FAQ

Jump to bottom

Nicholas Marriott edited this page on Jan 22, 2024 · 37 revisions

```
PLEASE NOTE: most display problems are due to incorrect TERM! Before
reporting problems make SURE that TERM settings are correct inside and
outside tmux.

Inside tmux TERM must be "screen", "tmux" or similar (such as "tmux-256color").
Don't bother reporting problems where it isn't!

Outside, it should match your terminal: particularly, use "rxvt" for rxvt
and derivatives.
```

What is `TERM` and what does it do?

The environment variable `TERM` tells applications the name of a terminal description to read from the *terminfo(5)* database. Each description consists of a number of named capabilities which tell applications what to send to control the terminal. For example, the `cup` capability contains the escape sequence used to move the cursor up.

It is important that `TERM` points to the correct description for the terminal an application is running in - if it doesn't, applications may misbehave.

The *infocmp(1)* command shows the contents of a terminal description and the *tic(1)* command builds and installs a description from a file (the `-x` flag is normally required with both).

I found a bug in tmux! What do I do?

Check the latest version of tmux from Git to see if the problem is still present.

Please send bug reports by email to *nicholas.marriott@gmail.com* or *tmux-users@googlegroups.com* or by opening a GitHub issue. Please see the CONTRIBUTING file for information on what to include.

Why doesn't tmux do $x?

Please send feature requests by email to *tmux-users@googlegroups.com* or open a GitHub issue.

How often is tmux released? What is the version number scheme?

tmux releases are now made approximately every six months, around the same time as OpenBSD releases (tmux is part of OpenBSD) but not necessarily on the same day.

tmux version numbers (as reported by `tmux -V` and `display -p '#{version}'`) match one of the following:

- Main releases have a version with one digit after the period, such as 2.9 or 3.0. The number increases with each release, so 2.8, 2.9, 3.0, 3.1 and so on.
- Patch releases have a letter following the last digit, such as 2.9a or 3.0a. These contain a small number of changes to fix any bugs found shortly after release.
- The development source tree (master in Git) has a version prefixed by "next-", for example next-3.1 is the code that will eventually become the 3.1 release.
- Release candidates have an "-rc" suffix, optionally with a number. So the first 3.1 release candidate is 3.1-rc, the second 3.1-rc2 and so on.
- tmux in OpenBSD does not have a version number. Until OpenBSD 6.6, it did not support the `-V` flag; in 6.6 and later it does and reports the OpenBSD version number prefixed by "openbsd-", for example openbsd-6.6.

Aside from the patch releases and release candidates mentioned above, tmux version numbers have no special significance. tmux 3.0 is the release after tmux 2.9, nothing more.

Why do you use the screen terminal description inside tmux?

It is already widely available. `tmux` and `tmux-256color` entries are provided by modern *ncurses(3)* and can be used instead by setting the `default-terminal` option.

tmux exited with `server exited unexpectedly` or `lost server`. What does this mean?

This message usually means the tmux server has crashed.

If the problem can be reproduced, please open a issue reporting this - there is information in the CONTRIBUTING file on what to include and how to get logs from tmux.

It can also be useful to enable core dumps to see why tmux crashed. On most platforms this can be done by running `ulimit -c unlimited` before starting tmux; if it crashes again it may generate a core file in the directory where it is started. This may be called `tmux.core` or `core.*` or just `core`. It can be loaded into `gdb` like this:

```
$ gdb `which tmux` /path/to/core/file
...
(gdb) bt full
```

If this works, include the output in the issue.

tmux says `no sessions` when I try to attach but I definitely had sessions!

Check if tmux is still running with `pgrep` or `ps`. If not, then the server probably crashed or was killed and the sessions are gone.

If tmux is still running, most likely something deleted the socket in `/tmp`. The tmux server can be asked to recreate its socket by sending it the `USR1` signal, for example: `pkill -USR1 tmux`

I don't see any colour in my terminal! Help!

On a few platforms, common terminal descriptions such as `xterm` do not include colour. screen ignores this, tmux does not. If the terminal emulator in use supports colour, use a value for `TERM` which correctly lists this, such as `xterm-color`.

tmux freezes my terminal when I attach. I have to `kill -9` the shell it was started from to recover!

Some consoles don't like attempts to set the window title. Tell tmux not to do this by turning off the `set-titles` option (you can do this in `.tmux.conf`:

```
set -g set-titles off
```

If this doesn't fix it, send a bug report.

Why is C-b the prefix key? How do I change it?

The default key is C-b because the prototype of tmux was originally developed inside screen and C-b was chosen not to clash with the screen meta key.

To change it, change the `prefix` option, and - if required - move the binding of the `send-prefix` command from C-b (C-b C-b sends C-b by default) to the new key. For example:

```
set -g prefix C-a
unbind C-b
bind C-a send-prefix
```

How do I use UTF-8?

tmux requires a system that supports UTF-8 (that is, where the C library has a UTF-8 locale) and will not start if support is missing.

tmux will attempt to detect if the terminal it is running in supports UTF-8 by looking at the `LC_ALL`, `LC_CTYPE` and `LANG` environment variables.

If it believes the terminal is not compatible with UTF-8, any UTF-8 characters will be replaced with underscores. The `-u` flag explicitly tells tmux that the terminal supports UTF-8:

```
$ tmux -u new
```

How do I use a 256 colour terminal?

Provided the underlying terminal supports 256 colours, it is usually sufficient to add one of the following to `~/.tmux.conf`:

```
set -g default-terminal "screen-256color"
```

Or:

```
set -g default-terminal "tmux-256color"
```

And make sure that `TERM` outside tmux also shows 256 colours, or use the tmux `-2` flag.

How do I use RGB colour?

tmux must be told that the terminal outside supports RGB colour. This is done by specifying the `RGB` or `Tc` *terminfo(5)* flags. `RGB` is the official flag, `Tc` is a tmux extension.

With tmux 3.2 and later this can be added with the `terminal-features` option:

```
set -as terminal-features ",gnome*:RGB"
```

Or for any tmux version the `terminal-overrides` option:

```
set -as terminal-overrides ",gnome*:Tc"
```

For tmux itself, colours may be specified in hexadecimal, for example `bg=#ff0000`.

Why are tmux pane separators dashed rather than continuous lines?

Some terminals or certain fonts (particularly some Japanese fonts) do not correctly handle UTF-8 line drawing characters.

The `U8` capability forces tmux to use ACS instead of UTF-8 line drawing:

```
set -as terminal-overrides ",*:U8=0"
```

I want to use the mouse to select panes but the terminal to copy! How?

Terminals do not offer fine-grained mouse support - tmux can either turn on the mouse and receive all mouse events (clicks, scrolling, everything) or it can leave the mouse off and receive no events.

So it is not possible to configure tmux such that it handles some mouse behaviours and the terminal others, it is all or nothing (`mouse` option `on` or `off`).

However, when an application turns on the mouse, most terminals provide a way to bypass it. On many Linux terminals this is holding down the `Shift` key; for iTerm2 it is the `option` key.

Note that tmux makes no attempt to keep the terminal scrollback consistent (it is impossible to do this with multiple windows or panes), so it is very likely to be incomplete.

Disabling a mouse behaviour in tmux rather than having the terminal handle it is done by unbinding the appropriate key bindings, for example to stop tmux changing the current window when the status line is clicked on, unbind `MouseDown1Status` in the root table:

```
unbind -Troot MouseDown1Status
```

How do I translate `-fg`, `-bg` and `-attr` options into `-style` options?

Before tmux 1.9, styles (the colours and attributes of various things) were each configured with three options - one for the foreground colour (such as `mode-fg`), one for the background (such as `mode-bg`) and one for the attributes (such as `mode-attr`).

In tmux 1.9 each set of three options were combined into a single option (so `mode-fg`, `mode-bg` and `mode-attr` became `mode-style`) and in tmux 2.9 the old options were removed. So for example:

```
set -g mode-fg yellow
set -g mode-bg red
set -g mode-attr blink,underline
```

Should be changed to:

```
set -g mode-style fg=yellow,bg=red,blink,underline
```

The format of style options is described in the manual.

What is the `escape-time` option? Is zero a good value?

Terminal applications like tmux receive key presses as a stream of bytes with special keys marked by the ASCII ESC character (`\033`). The problem - and the reason for escape-time - is that as well as marking special keys, the same ASCII ESC is also used for the Escape key itself.

If tmux gets a `\033` byte followed a short time later by an x, has the user pressed Escape followed by x, or have they pressed M-x? There is no guaranteed way to know.

The solution to this problem used by tmux and most other terminal applications is to introduce a delay. When tmux receives `\033`, it starts a timer - if the timer expires without any following bytes, then the key is Escape. The downside to this is that there is a delay before an Escape key press is recognised.

If tmux is running on the same computer as the terminal, or over a fast network, then typically the bytes representing a key will all arrive together, so an `escape-time` of zero is likely to be fine. Over a slower network, a larger value would be better.

How do I make modified function and arrow keys (like C-Up, M-PageUp) work inside tmux?

tmux sends modified function keys using *xterm(1)*-style escape sequences. This can be verified using `cat`, for example pressing M-Left:

```
$ cat
^[[1;3D
```

If this is different, then `TERM` outside tmux is probably incorrect and tmux can't recognise the keys coming from the outside terminal.

If it is correct, then some applications inside tmux do not recognise these keys if `TERM` is set to `screen` or `screen-256color`, because these terminal descriptions lack the capabilities. The `tmux` and `tmux-256color` descriptions do have such capabilities, so using those instead may work. In `.tmux.conf`:

```
set -g default-terminal tmux-256color
```

What is the proper way to escape characters with `#(command)`?

When using the `#(command)` construction to include the output from a command in the status line, the command will be parsed twice. First, when it's read by the configuration file or the command-prompt parser, and second when the status line is being drawn and the command is passed to the shell. For example, to echo the string "(test)" to the status line, either single or double quotes could be used:

```
set -g status-right "#(echo \\\\(test\\\\))"
set -g status-right '#(echo \\\(test\\\))'
```

In both cases, the status-right option will be set to the string `#(echo \\(test\\))` and the command executed will be `echo \(test\)`.

tmux uses too much CPU. What do I do?

Automatic window renaming may use a lot of CPU, particularly on slow computers: if this is a problem, turn it off with `setw -g automatic-rename off`. If this doesn't fix it, please report the problem.

What is the best way to display the load average? Why no `#L`?

It isn't possible to get the load average portably in code and it is preferable not to add portability goop. The following works on at least Linux, *BSD and OS X:

```
uptime|awk '{split(substr($0, index($0, "load")), a, ":"); print a[2]}'
```

How do I attach the same session to multiple clients but with a different current window, like `screen -x`?

One or more of the windows can be linked into multiple sessions manually with `link-window`, or a grouped session with all the windows can be created with `new-session -t`.

I don't see italics! Or italics and reverse are the wrong way round!

GNU screen does not support italics and the `screen` terminal description uses the italics escape sequence incorrectly.

As of tmux 2.1, if `default-terminal` is set to `screen` or matches `screen-*`, tmux will behave like screen and italics will be disabled.

To enable italics, make sure you are using the tmux terminal description:

```
set -g default-terminal "tmux"
```

How do I see the default configuration?

Show the default session options by starting a new tmux server with no configuration file:

```
$ tmux -Lfoo -f/dev/null start\; show -g
```

Or the default window options:

```
$ tmux -Lfoo -f/dev/null start\; show -gw
```

How do I copy a selection from tmux to the system's clipboard?

See this page.

Why do I see dots around a session when I attach to it?

Until version 2.9, tmux limits the size of the window to the smallest attached client. If it didn't do this then it would be impossible to see the entire window. The dots mark the size of the window tmux can display.

To avoid this, detach all other clients when attaching:

```
$ tmux attach -d
```

Or from inside tmux by detaching individual clients with C-b D or all using:

```
C-b : attach -d
```

With 2.9 or later, setting the `window-size` option to `largest` will use the largest attached client rather than smallest.

How do I use *ssh-agent(1)* with tmux?

ssh-agent(1) sets an environment variable (`SSH_AUTH_SOCK`) which needs to be present in every shell process.

It is possible to make sure `SSH_AUTH_SOCK` is set before running tmux, then it will be in the global environment and will be set for every pane created in tmux. The `update-environment` option contains `SSH_AUTH_SOCK` by default so it will update `SSH_AUTH_SOCK` in the session environment when a session is attached or a new session is created. However, if `SSH_AUTH_SOCK` is *not* set when a session attached, `update-environment` will cause `SSH_AUTH_SOCK` to be *removed* from the environment and not set for new panes. See here and here.

In practice, it is more reliable to set up *ssh-agent(1)* in a shell profile that is run for every shell regardless of tmux. For a Bourne-style shell like *ksh(1)* or *bash(1)*, something like this in `.profile` , `.kshrc` , `.bash_profile` or `.bashrc` :

```
[ ! -f ~/.ssh.agent ] && ssh-agent -s >~/.ssh.agent
eval `cat ~/.ssh.agent` >/dev/null
if ! kill -0 $SSH_AGENT_PID 2>/dev/null; then
        ssh-agent -s >~/.ssh.agent
        eval `cat ~/.ssh.agent` >/dev/null
fi
```

What is the passthrough escape sequence and how do I use it?

tmux takes care not to send escape sequences to a terminal that it isn't going to understand because it can't predict how it will react.

However, it can be forced to pass an escape sequence through by wrapping it in a special form of the DCS sequence with the content prefixed by `tmux;` . Any `\033` characters in the wrapped sequence must be doubled, for example:

```
\033Ptmux;\033\033]1337;SetProfile=NewProfileName\007\033\\
```

Will pass this iTerm2 special escape sequence `\033]1337;SetProfile=NewProfileName\007` through to the terminal without tmux discarding it.

This feature should be used with care. Note that because tmux isn't aware of any changes made to the terminal state by the passthrough escape sequence, it is possible for it to undo them.

The passthrough escape sequence is no longer necessary for changing the cursor colour or style as tmux now has its own support (see the `Cs` , `Cr` , `Ss` and `Se` capabilities).

As of tmux 3.3, the `allow-passthrough` option must be set to `on` or `all` for the passthrough sequence to work.

How can I make .tmux.conf portable between tmux versions?

For `set-option` , the `-q` flag suppresses warnings about unknown options, for example:

```
set -gq mode-bg red
```

Since tmux 2.3, the running server version is available in the `version` format variable. This can be used with `if-shell` , `if-shell -F` (since tmux 2.0) or `%if` (since tmux 2.4) to check for specific server versions.

The `m` (since tmux 2.6) and `m/r` (since tmux 3.0) modifiers are most useful for this. For example to show a green status line if running a development build, blue if version 3.1 or above and red otherwise:

```
%if #{m/r:^next-,#{version}}
set -g status-style bg=green
%elif #{&&:#{m/r:^[0-9]+\.[0-9]+$,#{version}},#{e|>=|f:#{version},3.1}}
set -g status-style bg=blue
```

```
%else
set -g status-style bg=red
%endif
```

For versions older than tmux 2.3, `if-shell` and `tmux -V` must be used.

Note that on OpenBSD version numbers the tmux version number tracks the OpenBSD version, see this FAQ entry for information on tmux version numbers.

Why don't XMODEM, YMODEM and ZMODEM work inside tmux?

tmux is not a file transfer program and these protocols are more effort to support than their remaining popularity deserves. Detach tmux before attempting to use them.

Clone this wiki locally

```
https://github.com/tmux/tmux.wiki.git
```