arbitrary highlighting, perhaps we can get a &quot;highlight differences&quot; mode like used to exist in ClusTerm.</td>
</tr>
<tr>
<td>This could also be limited to highlighting diffs between those in the same group.</td>
</tr>
<tr>
<td>Synchronised scroll based on groups</td>
</tr>
<tr>
<td>Triggers (actions) based on regex for received text</td>
</tr>
<tr>
<td>A &quot;swap&quot; mode for drag and drop</td>
</tr>
<tr>
<td>Encrypted dumping/logging to disk</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Might be able to misuse the ClusTerm method of overwriting to &quot;highlight&quot; (gtk2 only)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layouts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layout Launcher</td>
</tr>
<tr>
<td>Could bind the shortcut as a global toggle to hide show</td>
</tr>
<tr>
<td>Could save</td>
</tr>
<tr>
<td>window position/size</td>
</tr>
<tr>
<td>hidden status</td>
</tr>
<tr>
<td>always on top</td>
</tr>
<tr>
<td>pin to visible workspace</td>
</tr>
<tr>
<td>Layout needs to save/load more settings</td>
</tr>
<tr>
<td>Per layout?</td>
</tr>
<tr>
<td>Group mode status (all, group, off)</td>
</tr>
<tr>
<td>Split to this group</td>
</tr>
<tr>
<td>Autoclean groups</td>
</tr>
<tr>
<td>Per window</td>
</tr>
<tr>
<td>always on top</td>
</tr>
<tr>
<td>pin to visible workspace</td>
</tr>
<tr>
<td>Per tab</td>
</tr>
<tr>
<td>Per terminal</td>
</tr>
<tr>
<td>Store the custom command and working directory when we load a layout, so making small changes and saving doesn't lose everything.</td>
</tr>
<tr>
<td>It could be possible to detect the current command and working directory with psutil, but could be tricky. (i.e. do we ignore bash?)</td>
</tr>
</tbody>
</table>
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 tab text (#1054300-patch), titlebar text, 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.
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

Keybindings
Add a list of the default keybindings to the Preferences -> Keybindings
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

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.

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.

Since then I have resolved to port fixes and features between the two versions. As I do this I explore and find outstanding issues with the port, and it is slowly becoming more usable.

Eventually the GTK2 version of Terminator will go into a deprecated/maintenance mode. Unfortunately due to needing a relatively new version of libvte, that switch will not be in the immediate future. I’m running trusty (14.04 LTS) and even there I had to build libvte 0.38 from source. This makes the GTK3 out of reach for the “Joe Bloggs” of the world. I could try and maintain my own PPA of the component, but that doesn’t help Fedora/OpenSUSE/Arch etc. users.
Even getting “Joe Bloggs” to add a PPA can be a struggle.

And for a real nightmare, I tried to compile the 0.40 version and the thing lit up with a smorgasbord of items where my installed packages were not new enough.

If you are feeling brave and adventurous, there are some instructions in this blog post that will help you get the GTK3 version running. Assistance knocking off the remaining rough edges will be very much appreciated.

For the record, as of October 2015, with the gtk3 branch at revision 1612, these are the outstanding items:

<table>
<thead>
<tr>
<th>Outstanding GTK3 port tasks/items/reviews etc.</th>
</tr>
</thead>
</table>
| Outstanding trunk revisions: 1634 & 1637, 1647 (assuming all is good), 1663

Need to go through all the Gtk.STOCK_+ items and remove. Deprecated in 3.10. Very low priority as won't be problem till 4.0.

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(8)/FIXME(6) FOR GTK3 items to be dealt with

<table>
<thead>
<tr>
<th>Outstanding GTK3 port tasks/items/reviews etc. for future release</th>
</tr>
</thead>
</table>
| vte 0.40+
| Reimplement/restore the word_chars stuff.

Once 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 even 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.

**Terminator API Docs**

Strictly speaking this isn’t an API as such, because it is just using sphinx-apidoc over the Terminator code base. It’s perhaps helpful to have this as a document that can be browsed.

**Terminator API docs**

As it stands, this is rather incomplete, or too terse with no examples given. If you look at the terminatorlib.configobj package, you will see fairly extensive documentation, along with walk-throughs, etc. This particular package was written elsewhere, and brought into Terminator to provide configuration handling.

There are also some aspects of the way this document builds that I’m not too happy about. The seemingly unnecessary terminatorlib root-node in the side bar; the lack of class/method links in the sidebar; all .py files on the same page (this can be changed, but then even less is displayed in the sidebar.) If you can help, join the A-Team... Or better yet, send me some changes that fix this.

**Other Docs for Developers**

Here is a list of some useful sets of documentation:
<table>
<thead>
<tr>
<th>General</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td><a href="https://docs.python.org/release/2.7/index.html">https://docs.python.org/release/2.7/index.html</a></td>
</tr>
<tr>
<td>GNOME Dev. Center</td>
<td><a href="https://developer.gnome.org/">https://developer.gnome.org/</a></td>
</tr>
<tr>
<td>Launchpad Help</td>
<td><a href="https://help.launchpad.net/">https://help.launchpad.net/</a></td>
</tr>
<tr>
<td>GTK 2</td>
<td></td>
</tr>
<tr>
<td>VTE for GTK 2</td>
<td><a href="https://developer.gnome.org/vte/0.28/">https://developer.gnome.org/vte/0.28/</a></td>
</tr>
<tr>
<td>GTK 3</td>
<td></td>
</tr>
<tr>
<td>GObject Introspection</td>
<td><a href="https://wiki.gnome.org/Projects/GObjectIntrospection">https://wiki.gnome.org/Projects/GObjectIntrospection</a></td>
</tr>
<tr>
<td>PyGObject Introspection</td>
<td><a href="https://wiki.gnome.org/Projects/PyGObject">https://wiki.gnome.org/Projects/PyGObject</a></td>
</tr>
<tr>
<td>Many PIGO autodocs</td>
<td><a href="http://lazka.github.io/pgi-docs/">http://lazka.github.io/pgi-docs/</a></td>
</tr>
<tr>
<td>VTE for GTK 3</td>
<td><a href="https://developer.gnome.org/vte/0.38/">https://developer.gnome.org/vte/0.38/</a></td>
</tr>
</tbody>
</table>
[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.