Formats

Jump to bottom

Nicholas Marriott edited this page on Jul 5, 2022 · 33 revisions

Working with formats

Formats are a powerful way to get information about a running tmux server and control the output of various commands.

They are used widely, for example:

- Displaying information (`display-message`).
- The format of text on the status lines, the terminal title (`set-titles-string`), and automatic rename (`automatic-rename-format`).
- The output of list commands (`-F` flags to `list-clients` , `list-commands` , `list-sessions` , `list-windows` , `list-panes` , `list-buffers`).
- Targets printed by and arguments passed to new window and session (`-F` and `-c` flags to `new-session` , `break-pane` , `new-window` , `split-window`).
- Displayed text and filters in choose modes (`-F` and `-f` to `choose-client` , `choose-tree` , `choose-buffer` , as well as the `f` key).
- Setting option values (`-F` flag to `set-option`).
- Running commands (`-F` flag to `if-shell` and `run-shell`).
- Menu content (`display-menu`).
- Parse-time conditionals in the configuration file (`%if`).

This document gives a description of their syntax with examples.

Formats are also documented in the manual here, together with a list of all the variables.

Note that some of these features are only available in tmux 3.1 and later (most notably: padding and multiple `s` modifiers).

Basic use

A format is a string containing special directives contained in `#{}` which tmux will expand (note that this is different from `#[]` which is used for embedded styles). Each `#{}` can reference named variables with some information about the server, session, client or similar. Not all variables are always present - an unknown or missing variable is replaced with nothing.

The simplest use is to display some information, for example to get the tmux server PID using `display-message` (the `-p` flag prints the result rather than displaying it in the status line):

```
$ tmux display -p '#{pid}'
98764
```

Or to modify the `list-windows` output:

```
$ tmux lsw -F '#{window_id} #{window_name}'
@0 irssi
@1 mutt
@3 emacs
@4 ksh
```

`display-message` is a useful command for working with formats. The `-a` flag lists the formats it knows about:

```
$ tmux display -a|fgrep width=
client_width=143
pane_width=143
window_width=143
```

`-v` prints verbose information on how the format is evaluated to help spot mistakes:

```
$ tmux display -vp '#{pid}'
# expanding format: #{pid}
# found #{}: pid
# format 'pid' found: 98764
# replaced 'pid' with '98764'
# result is: 98764
98764
```

Because `#` is special in formats, it needs to be doubled (`##`) to include a `#` :

```
$ tmux display -p '##{pid}'
#{pid}
```

Many formats have a single-character alias, such as `#S` for `#{session_name}` , but these can't be used with modifiers and their use is not encouraged.

Options and environment variables

As well as format variables, a `#{}` in a format can contain the name of a tmux option, such as a user option:

```
$ tmux set @foo 'hello'
$ tmux display -p 'hello #{@foo}'
hello hello
```

Or the name of an environment variable in the global environment:

```
$ tmux showenv -g USER
USER=nicholas
$ tmux display -p '#{USER}'
nicholas
```

Most of the examples below use user options (such as `@v`) but any format variable or option or environment variable may be used instead.

Simple modifiers

The result of expanding a variable can be changed with modifiers.

There are a few different forms of modifier, but most consist of an operator with zero or more arguments separated by a punctuation character (`|` or `/` are most usual), followed by a colon and one or more additional arguments which are the variables or formats the modifier is applied to.

The simplest modifiers take no arguments and a single variable to modify. The `t` modifier displays a time variable as a human readable string:

```
$ tmux lsw -F '#{t:window_activity}'
Fri Nov 29 13:52:35 2019
Fri Nov 29 13:37:53 2019
Fri Nov 29 12:06:46 2019
Thu Nov 28 15:51:20 2019
Thu Nov 28 07:41:05 2019
```

`b` and `d` trim the file and directory name from a path:

```
$ tmux set @p `pwd`
$ tmux display -p '#{d:@p}'
/usr/src/usr.bin
$ tmux display -p '#{b:@p}'
tmux
```

Trimming and padding

Format variables may be trimmed or padded. The `=` modifier trims and the `p` modifier pads. They both take at least one argument, the width - a positive width means to trim on the left or pad on the right and a negative the opposite:

```
$ tmux set @v "foobar"
$ tmux display -p '#{=3:@v}'
foo
$ tmux display -p '#{=-3:@v}'
bar
$ tmux display -p '#{p9:@v}baz'
foobar   baz
$ tmux display -p '#{p-9:@v}baz'
   foobarbaz
```

Multiple modifiers can be applied together by separating them with a `;`, for example:

```
$ tmux set @v "foobar"
$ tmux display -p '#{=3;p-6:@v}'
   foo
```

The `=` trim modifier accepts a second argument which is a string to append or prepend to the result to show it has been trimmed. When giving more than one argument to a modifier, they must be separated by a punctuation character, including one right after the operator. `/` or `|` are most common separators:

```
$ tmux set @v "foobar"
$ tmux display -p '#{=|6|...:@v}'  # nothing trimmed
foobar
$ tmux display -p '#{=|5|...:@v}'  # trimmed on the right
fooba...
$ tmux display -p '#{=|-5|...:@v}' # trimmed on the left
...oobar
```

Comparisons

For comparison purposes, tmux considers the result of a format to be either true or false. The result is true if it is not empty and not a single zero (`0`), otherwise it is false.

There are several comparison operators available. Rather than the single variable given to `t` and `b` , these take two variables to compare, separated by a comma. Both of these may be formats themselves rather than variables.

`==` , `!=` , `<` , `>` , `<=` and `>=` are string comparisons:

```
$ tmux set @v foo
$ tmux display -p '#{==:#{@v}bar,foobar}'
1
$ tmux display -p '#{!=:#{@v}bar,foobar}'
0
$ tmux display -p '#{<:#{@v},bar}'
0
```

`||` is true if either of its arguments are true and `&&` if both are:

```
$ tmux set @v foo
$ tmux display -p '#{||:0,#{@v}}'
1
$ tmux display -p '#{&&:0,#{@v}}'
0
```

A ternary choice operator `?` is also available. This is slightly different from the other comparison modifiers (it was implemented earlier) and has no colon between the operator and the condition to check. The condition is followed by two result formats, the first is chosen if the condition is true and the second if it is false. Either may be empty. For example:

```
$ tmux set @v 0
$ tmux display -p '#{?@v,yes,no}'
no
$ tmux display -p '#{?#{==:#{@v},0},yes,no}'
yes
$ tmux display -p '#{?#{==:#{@v},0},#{@v} is true,#{@v} is false}'
0 is true
```

Inside the results of the choice operator, a comma can be inserted by escaping it as `#,` .

Substitution

Formats support substitution through the `s` modifier. This is similar to *sed(1)* substitution and takes two or three arguments - a regular expression to search for, the string to replace it with and a set of flags. Both the regular expression to search for and the string to replace with may be formats themselves. Patterns in brackets are expanded in the replacement by number (`\1` , `\2` and so on).

Like `t` and `b` and `d` , the variable being replaced cannot be a format but must be a variable. Examples are:

```
$ tmux set @v foobar
$ tmux display -p '#{s|foo|bar|:@v}'
barbar
$ tmux display -p '#{s|(foo)(bar)|\2\1|:@v}'
barfoo
$ tmux set @w foo
$ tmux display -p '#{s|#{@w}|xxx|:@v}'
xxxbar
```

The third argument supports one flag, `i` , which means the regular expression is case insensitive:

```
$ tmux set @v foobar
$ tmux display -p '#{s|FOO|xxx|:@v}'
foobar
$ tmux display -p '#{s|FOO|xxx|i:@v}'
xxxbar
```

Multiple substitutions may be done in series by separating them with `;` , like so:

```
$ tmux set @v foobar
$ tmux display -p '#{s|foo|xxx|;s|bar|yyy|:@v}'
xxxyyy
```

Expressions

In tmux 3.2 and later versions, some mathematical operations are available as format modifiers. These are given using the `e` modifier. The first argument is one of:

Argument	Operation
`+`	Add
`-`	Subtract
`*`	Multiply
`/`	Divide
`m`	Modulus

The second argument can be the flag `f` to use floating point numbers otherwise integers are used. The third argument is the number of decimal places to show in the result - the default is zero for integers and two for floating point numbers.

For example:

```
$ tmux display -p '#{e|+|:1,1}'
2
$ tmux display -p '#{e|/|f|4:10,3}'
3.3333
```

Matching and searching

The matching modifier `m` is similar to the comparison modifiers and compares two formats - the first is a pattern matched against the second. By default this expects an *fnmatch(3)* pattern, but the `r` flag specifies a regular expression. The `i` flag means case insensitive.

```
$ tmux set @v foobar
$ tmux display -p '#{m:*foo*,#{@v}}'
1
```

```
$ tmux display -p '#{m|ri:^FOO,#{@v}}'
1
```

The `C` modifier searches for the given format in the pane content and returns the line number or zero if not found.

```
$ # xyz
$ tmux display -p '#{C:x*z}'
4
$ tmux display -p '#{C|r:x.z}'
2
```

Loops

The `S`, `W` and `P` modifiers loop over every session, every window in the current session and every pane in the current window and expand the given format for each. `W` or `P` may be given a second format which is used for the current window and active pane. For example:

```
$ tmux display -p '#{W:#{window_name} }'
irssi mutt emacs ksh ksh ksh ksh ksh ksh emacs emacs ksh ksh ksh ksh ksh
$ tmux display -p '#{W:#{window_name} ,--- }'
irssi mutt emacs ksh ksh ksh ksh ksh --- emacs emacs ksh ksh ksh ksh ksh
```

Literals and quoting

The `l` modifier results in the literal string given to it, this can be used for a literal as the first argument to the `?` choice modifier, for example:

```
$ tmux display -p '#{?#{l:1},a,b}'
a
```

The `q` modifier quotes any special characters, typically used if the result is being passed as an argument to another command.

```
$ tmux set @v '()'
$ tmux display -p '#{q:@v}'
\(\)
```

Multiple expansion

tmux has two modifiers which expand their result twice: `E` and `T`. `#{E:status-left}` will expand the contents of the `status-left` option. `T` is the same but also expands *strftime(3)* conversion specifiers (like `%H` and `%M`).

```
$ tmux display -p '#{status-left}'
[#{session_name}]
$ tmux display -p '#{T:status-left}'
[0]
```

Using formats together

Often formats are nested and used together in much more complicated ways than the examples here. For example, here is part of the default `status-format[0]`:

```
#{?#{&&:#{||:#{window_activity_flag},#{window_silence_flag}},#{!=:#{window-statu
activity-style},default}}, #{window-status-activity-style},}
```

This expands to the content of the `window-status-activity-style` option if either of `window_activity_flag` or `window_silence_flag` is true and the `window-status-activity-style` option is not `default`.

Choose modes and formats

Formats are used for two purposes in the three choose modes: for the format of each line and for filters.

The format of each line is specified with `-F` to `choose-buffer`, `choose-tree` or `choose-client`. The default formats are themselves available in formats:

```
$ tmux display -p '#{client_mode_format}'
session #{session_name} (#{client_width}x#{client_height}, #{t:client_activity})
$ tmux lsc -F '#{E:client_mode_format}'
session 0 (143x44, Mon Dec  2 12:51:29 2019)
session 0 (81x24, Mon Dec  2 12:51:26 2019)
```

`choose-tree` is the most complicated because it may be used for a line containing a session, a window or a pane. The same format does all three by using the `pane_format`, `window_format` and `session_format` variables. The first is true for all three types of line; the second only for panes and windows; and the third only for sessions.

All three modes support filters using the `-f` flag or by pressing the `f` key. A filter is a format which is expanded for each line, if it is true then the line is included in the list and if false it is not. For example to only show sessions named `mysession`:

```
$ tmux choose-tree -sf '#{==:#{session_name},mysession}'
```

Or only windows with a pane containing the text `foo`:

```
$ tmux choose-tree -wf '#{C:foo}'
```

The `find-window` command works by automatically generating a filter like this.

Summary of modifiers

The table below shows the syntax of the modifiers. The following fields are used:

- `variable` means a variable name only, such as `session_name` or `@foo`. This is not expanded and can't contain `#{}`.
- `format` means a string fully expanded as a format, such as `abc` or `abc#{@foo}`. Variables must be included in `#{}` or they are not expanded: `session_name` is the literal string `session_name`, it must be `#{session_name}` to expand it.
- `single` means a single format or a variable, but not a string, so `#{session_name}` or `session_name` will both be expanded but `abc#{session_name}` will not.
- `string` means a string that is not expanded, such as `abc`.
- `flags` is an optional set of single character flags.

Format	Description
`#{t:variable}`	Time to human readable string
`#{b:variable}`	File name of path
`#{d:variable}`	Directory name of path

Format	Description
`#{=N:variable}`	Trim to width N
`#{=/N/format:variable}`	Trim to width N with a marker if trimmed
`#{pN:variable}`	Pad to width N
`#{==:format,format}`	Compare two formats (also `!=` `<` `>` `<=` `>=`)
`#{?single,format,format}`	Choose from two formats
`#{s/format/format/flags:variable}`	Substitute pattern with a string, flag `i`
`#{m/flags:format,format}`	Match a pattern against a format, flags `r` and `i`
`#{C/flags:format}`	Search for a format, flags `r` and `i`
`#{P:format,format}`	Loop over each pane (also `S`, `W`)
`#{l:string}`	Literal string
`#{E:variable}`	Expand variable content
`#{T:variable}`	Expand variable content with time conversion specifiers
`#{q:variable}`	Quote special characters

Clone this wiki locally

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