
tmuxp Documentation

Release 1.5.5

Tony Narlock

Aug 15, 2020

CONTENTS

1 tmux session manager 1

1.1 Installation 1

1.2 Load a tmux session 1

1.3 User-level configurations 2

1.4 Pre-load hook 2

1.5 Load in detached state 2

1.6 Screenshot 2

1.7 Freeze a tmux session 2

1.8 Pre-load hook 3

1.9 Load in detached state 3

1.10 Screenshot 3

1.11 Freeze a tmux session 3

1.12 Docs / Reading material 3

1.13 Donations 3

1.14 Project details 4

Python Module Index 73

Index 75

TMUX SESSION MANAGER

We need help! tmuxp is a trusted session manager for tmux. If you could lend your time to helping answer issues and QA pull requests, please do! See [issue #290](#)!

New to tmux? [The Tao of tmux](#) is available on Leanpub and [Amazon Kindle](#). Read and browse the book for free [on the web](#).

1.1 Installation

```
$ pip install --user tmuxp
```

1.2 Load a tmux session

Load tmux sessions via json and YAML, [tmuxinator](#) and [teamocil](#) style.

```
session_name: 4-pane-split
windows:
- window_name: dev window
  layout: tiled
  shell_command_before:
    - cd ~/                                # run as a first command in all panes
  panes:
    - shell_command:                        # pane no. 1
      - cd /var/log                        # run multiple commands in this pane
      - ls -al | grep \.log
    - echo second pane                     # pane no. 2
    - echo third pane                      # pane no. 3
    - echo forth pane                      # pane no. 4
```

Save as *mysession.yaml*, and load:

```
$ tmuxp load ./mysession.yaml
```

Projects with *.tmuxp.yaml* or *.tmuxp.json* load via directory:

```
$ tmuxp load path/to/my/project/
```

Load multiple at once (in bg, offer to attach last):

```
$ tmuxp load mysession ./another/project/
```

simple and very elaborate config examples

1.3 User-level configurations

tmuxp checks for configs in user directories:

- `$TMUXP_CONFIGDIR`, if set
- `$XDG_CONFIG_HOME`, usually `$HOME/.config/tmuxp/`
- `$HOME/.tmuxp/`

Load your tmuxp config from anywhere by using the filename, assuming `~/.config/tmuxp/mysession.yaml` (or `.json`):

```
$ tmuxp load mysession
```

See [author's tmuxp configs](#) and the projects' `tmuxp.yaml`.

1.4 Pre-load hook

Run custom startup scripts (such as installing project dependencies before loading tmux. See the `bootstrap_env.py` and `before_script` example

1.5 Load in detached state

You can also load sessions in the background by passing `-d` flag

1.6 Screenshot

1.7 Freeze a tmux session

- `$TMUXP_CONFIGDIR`, if set
- `$XDG_CONFIG_HOME`, usually `$HOME/.config/tmuxp/`
- `$HOME/.tmuxp/`

Load your tmuxp config from anywhere by using the filename, assuming `~/.config/tmuxp/mysession.yaml` (or `.json`):

```
$ tmuxp load mysession
```

See [author's tmuxp configs](#) and the projects' `tmuxp.yaml`.

1.8 Pre-load hook

Run custom startup scripts (such as installing project dependencies before loading tmux. See the [bootstrap_env.py](#) and [before_script](#) example

1.9 Load in detached state

You can also load sessions in the background by passing `-d` flag

1.10 Screenshot

1.11 Freeze a tmux session

Snapshot your tmux layout, pane paths, and window/session names.

```
$ tmuxp freeze session-name
```

See more about [freezing tmux](#) sessions.

1.12 Docs / Reading material

See the [Quickstart](#).

[Documentation](#) homepage (also in)

Want to learn more about tmux itself? [Read The Tao of Tmux online](#).

1.13 Donations

Your donations fund development of new features, testing and support. Your money will go directly to maintenance and development of the project. If you are an individual, feel free to give whatever feels right for the value you get out of the project.

See donation options at <https://git-pull.com/support.html>.

1.14 Project details

tmux support	1.8, 1.9a, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8
python support	2.7, >= 3.5, pypy, pypy3
config support	yaml, json, python dict
Source	https://github.com/tmux-python/tmuxp
Docs	http://tmuxp.git-pull.com
API	http://tmuxp.git-pull.com/en/latest/api.html
Changelog	http://tmuxp.git-pull.com/en/latest/history.html
Issues	https://github.com/tmux-python/tmuxp/issues
Travis	http://travis-ci.org/tmux-python/tmuxp
Test Coverage	https://codecov.io/gh/tmux-python/tmuxp
pypi	https://pypi.python.org/pypi/tmuxp
Open Hub	https://www.openhub.net/p/tmuxp
License	MIT.
git repo	<pre>\$ git clone https://github.com/tmux- ↪python/tmuxp.git</pre>
install stable	<pre>\$ pip install --user tmuxp</pre>
install dev	<pre>\$ git clone https://github.com/tmux- ↪python/tmuxp.git tmuxp \$ cd ./tmuxp \$ virtualenv .venv \$ source .venv/bin/activate \$ pip install -e . See the developing and testing page in the docs for more.</pre>
tests	<pre>\$ make test</pre>

Explore:

1.14.1 About

tmuxp helps you manage tmux workspaces.

Built on a object relational mapper for tmux. tmux users can reload common workspaces from YAML, JSON and `dict` configurations like [tmuxinator](#) and [teamocil](#).

tmuxp is used by developers for tmux automation at great companies like [Bugsnag](#), [Pragmatic Coders](#) and many others.

To jump right in, see [Quickstart](#) and [Example Configurations](#).

Interested in some kung-fu or joining the effort? [API Reference](#) and [Developing and Testing](#).

MIT-licensed. Code on [github](#).

Differences from tmuxinator / teamocil

Note: If you use teamocil / tmuxinator and can clarify or add differences, please free to [edit this page](#) on github.

Similarities

Load sessions Loads tmux sessions from config

YAML Supports YAML format

Inlining / shorthand configuration All three support short-hand and simplified markup for panes that have one command.

Maturity and stability As of 2016, all three are considered stable, well tested and adopted.

Missing

Version support tmuxp only supports `tmux >= 1.8`. Teamocil and tmuxinator may have support for earlier versions.

Differences

Programming Language python. teamocil and tmuxinator uses ruby.

Workspace building process teamocil and tmuxinator process configs directly shell commands. tmuxp processes configuration via ORM layer.

Additional Features

CLI tmuxp's CLI can attach and kill sessions with tab-completion support. See *Command Line Interface*.

Import config import configs from Teamocil / Tmuxinator¹. See *Import*.

Session freezing Supports session freezing into YAML and JSON format¹. See *Freeze sessions*.

JSON config JSON config support. See *Example Configurations*.

ORM-based API via *libtmux* - Utilizes `tmux >= 1.8`'s unique ID's for panes, windows and sessions to create an object relational view of the `tmux Server`, its `Session`, `Window`, and `Pane`. See *libtmux's internals*.

Conversion `$ tmuxp convert <filename>` can convert files to and from JSON and YAML.

¹ While freezing and importing sessions is a great way to save time, tweaking will probably be required - There is no substitute to a config made with love.

Minor tweaks

- Unit tests against live tmux version to test statefulness of tmux sessions, windows and panes. See [Travis CI](#).
- Load + switch to new session from inside tmux.
- Resume session if config loaded.
- Pre-commands virtualenv / rvm / any other commands.
- Load config from anywhere `$ tmuxp load /full/file/path.json`.
- Load config `.tmuxp.yaml` or `.tmuxp.json` from current working directory with `$ tmuxp load ..`
- `$ tmuxp -2, $ tmuxp -8` for forcing term colors a la *tmux(1)*.
- `$ tmuxp -L<socket-name>, $ tmuxp -S<socket-path>` for sockets and `$ tmuxp -f<config-file>` for config file.

1.14.2 Quickstart

Installation

Assure you have at least tmux **>= 1.8** and python **>= 2.6**.

```
$ pip install tmuxp
```

You can upgrade to the latest release with:

```
$ pip install tmuxp -U
```

Then install [Completion](#).

CLI

See also:

[Example Configurations](#), [Command Line Interface](#), [Completion](#).

tmuxp launches workspaces / sessions from JSON and YAML files.

Configuration files can be stored in `$HOME/.tmuxp` or in project directories as `.tmuxp.py`, `.tmuxp.json` or `.tmuxp.yaml`.

Every configuration is required to have:

1. `session_name`
2. list of windows
3. list of panes for every window in windows

Create a file, `~/tmuxp/example.yaml`:

```
session_name: 2-pane-vertical
windows:
- window_name: my test window
  panes:
    - echo hello
    - echo hello
```

```
$ tmuxp load example.yaml
```

This creates your tmuxp session.

Load multiple tmux sessions at once:

```
$ tmuxp load example.yaml anothersession.yaml
```

tmuxp will offer to switch-client for you if you're already in a session.

You can also have a custom tmuxp config directory by setting the `TMUX_CONFIGDIR` in your environment variables.

```
$ TMUXP_CONFIGDIR=$HOME/.tmuxpmoo tmuxp load cpython
```

Or in your `~/.bashrc` / `~/.zshrc` you can set:

```
export TMUXP_CONFIGDIR=$HOME/.yourconfigdir/tmuxp
```

You can also Import configs `teamocil` and `tmuxinator`.

Pythonics

See also:

`libtmux` python API documentation and *Developing and Testing*, About.

ORM - Object Relational Mapper

AL - Abstraction Layer

python abstraction layer

tmuxp python api	<i>tmux(1)</i> equivalent
<code>libtmux.Server.new_session()</code>	<code>\$ tmux new-session</code>
<code>libtmux.Server.list_sessions()</code>	<code>\$ tmux list-sessions</code>
<code>libtmux.Session.list_windows()</code>	<code>\$ tmux list-windows</code>
<code>libtmux.Session.new_window()</code>	<code>\$ tmux new-window</code>
<code>libtmux.Window.list_panes()</code>	<code>\$ tmux list-panes</code>
<code>libtmux.Window.split_window()</code>	<code>\$ tmux split-window</code>
<code>libtmux.Pane.send_keys()</code>	<code>\$ tmux send-keys</code>

1.14.3 Example Configurations

Short hand / inline style

tmuxp has a short-hand syntax to for those who wish to keep their configs punctual.

```
short hand
```

```
+-----+
| 'did you know' |
| 'you can inline' |
+-----+
| 'single commands' |
|                   |
+-----+
| 'for panes'      |
|                   |
+-----+
```

YAML

```
session_name: shorthands
windows:
- window_name: long form
  panes:
  - shell_command:
    - echo 'did you know'
    - echo 'you can inline'
  - shell_command: echo 'single commands'
  - echo 'for panes'
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "echo 'did you know'",
            "echo 'you can inline'"
          ]
        },
        {
          "shell_command": "echo 'single commands'"
        },
        "echo 'for panes'"
      ],
      "window_name": "long form"
    }
  ],
  "session_name": "shorthands"
}
```

Blank panes

No need to repeat `pwd` or a dummy command. A `null`, `'blank'`, `'pane'` are valid.

Note `' '` counts as an empty carriage return.

YAML

```
session_name: Blank pane test
windows:
  # Emptiness will simply open a blank pane, if no shell_command_before.
  # All these are equivalent
  - window_name: Blank pane test
    panes:
      -
      - pane
      - blank
  - window_name: More blank panes
    panes:
      - null
      - shell_command:
      - shell_command:
      -
  # an empty string will be treated as a carriage return
  - window_name: Empty string (return)
    panes:
      - ''
      - shell_command: ''
      - shell_command:
      - ''
  # a pane can have other options but still be blank
  - window_name: Blank with options
    panes:
      - focus: true
      - start_directory: /tmp
```

JSON

```
{
  "windows": [
    {
      "panes": [
        null,
        "pane",
        "blank"
      ],
      "window_name": "Blank pane test"
    },
    {
      "panes": [
        null,
        {
          "shell_command": null
        },
      ],
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        {
            "shell_command": [
                null
            ]
        },
        {
            "window_name": "More blank panes"
        },
        {
            "panes": [
                "",
                {
                    "shell_command": ""
                },
                {
                    "shell_command": [
                        ""
                    ]
                }
            ],
            "window_name": "Empty string (return)"
        },
        {
            "panes": [
                {
                    "focus": true
                },
                {
                    "start_directory": "/tmp"
                }
            ],
            "window_name": "Blank with options"
        }
    ],
    "session_name": "Blank pane test"
}

```

2 panes

2 pane

```

+-----+
| $ pwd |
|       |
|       |
+-----+
| $ pwd |
|       |
|       |
+-----+

```

YAML

```
session_name: 2-pane-vertical
windows:
- window_name: my test window
  panes:
  - echo hello
  - echo hello
```

JSON

```
{
  "windows": [
    {
      "panes": [
        "echo hello",
        "echo hello"
      ],
      "window_name": "my test window"
    }
  ],
  "session_name": "2-pane-vertical"
}
```

3 panes

3 panes

```
+-----+
| $ pwd |
|       |
|       |
+-----+
| $ pwd | $ pwd |
|       |       |
|       |       |
+-----+
```

YAML

```
session_name: 3-panes
windows:
- window_name: dev window
  layout: main-vertical
  shell_command_before:
  - cd ~/
  panes:
  - shell_command:
    - cd /var/log
```

(continues on next page)

(continued from previous page)

```
- ls -al | grep \.log
- echo hello
- echo hello
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "cd /var/log",
            "ls -al | grep \\.log"
          ]
        },
        {
          "shell_command": [
            "echo hello",
            "echo hello"
          ]
        }
      ],
      "shell_command_before": [
        "cd ~/"
      ],
      "layout": "main-vertical",
      "window_name": "dev window"
    }
  ],
  "session_name": "3-panes"
}
```

4 panes

4 panes

```
+-----+-----+
| $ pwd | $ pwd |
|       |       |
|       |       |
+-----+-----+
| $ pwd | $ pwd |
|       |       |
|       |       |
+-----+-----+
```


YAML

```
session_name: 4-pane-split
windows:
- window_name: dev window
  layout: tiled
  shell_command_before:
    - cd ~/
  panes:
    - shell_command:
        - cd /var/log
        - ls -al | grep \.log
    - echo hello
    - echo hello
    - echo hello
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "cd /var/log",
            "ls -al | grep \\.log"
          ]
        },
        "echo hello",
        "echo hello",
        "echo hello"
      ],
      "shell_command_before": [
        "cd ~/"
      ],
      "layout": "tiled",
      "window_name": "dev window"
    }
  ],
  "session_name": "4-pane-split"
}
```

Start Directory

Equivalent to `tmux new-window -c <start-directory>`.

YAML

```
session_name: start directory
start_directory: /var/
windows:
- window_name: should be /var/
  panes:
  - shell_command:
    - echo "\033c
    - it trickles down from session-level"
    - echo hello
- window_name: should be /var/log
  start_directory: log
  panes:
  - shell_command:
    - echo '\033c
    - window start_directory concatenates to session start_directory
    - if it is not absolute'
    - echo hello
- window_name: should be ~
  start_directory: '~'
  panes:
  - shell_command:
    - 'echo "\\033c ~ has precedence. note: remember to quote ~ in YAML'
    - echo hello
- window_name: should be /bin
  start_directory: /bin
  panes:
  - echo '\033c absolute paths also have precedence.'
  - echo hello
- window_name: should be config's dir
  start_directory: ./
  panes:
  - shell_command:
    - echo '\033c
    - ./ is relative to config file location
    - ../ will be parent of config file
    - ./test will be \"test\" dir inside dir of config file'
  - shell_command:
    - echo '\033c
    - This way you can load up workspaces from projects and maintain
    - relative paths.'
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "echo '\\033c",
            "it trickles down from session-level\""
          ]
        },
        "echo hello"
      ],
      "window_name": "should be /var/"
    },
    {
      "panes": [
        {
          "shell_command": [
            "echo '\\033c",
            "window start_directory concatenates to session start_directory",
            "if it is not absolute'"
          ]
        },
        "echo hello"
      ],
      "start_directory": "log",
      "window_name": "should be /var/log"
    },
    {
      "panes": [
        {
          "shell_command": [
            "echo '\\033c ~ has precedence. note: remember to quote ~ in YAML"
          ]
        },
        "echo hello"
      ],
      "start_directory": "~",
      "window_name": "should be ~"
    },
    {
      "panes": [
        "echo '\\033c absolute paths also have precedence.'",
        "echo hello"
      ],
      "start_directory": "/bin",
      "window_name": "should be /bin"
    },
    {
      "panes": [
        {
          "shell_command": [
            "echo '\\033c",
            "./ is relative to config file location",
            "../ will be parent of config file",

```

(continues on next page)

(continued from previous page)

```
        "._/test will be \\\\"test\\\\" dir inside dir of config file'"
    ]
  },
  {
    "shell_command": [
      "echo '\\033c",
      "This way you can load up workspaces from projects and maintain",
      "relative paths.'"
    ]
  }
],
"start_directory": "./",
>window_name": "should be config's dir"
}
],
"session_name": "start directory",
"start_directory": "/var/"
}
```

Environment variable replacing

tmuxp will replace environment variables wrapped in curly brackets for values of these settings:

- start_directory
- before_script
- session_name
- window_name
- shell_command_before
- global_options
- options in session scope and window scope

tmuxp replaces these variables before-hand with variables in the terminal tmuxp invokes in.

In this case of this example, assuming the username “user”:

```
$ MY_ENV_VAR=foo tmuxp load examples/env-variables.yaml
```

and your session name will be session - user (foo).

Shell variables in shell_command do not support this type of concatenation. shell_command and shell_command_before both support normal shell variables, since they are sent into panes automatically via send-key in tmux(1). See ls \$PWD in example.

If you have a special case and would like to see behavior changed, please make a ticket on the [issue tracker](#).

YAML

```

start_directory: "${PWD}/test"
shell_command_before: "echo ${PWD}"
before_script: "${MY_ENV_VAR}/test3.sh"
session_name: session - ${USER} (${MY_ENV_VAR})
windows:
- window_name: editor
  panes:
  - shell_command:
    - tail -F /var/log/syslog
    start_directory: /var/log
- window_name: logging for ${USER}
  options:
    automatic-rename: true
  panes:
  - shell_command:
    - htop
    - ls $PWD

```

JSON

```

{
  "before_script": "${MY_ENV_VAR}/test3.sh",
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "tail -F /var/log/syslog"
          ]
        }
      ],
      "start_directory": "/var/log",
      "window_name": "editor"
    },
    {
      "panes": [
        {
          "shell_command": [
            "htop",
            "ls $PWD"
          ]
        }
      ],
      "window_name": "logging for ${USER}",
      "options": {
        "automatic-rename": true
      }
    }
  ],
  "shell_command_before": "echo ${PWD}",
  "start_directory": "${PWD}/test",
  "session_name": "session - ${USER} (${MY_ENV_VAR})"
}

```

Environment variables

tmuxp will set session environment variables.

YAML

```
session_name: Environment variables test
environment:
  EDITOR: /usr/bin/vim
  HOME: /tmp/hm
windows:
  # Emptiness will simply open a blank pane, if no shell_command_before.
  # All these are equivalent
  - window_name: Blank pane test
    panes:
      -
```

JSON

```
{
  "environment": {
    "EDITOR": "/usr/bin/vim",
    "HOME": "/tmp/hm",
  },
  "windows": [
    {
      "panes": [
        null,
      ],
      "window_name": "Blank pane test"
    },
  ],
  "session_name": "Environment variables test"
}
```

Focusing

tmuxp allows `focus: true` for assuring windows and panes are attached / selected upon loading.

YAML

```
session_name: focus
windows:
  - window_name: attached window
    focus: true
    panes:
      - shell_command:
          - echo hello
          - echo 'this pane should be selected on load'
        focus: true
```

(continues on next page)

(continued from previous page)

```

- shell_command:
  - cd /var/log
  - echo hello
- window_name: second window
shell_command_before: cd /var/log
panes:
- pane
- shell_command:
  - echo 'this pane should be focused, when window switched to first time'
focus: true
- pane

```

JSON

```

{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "echo hello",
            "echo 'this pane should be selected on load'"
          ],
          "focus": true
        },
        {
          "shell_command": [
            "cd /var/log",
            "echo hello"
          ]
        }
      ],
      "window_name": "attached window on load",
      "focus": true
    },
    {
      "panes": [
        "pane",
        {
          "shell_command": [
            "echo 'this pane should be focused, when window switched to first time'"
          ],
          "focus": true
        },
        "pane"
      ],
      "shell_command_before": "cd /var/www",
      "window_name": "second window"
    }
  ],
  "session_name": "focus window and pane when loading sessions"
}

```

Terminal History

tmuxp allows `suppress_history: false` to override the default command / suppression when building the workspace. This will add the `shell_command` to the bash history in the pane.

YAML

```
session_name: suppress
suppress_history: false
windows:
- window_name: appended
  focus: true
  suppress_history: false
  panes:
  - echo "window in the history!"

- window_name: suppressed
  suppress_history: true
  panes:
  - echo "window not in the history!"

- window_name: default
  panes:
  - echo "session in the history!"

- window_name: mixed
  suppress_history: false
  panes:
  - shell_command:
    - echo "command in the history!"
    suppress_history: false
  - shell_command:
    - echo "command not in the history!"
    suppress_history: true
  - shell_command:
    - echo "window in the history!"
```

JSON

```
{
  "windows": [
    {
      "panes": [
        "echo 'window in the history!'"
      ],
      "focus": true,
      "suppress_history": false,
      "window_name": "appended"
    },
    {
      "panes": [
        "echo 'window not in the history!'"
      ],
      "suppress_history": true,
```

(continues on next page)

(continued from previous page)

```

    "window_name": "suppressed"
  },
  {
    "panes": [
      "echo 'session in the history!'"
    ],
    "window_name": "default"
  },
  {
    "panes": [
      {
        "shell_command": "echo 'command in the history!'",
        "suppress_history": false
      },
      {
        "shell_command": "echo 'command not in the history!'",
        "suppress_history": true
      },
      {
        "shell_command": "echo 'window not in the history!'"
      }
    ],
    "suppress_history": true,
    "window_name": "mixed"
  }
],
"suppress_history": false,
"session_name": "suppress"
}

```

Window Index

You can specify a window's index using the `window_index` property. Windows without `window_index` will use the lowest available window index.

YAML

```

session_name: Window index example
windows:
- window_name: zero
  panes:
    - echo "this window's index will be zero"
- window_name: five
  panes:
    - echo "this window's index will be five"
  window_index: 5
- window_name: one
  panes:
    - echo "this window's index will be one"

```

JSON

```
{
  "windows": [
    {
      "panes": [
        "echo \"this window's index will be zero\""
      ],
      "window_name": "zero"
    },
    {
      "panes": [
        "echo \"this window's index will be five\""
      ],
      "window_index": 5,
      "window_name": "five"
    },
    {
      "panes": [
        "echo \"this window's index will be one\""
      ],
      "window_name": "one"
    }
  ],
  "session_name": "Window index example"
}
```

Set tmux options

Works with global (server-wide) options, session options and window options.

Including automatic-rename, default-shell, default-command, etc.

YAML

```
session_name: test window options
start_directory: '~'
global_options:
  default-shell: /bin/sh
  default-command: /bin/sh
options:
  main-pane-height: ${MAIN_PANE_HEIGHT} # works with env variables
windows:
- layout: main-horizontal
  options:
    automatic-rename: on
  panes:
  - shell_command:
    - man echo
    start_directory: '~'
  - shell_command:
    - echo "hey"
  - shell_command:
    - echo "moo"
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "man echo"
          ],
          "start_directory": "~"
        },
        {
          "shell_command": [
            "echo \"hey\""
          ]
        },
        {
          "shell_command": [
            "echo \"moo\""
          ]
        }
      ],
      "layout": "main-horizontal",
      "options": {
        "automatic-rename": true
      }
    }
  ],
  "session_name": "test window options",
  "start_directory": "~",
  "global_options": {
    "default-shell": "/bin/sh",
    "default-command": "/bin/sh"
  },
  "options": {
    "main-pane-height": "${MAIN_PANE_HEIGHT}"
  }
}
```

Set window options after pane creation

Apply window options after panes have been created. Useful for `synchronize-panes` option after executing individual commands in each pane during creation.

YAML

```
session_name: 2-pane-synchronized
windows:
- window_name: Two synchronized panes
  panes:
  - ssh server1
  - ssh server2
  options_after:
    synchronize-panes: on
```

JSON

```
{
  "session_name": "2-pane-synchronized",
  "windows": [
    {
      "window_name": "Two synchronized panes",
      "panes": [
        "ssh server1",
        "ssh server2"
      ],
      "options_after": {
        "synchronize-panes": true
      }
    }
  ]
}
```

Main pane height

YAML

```
session_name: main-pane-height
start_directory: '~'
windows:
- layout: main-horizontal
  options:
    main-pane-height: 30
  panes:
  - shell_command:
    - top
    start_directory: '~'
  - shell_command:
    - echo "hey"
  - shell_command:
    - echo "moo"
  window_name: my window name
```

JSON

```
{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "top"
          ],
          "start_directory": "~"
        },
        {
          "shell_command": [
            "echo \"hey\""
          ]
        },
        {
          "shell_command": [
            "echo \"moo\""
          ]
        }
      ],
      "layout": "main-horizontal",
      "options": {
        "main-pane-height": 30
      },
      "window_name": "editor"
    }
  ],
  "session_name": "main pane height",
  "start_directory": "~"
}
```

Super-advanced dev environment

See also:

tmuxp developer config in the *Developing and Testing* section.

YAML

```
session_name: tmuxp
start_directory: ./ # load session relative to config location (project root).
before_script: ./tmuxp-before-script.sh
shell_command_before:
- '[ -f .venv/bin/activate ] && source .venv/bin/activate && reset'
windows:
- window_name: tmuxp
  focus: True
  layout: main-horizontal
  options:
    main-pane-height: 35
  panes:
```

(continues on next page)

(continued from previous page)

```
- focus: true
- pane
- make watch_test
- window_name: docs
  layout: main-horizontal
  options:
    main-pane-height: 35
  start_directory: doc/
  panes:
    - focus: true
    - pane
    - make serve
    - make SPHINXBUILD='poetry run sphinx-build' watch
```

JSON

```
{
  "session_name": "tmuxp",
  "start_directory": "./",
  "before_script": "./tmuxp-before-script.sh",
  "shell_command_before": [
    "[ -f .venv/bin/activate ] && source .venv/bin/activate && reset"
  ],
  "windows": [
    {
      "window_name": "tmuxp",
      "focus": true,
      "layout": "main-horizontal",
      "options": {
        "main-pane-height": 35
      },
      "panes": [
        {
          "focus": true
        },
        "pane",
        "make watch_test"
      ]
    },
    {
      "window_name": "docs",
      "layout": "main-horizontal",
      "options": {
        "main-pane-height": 35
      },
      "start_directory": "doc/",
      "panes": [
        {
          "focus": true
        },
        "pane",
        "make serve",
        "make SPHINXBUILD='poetry run sphinx-build' watch"
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    }
  ]
}
```

Bootstrap project before launch

You can use `before_script` to run a script before the tmux session starts building. This can be used to start a script to create a virtualenv or download a virtualenv/rbenv/package.json's dependency files before tmuxp even begins building the session.

It works by using the [Exit Status](#) code returned by a script. Your script can be any type, including bash, python, ruby, etc.

A successful script will exit with a status of 0.

Important: the script file must be `chmod executable +x` or 755.

Run a python script (and check for it's return code), the script is *relative to the ``.tmuxp.yaml``'s root* (Windows and panes omitted in this example):

```
session_name: my session
before_script: ./bootstrap.py
# ... the rest of your config
```

```
{
  "session_name": "my session",
  "before_script": "./bootstrap.py"
}
```

Run a shell script + check for return code on an absolute path. (Windows and panes omitted in this example)

```
session_name: another example
before_script: /absolute/path/this.sh # abs path to shell script
# ... the rest of your config
```

```
{
  "session_name": "my session",
  "before_script": "/absolute/path/this.sh"
}
```

Per-project tmux config

You can load your software project in tmux by placing a `.tmuxp.yaml` or `.tmuxp.json` in the project's config and loading it.

tmuxp supports loading configs via absolute filename with `tmuxp load` and via `$ tmuxp load .` if config is in directory.

```
$ tmuxp load ~/workspaces/myproject.yaml
```

See examples of tmuxp in the wild. Have a project config to show off? Edit this page.

- <https://github.com/tony/dockerfiles/blob/master/.tmuxp.yaml>
- <https://github.com/tony/vcspull/blob/master/.tmuxp.yaml>

- <https://github.com/tony/sphinxcontrib-github/blob/master/tmuxp.yaml>

You can use `start_directory: ./` to make the directories relative to the config file / project root.

Bonus: pipenv auto-bootstrapping

New in version 1.3.4: `before_script` CWD's into the root (session)-level `start_directory`.

If you use [pipenv](#) / [poetry](#), you can use a script like this to ensure your packages are installed:

```
# assuming your .tmuxp.yaml is in your project root directory
session_name: my pipenv project
start_directory: ./
before_script: pipenv install --dev --skip-lock # ensure dev deps install
windows:
- window_name: django project
  focus: true
  panes:
  - blank
  - pipenv run ./manage.py runserver
```

You can also source yourself into the virtual environment using `shell_command_before`:

```
# assuming your .tmuxp.yaml is in your project root directory
session_name: my pipenv project
start_directory: ./
before_script: pipenv install --dev --skip-lock # ensure dev deps install
shell_command_before:
- '[ -d `pipenv --venv` ] && source `pipenv --venv`/bin/activate && reset'
windows:
- window_name: django project
  focus: true
  panes:
  - blank
  - ./manage.py runserver
```

Kung fu

Note: tmuxp sessions can be scripted in python. The first way is to use the ORM in the [API Reference](#). The second is to pass a `dict` into `WorkspaceBuilder` with a correct schema. See: `tmuxp.config.validate_schema()`.

Add yours? Submit a pull request to the [github](#) site!

1.14.4 Command Line Interface

Completion

In bash (~/.bashrc):

```
eval "$(_TMUXP_COMPLETE=source tmuxp) "
```

In zsh (~/.zshrc):

```
eval "$(_TMUXP_COMPLETE=source_zsh tmuxp) "
```

Freeze sessions

```
tmuxp freeze <session_name>
```

You can save the state of your tmux session by freezing it.

Tmuxp will offer to save your session state to .json or .yaml.

Load session

You can load your tmuxp file and attach the vim session via a few shorthands:

1. The directory with a .tmuxp.{yaml,yml,json} file in it
2. The name of the project file in your \$HOME/.tmuxp folder
3. The direct path of the tmuxp file you want to load

```
# path to folder with .tmuxp.{yaml,yml,json}
tmuxp load .
tmuxp load ../
tmuxp load path/to/folder/
tmuxp load /path/to/folder/

# name of the config, assume $HOME/.tmuxp/myconfig.yaml
tmuxp load myconfig

# direct path to json/yaml file
tmuxp load ./myfile.yaml
tmuxp load /abs/path/to/myfile.yaml
tmuxp load ~/myfile.yaml
```

Absolute and relative directory paths are supported.

```
$ tmuxp load <filename>
```

Files named .tmuxp.yaml or .tmuxp.json in the current working directory may be loaded with:

```
$ tmuxp load .
```

Multiple sessions can be loaded at once. The first ones will be created without being attached. The last one will be attached if there is no -d flag on the command line.

```
$ tmuxp load <filename1> <filename2> ...
```

Import

From teamocil

```
tmuxp import teamocil /path/to/file.{json,yaml}
```

From tmuxinator

```
tmuxp import tmuxinator /path/to/file.{json,yaml}
```

Convert between YAML and JSON

```
tmuxp convert /path/to/file.{json,yaml}
```

tmuxp automatically will prompt to convert `.yaml` to `.json` and `.json` to `.yaml`.

1.14.5 Developing and Testing

Our tests are inside `tests/`. Tests are implemented using `pytest`.

`make test` will create a tmux server on a separate `socket_name` using `$ tmux -L test_case`.

Install the latest code from git

Using pip

To begin developing, check out the code from github:

```
$ git clone git@github.com:tmux-python/tmuxp.git
$ cd tmuxp
```

Now create a virtualenv, if you don't know how to, you can create a virtualenv with:

```
$ virtualenv .venv
```

Then activate it to your current tty / terminal session with:

```
$ source .venv/bin/activate
```

Good! Now let's run this:

```
$ pip install -e .
```

This has `pip`, a python package manager install the python package in the current directory. `-e` means `--editable`, which means you can adjust the code and the installed software will reflect the changes.

```
$ tmuxp
```

Using poetry

To begin developing, check out the code from github:

```
$ git clone git@github.com:tmux-python/tmuxp.git
$ cd tmuxp
```

You can create a virtualenv, and install all of the locked packages as listed in poetry.lock:

```
$ poetry install
```

If you ever need to update packages during your development session, the following command can be used to update all packages as per poetry settings or individual package (second command):

```
$ poetry update
$ poetry update requests
```

Then activate it to your current tty / terminal session with:

```
$ poetry shell
```

That is it! You are now ready to code!

Test Runner

As you seen above, the `tmuxp` command will now be available to you, since you are in the virtual environment, your *PATH* environment was updated to include a special version of `python` inside your `.venv` folder with its own packages.

```
$ make test
```

You probably didn't see anything but tests scroll by.

If you found a problem or are trying to write a test, you can file an [issue on github](#).

Test runner options

Test only a file:

```
$ py.test tests/test_config.py
```

will test the `tests/test_config.py` tests.

```
$ py.test tests/test_config.py::test_export_json
```

tests `test_export_json` inside of `tests/test_config.py`.

Multiple can be separated by spaces:

```
$ py.test tests/test_{window,pane}.py tests/test_config.py::test_export_json
```

Visual testing

You can watch tmux testsuite build sessions visually by keeping a client open in a separate terminal.

Create two terminals:

- Terminal 1: `$ tmux -L test_case`
- Terminal 2: `$ cd` into the tmuxp project and into the `virtualenv` if you are using one, see details on installing the dev version of tmuxp above. Then:

```
$ py.test tests/test_workspacebuilder.py
```

Terminal 1 should have flickered and built the session before your eyes. tmuxp hides this building from normal users.

Run tests on save

You can re-run tests automatically on file edit.

Note: This requires `entr(1)`.

Install `entr`. Packages are available available on most Linux and BSD variants, including Debian, Ubuntu, FreeBSD, OS X.

To run all tests upon editing any `.py` file:

```
$ make watch_test
```

You can also re-run a specific test file or any other `py.test` usage argument:

```
$ make watch_test test=tests/test_config.py
$ make watch_test test='-x tests/test_config.py tests/test_util.py'
```

Rebuild sphinx docs on save

Rebuild the documentation when an `.rst` file is edited:

```
$ cd doc
$ make watch

# If you're not source'd via `poetry shell`, you can use this:
$ make SPHINXBUILD='poetry run sphinx-build' watch
```

tmuxp developer config

```

.. (up a dir)
/srv/www/tmuxp/
.env/
  _pycache_/
dist/
doc/
examples/
pkg/
tmuxp.egg-info/
tmuxp/
  testsuite/
    _tmuxp/
      _pycache_/
      __init__.py
      helpers.py
      test_cli.py
      test_config.py
      test_pane.py
      test_server.py
      test_session.py
      test_tmuxobject.py
      test_window.py
      test_workspacebuilder.py
    __init__.py
    __main__.py
    cli.py
    config.py
    exc.py
    formats.py
    log.py
    pane.py
NERD_tree 1

.getByid ... ok
test_where (tmuxp.testsuite.test_tmuxobject.TmuxObjectTest)
.where ... ok

-----
Ran 53 tests in 2.053s

OK (skipped=2)
-- done --

harvesting • 3,8,0- 1:tmuxp* 2:docs# 17 days Thu 1:18:25 PM 2013-10-17

```

After you *Install the latest code from git*, when inside the tmuxp checkout:

```
$ tmuxp load .
```

this will load the `.tmuxp.yaml` in the root of the project.

```

session_name: tmuxp
start_directory: ./ # load session relative to config location (project root).
before_script: ./tmuxp-before-script.sh
shell_command_before:
- '[ -f .venv/bin/activate ] && source .venv/bin/activate && reset'
windows:
- window_name: tmuxp
  focus: True
  layout: main-horizontal
  options:
    main-pane-height: 35
  panes:
    - focus: true
    - pane
    - make watch_test
- window_name: docs
  layout: main-horizontal
  options:
    main-pane-height: 35
  start_directory: doc/
  panes:
    - focus: true
    - pane
    - make serve

```

(continues on next page)

(continued from previous page)

```
- make SPHINXBUILD='poetry run sphinx-build' watch
```

Travis CI

tmuxp uses [travis-ci](#) for continuous integration / automatic unit testing.

To view the tmux and python versions tested see the [.travis.yml](#). Builds are done on master and pull requests and can be viewed on the [travis build site](#).

Testing options

RETRY_TIMEOUT_SECONDS can be toggled if certain workspace builder tests are being stubborn.

e.g. RETRY_TIMEOUT_SECONDS=10 `py.test`

```
language: python
sudo: false
dist: bionic # 18.04
python:
  - 2.7
  - 3.7
  - pypy3
if: (type = push AND branch IN (master)) OR (type = pull_request)
env:
  global:
    - RETRY_TIMEOUT_SECONDS=15
  matrix:
    - TMUX_VERSION=master
    - TMUX_VERSION=3.0a
    - TMUX_VERSION=2.9
    - TMUX_VERSION=2.8
    - TMUX_VERSION=2.8
    - TMUX_VERSION=2.7
    - TMUX_VERSION=2.6
    - TMUX_VERSION=2.5
    - TMUX_VERSION=2.4
    - TMUX_VERSION=2.3
    - TMUX_VERSION=2.2
    - TMUX_VERSION=2.1
    - TMUX_VERSION=2.0
    - TMUX_VERSION=1.9a
    - TMUX_VERSION=1.8
matrix:
  finish_fast: true
  allow_failures:
    - env: TMUX_VERSION=master
    - python: pypy3
cache:
  - pip
  - apt
  - directories:
    - ~/source
    - ~/build
before_install:
  - pip install poetry
```

(continues on next page)

(continued from previous page)

```

- poetry --version
- poetry version
install:
- poetry install -v
before_script:
- mkdir -p ~/source
- mkdir -p ~/build/tmux-$TMUX_VERSION
- if [[ ! -d ~/source/tmux-$TMUX_VERSION ]]; then git clone https://github.com/tmux/
tmux.git ~/source/tmux-$TMUX_VERSION; fi
- cd ~/source/tmux-$TMUX_VERSION
- git checkout $TMUX_VERSION
- export PATH=$HOME/build/tmux-$TMUX_VERSION/bin:$PATH
- if [[ $TMUX_VERSION == "master" || ! $(tmux -V | grep $TMUX_VERSION) ]]; then sh_
autogen.sh; fi
- if [[ $TMUX_VERSION == "master" || ! $(tmux -V | grep $TMUX_VERSION) ]]; then ./
configure --prefix=$HOME/build/tmux-$TMUX_VERSION && make && make install; fi
- cd $TRAVIS_BUILD_DIR
- tmux -V
script:
- poetry run py.test --cov
- make flake8
addons:
  apt:
    packages:
      - libevent-dev
      - libncurses-dev
after_success:
- codecov

```

1.14.6 API Reference

See also:

See [libtmux's API](#) and [Quickstart](#) to see how you can control tmux via python API calls.

Internals

`util.run_before_script` (*cwd=None*)
Function to wrap try/except for `subprocess.check_call()`.

CLI

`cli._reattach()`
Reattach session (depending on env being inside tmux already or not)
Parameters `session` (`libtmux.Session`) –

Notes

If `TMUX` environmental variable exists in the environment this script is running, that means we're in a `tmux` client. So `tmux switch-client` will load the session.

If not, `tmux attach-session` loads the client to the target session.

`cli.get_config_dir()`

Return `tmuxp` configuration directory.

`TMUXP_CONFIGDIR` environmental variable has precedence if set. We also evaluate `XDG` default directory from `XDG_CONFIG_HOME` environmental variable if set or its default. Then the old default `~/.tmuxp` is returned for compatibility.

Returns absolute path to `tmuxp` config directory

Return type `str`

`cli.get_teamocil_dir()`

Return `teamocil` configuration directory.

Returns absolute path to `teamocil` config directory

Return type `str`

See also:

`tmuxp.config.import_teamocil()`

`cli.get_tmuxinator_dir()`

Return `tmuxinator` configuration directory.

Checks for `TMUXINATOR_CONFIG` environmental variable.

Returns absolute path to `tmuxinator` config directory

Return type `str`

See also:

`tmuxp.config.import_tmuxinator()`

`cli.load_workspace(socket_name=None, socket_path=None, colors=None, detached=False, answer_yes=False)`

Load a `tmux` “workspace” session via `tmuxp` file.

Parameters

- **config_file** (`str`) – absolute path to config file
- **socket_name** (`str`, *optional*) – `tmux -L <socket-name>`
- **socket_path** (`str`, *optional*) – `tmux -S <socket-path>`
- **colors** (`str`, *optional*) –
 - ‘-2’ Force `tmux` to support 256 colors
- **detached** (`bool`) – Force detached state. default `False`.
- **answer_yes** (`bool`) – Assume yes when given prompt. default `False`.

Notes

tmuxp will check and load a configuration file. The file will use kaptan to load a JSON/YAML into a `dict`. Then `config.expand()` and `config.trickle()` will be used to expand any shorthands, template variables, or file paths relative to where the config/script is executed from.

`config.expand()` accepts the directory of the config file, so the user's configuration can resolve absolute paths relative to where the config file is. In otherwords, if a config file at `/var/moo/hi.yaml` has `./` in its configs, we want to be sure any file path with `./` is relative to `/var/moo`, not the user's PWD.

A `libtmux.Server` object is created. No tmux server is started yet, just the object.

The prepared configuration and the server object is passed into an instance of `WorkspaceBuilder`.

A sanity check against `libtmux.common.which()` is ran. It will raise an exception if tmux isn't found.

If a tmux session under the same name as `session_name` in the tmuxp configuration exists, tmuxp offers to attach the session. Currently, tmuxp does not allow appending a workspace / incremental building on top of a current session (pull requests are welcome!).

`build()` will build the session in the background via using tmux's detached state (`-d`).

After the session (workspace) is built, unless the user decided to load the session in the background via `tmuxp -d` (which is in the spirit of tmux's `-d`), we need to prompt the user to attach the session.

If the user is already inside a tmux client, which we detect via the `TMUX` environment variable being present, we will prompt the user to switch their current client to it.

If they're outside of tmux client - in a plain-old PTY - we will automatically `attach`.

If an exception is raised during the building of the workspace, tmuxp will prompt to cleanup (`$ tmux kill-session`) the session on the user's behalf. An exception raised during this process means it's not easy to predict how broken the session is.

Changed in version tmux: 2.6+

In tmux 2.6, the way layout and proportion's work when interfacing with tmux in a detached state (outside of a client) changed. Since tmuxp builds workspaces in a detached state, the `WorkspaceBuilder` isn't able to rely on functionality requiring awareness of session geometry, e.g. `set-layout`.

Thankfully, tmux is able to defer commands to run after the user performs certain actions, such as loading a client via `attach-session` or `switch-client`.

Upon client switch, `client-session-changed` is triggered¹.

References

`cli._validate_choices()`

Callback wrapper for validating click.prompt input.

Parameters `options (list)` – List of allowed choices

Returns callback function for `value_proc` in `click.prompt()`.

Return type `callable()`

Raises `click.BadParameter` –

¹ `cmd-switch-client.c` hook. GitHub repo for tmux. <https://github.com/tmux/tmux/blob/2.6/cmd-switch-client.c#L132>. Accessed April 8th, 2018.

Configuration

Finding

`config.is_config_file(extensions=['.yaml', '.yaml', '.json'])`

Return True if file has a valid config file type.

Parameters

- **filename** (*str*) – filename to check (e.g. `mysession.json`).
- **extensions** (*str* or *list*) – filetypes to check (e.g. `['.yaml', '.json']`).

Returns

Return type `bool`

`config.in_dir(extensions=['.yaml', '.yaml', '.json'])`

Return a list of configs in `config_dir`.

Parameters

- **config_dir** (*str*) – directory to search
- **extensions** (*list*) – filetypes to check (e.g. `['.yaml', '.json']`).

Returns

Return type `list`

`config.in_cwd()`

Return list of configs in current working directory.

If filename is `.tmuxp.py`, `.tmuxp.json`, `.tmuxp.yaml`.

Returns configs in current working directory

Return type `list`

Import and export

`config.validate_schema()`

Return True if config schema is correct.

Parameters **sconf** (*dict*) – session configuration

Returns

Return type `bool`

`config.expandshell()`

Return expanded path based on user's `$HOME` and `env`.

`os.path.expanduser()` and `os.path.expandvars()`

Parameters **path** (*str*) – path to expand

Returns path with shell variables expanded

Return type `str`

`config.expand(cwd=None, parent=None)`

Return config with shorthand and inline properties expanded.

This is necessary to keep the code in the `WorkspaceBuilder` clean and also allow for neat, short-hand configurations.

As a simple example, internally, tmuxp expects that config options like `shell_command` are a list (array):

```
'shell_command': ['htop']
```

tmuxp configs allow for it to be simply a string:

```
'shell_command': 'htop'
```

Kaptan will load JSON/YAML files into python dicts for you.

Parameters

- **sconf** (*dict*) – the configuration for the session
- **cwd** (*str*) – directory to expand relative paths against. should be the dir of the config directory.
- **parent** (*str*) – (used on recursive entries) start_directory of parent window or session object.

Returns

Return type *dict*

`config.inline()`

Return config in inline form, opposite of `config.expand()`.

Parameters **sconf** (*dict*) –

Returns configuration with optional inlined configs.

Return type *dict*

`config.trickle()`

Return a dict with “trickled down” / inherited config values.

This will only work if config has been expanded to full form with `config.expand()`.

tmuxp allows certain commands to be default at the session, window level. `shell_command_before` trickles down and prepends the `shell_command` for the pane.

Parameters **sconf** (*dict*) – the session configuration.

Returns

Return type *dict*

`config.import_teamocil()`

Return tmuxp config from a `teamocil` yaml config.

Parameters **sconf** (*dict*) – python dict for session configuration

Notes

Todos:

- change 'root' to a cd or start_directory
- width in pane -> main-pain-width
- with_env_var
- clear
- cmd_separator

`config.import_tmuxinator()`

Return tmuxp config from a `tmuxinator` yaml config.

Parameters `sconf` (*dict*) – python dict for session configuration.

Returns

Return type `dict`

Workspace Builder

class `tmuxp.workspacebuilder.WorkspaceBuilder` (*sconf*, *server=None*)

Load workspace from session `dict`.

Build tmux workspace from a configuration. Creates and names windows, sets options, splits windows into panes.

The normal phase of loading is:

1. `kaptan` imports json/yaml/ini. `.get()` returns python `dict`:

```
import kaptan
sconf = kaptan.Kaptan(handler='yaml')
sconf = sconf.import_config(self.yaml_config).get()
```

or from config file with extension:

```
import kaptan
sconf = kaptan.Kaptan()
sconf = sconf.import_config('path/to/config.yaml').get()
```

`kaptan` automatically detects the handler from filenames.

2. `config.expand()` `sconf` inline shorthand:

```
from tmuxp import config
sconf = config.expand(sconf)
```

3. `config.trickle()` passes down default values from session -> window -> pane if applicable:

```
sconf = config.trickle(sconf)
```

4. (You are here) We will create a `libtmux.Session` (a real `tmux(1)` session) and iterate through the list of windows, and their panes, returning full `libtmux.Window` and `libtmux.Pane` objects each step of the way:

```
workspace = WorkspaceBuilder(sconf=sconf)
```

It handles the magic of cases where the user may want to start a session inside tmux (when `$TMUX` is in the env variables).

build (*session=None*)

Build tmux workspace in session.

Optionally accepts *session* to build with only session object.

Without *session*, it will use `libtmux.Server` at `self.server` passed in on initialization to create a new `Session` object.

Parameters *session* (`libtmux.Session`) – session to build workspace in

config_after_window (*w, wconf*)

Actions to apply to window after window and pane finished.

When building a tmux session, sometimes its easier to postpone things like setting options until after things are already structurally prepared.

Parameters

- *w* (`libtmux.Window`) – window to create panes for
- *wconf* (*dict*) – config section for window

iter_create_panes (*w, wconf*)

Return `libtmux.Pane` iterating through window config dict.

Run `shell_command` with `$ tmux send-keys`.

Parameters

- *w* (`libtmux.Window`) – window to create panes for
- *wconf* (*dict*) – config section for window

Returns Newly created pane, and the section from the tmuxp configuration that was used to create the pane.

Return type tuple of (`libtmux.Pane`, *pconf*)

iter_create_windows (*s*)

Return `libtmux.Window` iterating through session config dict.

Generator yielding `libtmux.Window` by iterating through `sconf['windows']`.

Applies `window_options` to window.

Parameters *session* (`libtmux.Session`) – session to create windows in

Returns Newly created window, and the section from the tmuxp configuration that was used to create the window.

Return type tuple of (`libtmux.Window`, *wconf*)

`workspacebuilder.freeze()`

Freeze live tmux session and Return session config dict.

Parameters *session* (`libtmux.Session`) – session object

Returns tmuxp compatible workspace config

Return type dict

Exceptions

exception `tmuxp.exc.EmptyConfigException`
Configuration is empty.

exception `tmuxp.exc.ConfigError`
Error parsing tmuxp configuration dict.

exception `tmuxp.exc.BeforeLoadScriptError` (*returncode, cmd, output=None*)
Exception replacing `subprocess.CalledProcessError` for `tmuxp.util.run_before_script()`.

exception `tmuxp.exc.BeforeLoadScriptNotExists` (**args, **kwargs*)

1.14.7 History

Here you can find the recent changes to tmuxp

tmuxp 1.5.5 (2020-07-26)

- [#616](#) (via: [#599](#)) New command: `tmuxp ls`
List commands available via config directory. If the config is printed, it's loadable via `tmuxp load configfilename` without needing to type the full filepath. Thank you @pythops!
- [#480](#) Fix typo, thanks @jstoja
- [#578](#) Fix typo, thanks @mauroporras
- [#519](#) Fix typo, thanks @timgates42
- [#506](#) Fix Makefile typo, thanks @wolfgangpfner
- [#619](#) Update isort to 5.x
- Travis: Only run on master and PRs one time
- Travis: Add caching for tmux builds
- Travis: Test 2.9 and 3.0a
- [#613](#): Move from Pipenv to Poetry

tmuxp 1.5.4 (2019-11-06)

- [#500](#): Fix window focus
- Fix travis CI builds for python 3.7

tmuxp 1.5.3 (2019-06-06)

- [#377](#): Include examples in source distribution package

tmuxp 1.5.2 (2019-06-02)

- Loosen libtmux version constraint to >0.8 and <0.9 (libtmux 0.8.2 released today)
- [#484](#) CHANGES converted to plain reStructuredText
- [#491](#) `tmuxp freeze` will now infer active session with freezing
- [#490](#) Fix XDG's `$XDG_CONFIG_HOME` behavior
- [#483](#), [#482](#), [#480](#) Doc fixes
- [#487](#) Simplifying handling of configs with no panes (Fixes [#470](#))

tmuxp 1.5.1 (2019-02-18)

- Add tests/*.sh scripts to MANIFEST.in to include them in Pypi package.
- Include twine to dev packages on requirements and Pipfile files.

tmuxp 1.5.0 (2018-10-02)

- Support Click 7.0
- Remove unused `__future__` imports
- [#471](#) Update libtmux 0.8.0 -> 0.8.1
- [#404](#) from @anddam, support XDG base directory
- Sort imports
- Add configuration and make command for isort.
- Add sphinxcontrib-napoleon.
- Assure *requirements/dev.txt* dependencies are in *Pipfile*
- Update sphinx, releases to latest version
- Sync *requirements/*.txt* dependencies with *Pipfile*.
- Update docstring style to use numpy-style documentation. This enhances readability and plays nicely with sphinx documentation.
- Documentation overhaul.
 - Areas like `tmuxp.cli.load_workspace()` are now documented verbosely. This is so contributors helping on the project can more quickly gain situational awareness in this tricky area of code.

tmuxp 1.4.2 (2018-09-30)

- [#431](#) Include tests in source distribution

tmuxp 1.4.1 (2018-09-26)

- Loosen click restraint to <7

tmuxp 1.4.0 (2018-03-11)

- Bump libtmux to 0.8.0
- [#264](#) Update license from BSD to MIT
- [#348](#) Continuous integration updates and fixes for Travis CI
 - Update builds to use trusty
 - Remove older python 3 versions (before 3.6)
 - Update pypy versions
- [#349](#) flake8 via continuous integration
- Improve reliability of time-sensitive tests by using `while True` with a timeout.
- Update sphinx to 1.7.1
- Update alagitpull (sphinx theme) to 0.0.19. External websites open in new window.
- Update pytest to 3.4.1

tmuxp 1.3.5 (2017-11-10)

- [#312](#) Support for tmux 2.6 layout setting (via hooks) in the following scenarios:
 - loading outside tmux
 - loading inside tmux, via switch-client
 - loading inside tmux, with session in background (with -d), and reattaching/switching to after
 - loading session outside tmux in background with -d, and reattaching/switching after
- [#308](#) Fix bug where layouts don't correctly set on tmux 2.6
- Upgrade libtmux to 0.7.7

tmuxp 1.3.4 (2017-10-12)

- *before_script* now respects *start_directory* in the session root. This makes it easier to run things like *pipenv install* as a *before_script*.

tmuxp 1.3.3 (2017-10-07)

- Update libtmux to 0.7.5 for tmux 2.6 hotfix

tmuxp 1.3.2 (2017-08-20)

- [#184](#) - update libtmux to fix environmental variables in the session scope
- Update libtmux to 0.7.4
- Updates to pytest and pytest-rerunfailures

tmuxp 1.3.1 (2017-05-29)

- [#252](#) Fix bug where loading a session with a name matching a subset of current session causes undesired behavior.
- Update libtmux to 0.7.3
- Switch theme to alagitpull (alabaster subtheme)
- Remove unneeded doc dependency packages

tmuxp 1.3.0 (2017-04-27)

- [#239](#) Improve support for formatted options when freezing and using configs with them.
- [#236](#) Support for symlinked directories, thanks @rafi.
- [#235](#) Support for `options_after`, for setting options like `synchronize-panes`. Thanks @sebastianst.
- [#248](#) Drop python 2.6 support
- [#248](#) Upgrade libtmux to 0.7.1
- Upgrade colorama from 0.3.7 to 0.3.9

tmuxp 1.2.8 (2017-04-02)

- [#229](#) More helpful error message on systems missing tmux.
- Update libtmux from 0.6.4 to 0.6.5.

tmuxp 1.2.7 (2017-03-25)

- Support for OpenBSD.

tmuxp 1.2.6 (2017-02-24)

- [#218](#) Fix pane ordering by running `select-layout` before splits.

tmuxp 1.2.5 (2017-02-08)

- [#207](#) add custom tmuxp config directory via `TMUXP_CONFIGDIR` variable.
- [#199](#) support for running tmuxp on tmux master.
- update libtmux from 0.6.2 to 0.6.3.

tmuxp 1.2.4 (2017-01-13)

- [#198](#) bump click from 6.6 to 6.7
- [#195](#) pin packages for colorama and doc requirements

tmuxp 1.2.3 (2016-12-21)

- bump libtmux 0.6.0 to 0.6.1
- [#193](#) improve suppress history test, courtesy of @abeyer.
- [#191](#) documentation typo from @modille
- [#186](#) documentation typo from @joelwallis

tmuxp 1.2.2 (2016-09-16)

- [#181](#) Support tmux 2.3

tmuxp 1.2.1 (2016-09-16)

- [#132](#) Handle cases with invalid session names
- update libtmux from 0.5.0 to 0.6.0

tmuxp 1.2.0 (2016-06-16)

- [#65](#) Ability to specify `options` and `global_options` via configuration. Also you can specify environment variables via that.

Include tests and add example.

tmuxp 1.1.1 (2016-06-02)

- #167 fix attaching multiple sessions
- #165 fix typo in error output, thanks @fpietka
- #166 add new docs on zsh/bash completion
- Add back `tmuxp -V` for version info

tmuxp 1.1.0 (2016-06-01)

- #160 load tmuxp configs by name
- #134 Use `click` for command-line completion, Rewrite command line functionality for importing, config finding, conversion and prompts.
- Remove `-l` from `tmuxp import tmuxinator|teamocil`
- #158 `argparse` bug overcome by switch to `click`

tmuxp 1.0.2 (2016-05-25)

- #163 fix issue re-attaching sessions that are already loaded
- #159 improved support for `tmuxinator` imports, from @fpietka.
- #161 readme link fixes from @Omeryl.

tmuxp 1.0.1 (2016-05-25)

- switch to `readthedocs.io` for docs
- #157 bump `libtmux` to 0.4.1

tmuxp 1.0.0-rc1 (2016-05-25)

- version jump 0.11.1 to 1.0
- tests moved to `py.test` framework
- `libtmux` core split into its own project
- #145 Add new-window command functionality, @ikirudennis
- #146 Optionally disable shell history suppression, @kmactavish
- #147 Patching `unittest` timing for shell history suppression
- move doc building, tests and watcher to `Makefile`
- update `.tmuxp.yaml` and `.tmuxp.json` for `Makefile` change
- overhaul README

tmuxp 0.11.0 (2016-02-29)

- #137 Support for environment settings in configs, thanks @tasdomas
- Spelling correction, thanks @sehe.

tmuxp 0.10.0 (2016-01-30)

- #135 Load multiple tmux sessions at once, thanks @madprog.
- #131 #133 README and Documentation fixes

tmuxp 0.9.3 (2016-01-06)

- switch to `.venv` for virtualenv directory to not conflict with `.env` (used by `autoenv`).
- #130 move to `entr(1)` for file watching in tests. update docs.
- [compatibility] Support `Anaconda Python 2` and `3`

tmuxp 0.9.2 (2015-10-21)

- #122 Update to support tmux 2.1, thank you @estin.
- use travis container infrastructure for faster tests
- change test in workspace builder test to use `top(1)` instead of `man(1)`. `man(1)` caused errors on some systems where `PAGER` is set.

tmuxp 0.9.1 (2015-08-23)

- #119 Add fix python 3 for `sysutils/pytmuxp` on FreeBSD ports. See GH issue 119 and #201564 @ FreeBSD Bugzilla. Thanks Ruslan Makhmatkhanov.

tmuxp 0.9.0 (2015-07-08)

- Renamed `config.expandpath` to `config.expandshell`.
- compat 2.7/3.3 wrapper for `EnvironmentVarGuard` for testing.
- You can now use environment variables inside of `start_directory`, `before_script`, `shell_command_before`, `session_name` and `window_name`.
- [examples]: add example for environmental variables, `examples/env-variables.json` and `examples/env-variables.yaml`.
- #110 de-vendorize `colorama`. Thanks @marbu.
- #109 fix failure of `test_pane_order` on fedora machines from @marbu
- #105 append `.txt` extension to manuals (repo only) from @yegortimoshenko.
- #107 Fix `Server.attached_sessions` return type by @thomasballinger.
- update travis to use new tmux git repository.

tmuxp 0.8.1 (2015-05-09)

- [testing]: fix sniffer test runner in python 3
- new animated image demo for RTD and README

tmuxp 0.8.0 (2015-05-07)

- version bump 0.1.13 -> 0.8.0
- tmux 2.0 support
- Fix documentation for `:meth:Session.switch_client()`.
- Add `--log-level` argument.
- Refactor `{Server, Session, Window, Pane}.tmux` into:
 - `Server.cmd()`
 - `Session.cmd()`
 - `Window.cmd()`
 - `Pane.cmd()`(See conversation at <https://github.com/bitprophet/dotfiles/issues/5>)
- Refactor `util.tmux` into `util.tmux_cmd()`.

tmuxp 0.1.13 (2015-03-25)

- Remove `package_metadata.py` in favor of `__about__.py`.
- `scent.py` for building docs
- docutils from 0.11 to 0.12
- `bootstrap_env.py` will check for linux, darwin (OS X) and windows and install the correct `sniffer` file watcher plugin.
- testsuite for cli uses `tempfile.mkdtemp()` instead `TMP_DIR` (which resolved to `.tmuxp` in the testsuite directory).
- replace `watchingtestrunner` with `sniffer` in examples. `.tmuxp.conf` and `.tmux.json` updated
- updates to doc links
- make `checkbuild` for verifying internal / intersphinx doc references.
- Add Warning tmux versions less than 1.4 from [@techtonik](#).
- Add documentation on leading space in `send_keys` from [@thomasballinger](#).
- Update about page from teamocil and erb support from [@raine](#).

tmuxp 0.1.12 (2014-08-06)

- [config] `config.expand()` now resolves directories in configuration via `os.path.expanduser()` and `os.path.expandvars()`.
- [config] `config.expandpath()` for helping resolve paths.
- improved support for loading tmuxp project files from outside current working directory. e.g.

```
$ tmuxp load /path/to/my/project/.tmuxp.yaml
```

Will behave better with relative directories.

tmuxp 0.1.11 (2014-04-06)

- `before_script` now loads relative to project directory with `./`.
- Use `bootstrap_env.py` in tmuxp's `.tmuxp.yaml` and `.tmuxp.json` project files.
- Improvements to `util.run_before_script()`, `exc.BeforeLoadScriptFailed` behavior to print `stdout` and return `stderr` is a non-zero exit is returned.
- `run_script_before` has moved to `util`.
- `BeforeLoadScriptFailed` and `BeforeLoadScriptNotExists` has moved to the `exc` module.
- Tests for `run_script_before` refactored.

tmuxp 0.1.10 (2014-04-02)

- 2 bug fixes and allow panes with no shell commands to accept options, thanks for these 3 patches, [@ThiefMaster](#):
- [#73](#) Fix an error caused by spaces in `start_directory`.
- [#77](#) Fix bug where having a `-` in a `shell_command` would caused a build error.
- [#76](#) Don't require `shell_command` to pass options to panes (like `focus: true`).

tmuxp 0.1.9 (2014-04-01)

- The `--force` was not with us.

tmuxp 0.1.8 (2014-03-30)

- [#72](#) Create destination directory if it doesn't exist. Thanks [@ThiefMaster](#).
- New context manager for tests, `temp_session`.
- New testsuite, `testsuite.test_utils` for testing testsuite tools.
- New command, `before_script`, which is a file to be executed with a return code. It can be a bash, perl, python etc. script.
- [#56](#) `python_api_quickstart`

tmuxp 0.1.7 (2014-02-25)

- [#55](#) where tmuxp would crash with letter numbers in version. Write tests.

tmuxp 0.1.6 (2014-02-08)

- `Window.split_window()` now allows `-c start_directory`.
- [#35](#) Builder will now use `-c start_directory` to create new windows and panes.

This removes a hack where `default-path` would be set for new pane and window creations. This would bleed into tmux user sessions after creations.

tmuxp 0.1.5-1 (2014-02-05)

- [#49](#) bug where `package_manifest.py` missing from `MANIFEST.in` would cause error installing.

tmuxp 0.1.5 (2014-02-05)

- section heading normalization.
- tao of tmux section now treated as a chapter. tao of tmux may be split off into its own project.
- use conventions from [tony/cookiecutter-pypackage](#).

tmuxp 0.1.4 (2014-02-02)

- Fix `$ tmuxp freeze` CLI output.
- Update `_compat` support module.
- Fix extra space in [PEP 263](#).

tmuxp 0.1.3 (2014-01-29)

- [#48](#) Fix Python 3 CLI issue.
- [#48](#) `$ tmuxp` without option raises an error.
- Add space before send-keys to not populate bash and zsh history.

tmuxp 0.1.2 (2014-01-08)

- now using `werkzeug` / `flask` style testsuites.
- [#43](#) Merge `tmuxp -d` for loading in detached mode. Thanks [roxit](#).

tmuxp 0.1.1 (2013-12-25)

- #32 Fix bug where special characters caused unicode caused unexpected outcomes loading and freezing sessions.

tmuxp 0.1.0 (2013-12-18)

- fix duplicate print out of filename with using `tmuxp load ..`
- version to 0.1. No `--pre` needed. Future versions will not use rc.

tmuxp 0.1-rc8 (2013-12-17)

- `unicode_literals`
- Move py2/py3 compliancy code to `_compat`.

tmuxp 0.1-rc7 (2013-12-07)

- #33 Partial rewrite of `config.expand()`.
- `tmuxp` will exit silently with `Ctrl-c`.

tmuxp 0.1-rc6 (2013-12-06)

- #31 [examples] from stratoukos add `window_index` option, and example.

tmuxp 0.1-rc5 (2013-12-04)

- #28 `shell_command_before` in session scope of config causing duplication. New test.
- #26 #29 for OS X tests. Thanks stratoukos.
- #27 `$ tmuxp freeze` raises unhelpful message if session doesn't exist.

tmuxp 0.1-rc4 (2013-12-03)

- fix bug were `focus: true` would not launch sessions when using `$ tmuxp load` in a `tmux` session.

tmuxp 0.1-rc3 (2013-12-03)

- #25 `focus: true` not working in panes. Add tests for focusing panes in config.
- `Pane.select_pane()`.
- add new example for `focus: true`.

tmuxp 0.1-rc2 (2013-11-23)

- [#23](#) fix bug where workspace would not build with pane-base-index set to 1. Update tests to fix if pane-base-index is not 0.
- removed `$ tmuxp load --list` functionality. Update [Quickstart](#) accordingly.

tmuxp 0.1-rc1 (2013-11-23)

- [pep8](#) and [pep257](#) in unit tests.
- Changelog will now be updated on a version basis, use [pep440](#) versioning.

tmuxp 0.1-dev (2013-11-21)

- `Session.show_options()`, `Session.show_option()` now accept `g` to pass in `-g`.

tmuxp 0.1-dev (2013-11-20)

- `Window.show_window_options()`, `Window.show_window_option()` now accept `g` to pass in `-g`.
- [#15](#) Behavioral changes in the WorkspaceBuilder to fix pane ordering.
- [#21](#) Error with unit testing python 2.6 python configuration tests. Use `tempfile` instead.
- WorkspaceBuilder tests have been improved to use `async` better.

tmuxp 0.1-dev (2013-11-17)

- fix a bug where missing tmux didn't show correct warning.

tmuxp 0.1-dev (2013-11-15)

- Travis now tests python 2.6 as requirement and not allowed to fail.

tmuxp 0.1-dev (2013-11-13)

- [#19](#) accept `-y` argument to answer yes to questions.
- `cli.SessionCompleter()` no longer allows a duplicate session after one is added.
- ongoing work on *The Tao of tmux*.

tmuxp 0.1-dev (2013-11-09)

- [translation] documentation in Chinese from wrongwaycn.
- More work done on the *The Tao of tmux* page.
- `Pane.split_window()` for splitting Window at target-pane location.

tmuxp 0.1-dev (2013-11-08)

- [freeze] - `$ tmuxp freeze` will now freeze a window with a `start_directory` when all panes in a window are inside the same directory.
- [config] `config.inline()` will now turn panes with no other attributes and 1 command into a single item value.

```
- panes:
  - shell_command: top

# will now inline to:

- panes
  - top

This will improve ``$ tmuxp freeze``
```

tmuxp 0.1-dev (2013-11-07)

- Remove old logger (based on `tornado's log.py`), replace with new, simpler one.
- fix `teamocil` import.
- support import `teamocil root` to `start_directory`.

tmuxp 0.1-dev (2013-11-06)

- tagged v0.0.37. Many fixes. Python 2.6 support. Will switch to per-version changelog after 0.1 release.
- support for blank panes (`null`, `pane`, `blank`) and panes with empty strings.
- `tmuxp freeze` supports exporting to blank panes.
- `tmuxp freeze` will now return a blank pane for panes that would previously return a duplicate shell command, or generic python, node interpreter.

tmuxp 0.1-dev (2013-11-05)

- Support for `[-L socket-name]` and `[-S socket-path]` in autocompletion and when loading. Note, switching client into another socket may cause an error.
- Documentation tweaking to *API Reference*, *The Tao of tmux*.
- `pep257`, `pep8`.

tmuxp 0.1-dev (2013-11-04)

- pep257, pep8.
- tagged version v0.0.36.

tmuxp 0.1-dev (2013-11-02)

- Many documentation, pep257, pep8 fixes
- move old Server methods `__list_panes()`, `__list_windows` and `__list_sessions` into the single underscore.
- #12 fix for `$ tmuxp freeze` by @finder.
- Support for spaces in `$ tmuxp attach-session` and `$ tmuxp kill-session`, and `$ tmuxp freeze`.
- [config] support for relative paths of `start_directory`. Add an update config in *Start Directory* on *Example Configurations*.

tmuxp 0.1-dev (2013-11-01)

- New servers for Server arguments `socket_name`, `socket_path`, `config_file`.
- Server support for `-2` with `colors=256` and `colors=8`.
- `$ tmuxp -2` for forcing 256 colors and `tmuxp -8` for forcing 88.
- [config] Concatenation with `start_directory` via `config.trickle()` if window `start_directory` is alphanumeric / relative (doesn't start with `/`). See *Example Configurations* in *start directory*.
- Fix bug with import teamocil and tmuxinator
- Improve quality of tmuxinator imports. Especially `session_name` and `start_directory`.
- Allow saving with `~` in file destination.

tmuxp 0.1-dev (2013-10-31)

- `util.is_version()`
- correctly `config.trickle()` the `start_directory`.
- get `start_directory` working for configs
- fix `:meth:Window.kill_window()` target to `session_id:window_index` for compatibility and pass tests.
- [examples]: Example for `start_directory`.
- fix bug where first and second window would load in mixed order
- `Window.move_window()` for moving window.
- doc overhaul. front page, renamed `orm_al.rst` to `internals.rst`.

tmuxp 0.1-dev (2013-10-30)

- fix bug where if inside tmux, loading a workspace via `switch_client` wouldn't work.
- fix bug where `tmuxp load .` would return an error instead of a notice.
- `tmuxp freeze <filename>` experimental
- tmuxp now has experimental support for freezing live sessions.
- `Window.kill_window()`
- support for `start_directory` (work in progress)

tmuxp 0.1-dev (2013-10-29)

- `Window.select_pane()` now accepts `-l`, `-U`, `-D`, `-L`, `-R`.
- support for `automatic-rename` option.
- 3 new *Example Configurations*, 'main-pane-height', 'automatic-rename', and 'shorthands'.
- enhancements to prompts
- `tmuxp import` for teamocil and tmuxinator now has a wizard and offers to save in JSON or YAML format.
- [b6c2e84] Fix bug where `tmuxp load w/` session already loaded would switch/attach even if no was entered
- when workspace loader crashes, give option to kill session, attach it or detach it.
- `tmux 1.8 set-option / set-window-options command target-window` fix.
- `WorkspaceBuilder` now has `.session` attribute accessible publicly.
- tmux will now use `Session.switch_client()` and `Session.attach_session()` to open new sessions instead of `os.exec`.
- [config] tmuxp now allows a new shorter form for panes. Panes can just be a string. See the shorthand form in the *Example Configurations* section.
- [config] support loading `.yaml`.

tmuxp 0.1-dev (2013-10-28)

- fix `tmuxp load . fixed`
- fix `tmuxp convert <file> fixed`.
- pep257 fixes.
- `Pane` now has `Pane.set_width()` and `Pane.set_height()`.
- `./run_tests.py --tests` now automatically prepends `tmuxp.testsuite` to names.
- `Window.tmux()` and `Pane.tmux()` will automatically add their `{window/pane}_id` if one isn't specific.

tmuxp 0.1-dev (2013-10-27)

- `argcomplete` overhaul for CLI bash completion.
- `tmuxp load`, `tmuxp convert` and `tmuxp import` now support relative and full filenames in addition to searching the config directory.

tmuxp 0.1-dev (2013-10-26)

- initial version of `tmuxinator` and `teamocil` config importer. it does not support all options and it not guaranteed to fully convert window/pane size and state.
- `config.in_dir()` supports a list of extensions for filetypes to search, i.e. `['.yaml', '.json']`.
- `config.is_config_file()` now supports extensions argument as a string also.
- fix `$ tmuxp load -l` to work correctly alongside `$ tmuxp load filename`.

tmuxp 0.1-dev (2013-10-25)

- fix bug where `-v` and `--version` wouldn't print version.
- property handle case where no tmux server exists when `attach-session` or `kill-session` is used.
- test fixtures and initial work for importing `tmuxinator` and `teamocil` configs

tmuxp 0.1-dev (2013-10-24)

- clean out old code for `automatic-rename` option. it will be reimplemented fresh.
- check for `oh-my-zsh` when using `$SHELL zsh`. Prompt if `DISABLE_AUTO_TITLE` is unset or set to `true`.
- `tmuxp` can now `$ tmuxp convert <file>` from JSON `<=>` YAML, back and forth.
- New examples in JSON. Update the [Example Configurations](#) page in the docs.
- `[dev] .tmuxp.json` now exists as a config for `tmuxp` development and as an example.
- Fix bug where `tmuxp kill-session` would give bad output
- Fix bug in tab completion for listing sessions with no tmux server is active.

tmuxp 0.1-dev (2013-10-23)

- `zsh/bash/tcsh` completion improvements for tab-completion options
- `tmuxp kill-session` with tab-completion.
- `tmuxp attach-session` with tab-completion. Attach session will `switch-client` for you if you are inside of of a tmux client.
- `tmuxp load` for loading configs.
- unit test fixes.

tmuxp 0.1-dev (2013-10-21)

- Make 1.8 the official minimum version, give warning notice to upgrade tmux if out of date
- Fix regression causing unexpected build behavior due to unremoved code supporting old tmux versions.
- Added 2 new examples to the *Example Configurations* page.
- Examples now have graphics
- `$ tmuxp -v` will print the version info.

tmuxp 0.1-dev (2013-10-19)

- tmuxp will now give warning and `sys.exit()` with a message if tmux not found in system PATH
- internal overhaul of `Server`, `Session`, `Window`, and `Pane` continues.
 - `Server` has `@property Server.sessions()`, which is forward to `Server.list_sessions()` (kept to keep tmux commands in serendipity with api), `Server._list_sessions()` returns dict object from `Server.__list_sessions()` tmux command. `Server.__list_sessions()` exists to keep the command layered so it can be tested against in a matrix with travis and compatibility methods can be made.
 - `Session` now has `@property Session.windows()` returning a list of `Window` objects via `Session.list_windows()`. `@property Session._windows()` to `Session._list_windows()` to return a list of dicts without making objects.
 - `Window` now has `@property Window.panes()` returning a list of `Pane` objects via `Window.list_panes()`. `@property Window._panes()` to `Window._list_panes()` to return a list of dicts without making objects.

tmuxp 0.1-dev (2013-10-18)

- internal overhaul of `Server`, `Session`, `Window`, and `Pane`.
 - `Session`, `Window` and `Pane` now refer to a data object in `Server` internally and always pull the freshest data.
 - A lot of code and complexity regarding setting new data for objects has been reduced since objects use their unique key identifier to filter their objects through the windows and panes in `Server` object.
 - `Server` object is what does the updating now.
- [project] some research into supporting legacy tmux versions. tmux 1.6 and 1.7 support seem likely eventually if there is enough demand.
- python 3 support

tmuxp 0.1-dev (2013-10-17)

- updated README docs with new project details, screenshots
- new example `.tmuxp.yaml` file updated to include development workflow. Removed nodemon as the tool for checking files for now.
- Support for switching sessions from within tmux. In both cases after the session is built and if session already exists.

tmuxp 0.1-dev (2013-10-16)

- use `util.which()` from `salt.util` to find tmux binary.
- add `MANIFEST.in`, fix issue where package would not install because missing file
- bash / zsh completion.
- New page on [About](#).
- Updates to *The Tao of tmux* page.
- add vim modeline for rst to bottom of this page
- Server is now a subclass of `util.TmuxObject`.
- subclasses of `util.TmuxRelationalObject`, `Server`, `Session`, `Window`, `Pane` now have `util.TmuxRelationalObject.getById()` (similar to `.get()` in [backbone.js](#) collection), `util.TmuxRelationalObject.where()` and `util.TmuxRelationalObject.findWhere()` (`.where()` and `.findWhere()` in [underscore.js](#)), to easily find child objects.
- tmux object mapping has been split into `util.TmuxMappingObject`. The mapping and the relational has been decoupled to allow `Server` to have children while not being a dict-like object.
- `Server`, `Session`, `Window`, `Pane` now explicitly mixin subclasses.

tmuxp 0.1-dev (2013-10-15)

- new theme
- initial examples, misc. updates, front page update.
- support for `$ tmux .` to load `.tmuxp.{yaml/json/py}` in current working directory.
- support for `socket-name (-L)` and `socket-path (socket-path)`
- [config] Support for 1-command pane items.

```
session_name: my session
windows:
- window_name: hi
  panes:
  - bash
  - htop
```

- If session name is already exists, prompt to attach.

tmuxp 0.1-dev (2013-10-14)

- can now `-l` to list configs in current directory and `$HOME/.tmuxp`
- `tmuxp` can now launch configs and build sessions
- new exceptions
- `config.check_consistency()` to verify and diagnose issues with config files.
- `cli.startup()`
- `config.is_config_file()`
- `config.in_dir()`
- `config.in_cwd()`

tmuxp 0.1-dev (2013-10-13)

- `config.inline()` to produce far far better looking config exports and tests.
- `Pane.resize_pane()` and tests
- documentation fixes and updates
- `Session.refresh()`, `Window.refresh()`, `Pane.refresh()`.
- `Server.find()`, `Session.find()`, `Window.find()`.

tmuxp 0.1-dev (2013-10-12)

- Test documentation updates
- Builder is now `WorkspaceBuilder` + tests. - `WorkspaceBuilder` can build panes - `WorkspaceBuilder` can build windows and set options
- `Window.show_window_options()`, `Window.show_window_option()`, `Window.set_window_option()`
- `Session.show_options()`, `Session.show_option()`, `Session.set_option()`

tmuxp 0.1-dev (2013-10-11)

- More preparation for builder / session maker utility.
- test runner and test suite overhaul.
- Documentation for development environment and test runner updated.
- Travis now tests against `tmux 1.8` and latest source. Door open for future testing against python 3 and earlier `tmux` versions in the future.
- Quiet logger down in some areas
- `__future__` imports for future python 3 compatibility
- `setup.py` import `__version__` via regex from `tmuxp` package
- move beginnings of cli to `tmuxp.cli`

tmuxp 0.1-dev (2013-10-09)

- New logging module
- Removed dependency logutils
- Removed dependency sh

tmuxp 0.1-dev (2013-10-08)

- switch to semver

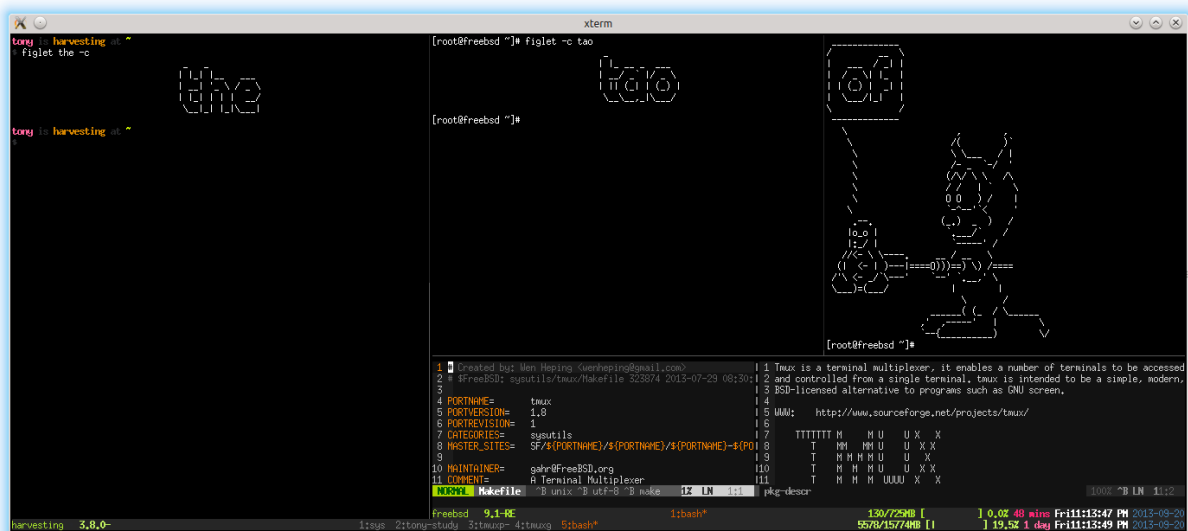
1.14.8 The Tao of tmux

Fig. 1: ISC-licensed terminal multiplexer.

tmux is geared for developers and admins who interact regularly with CLI (text-only interfaces)

In the world of computers, there are 2 realms:

1. The text realm
2. The graphical realm

tmux resides in the text realm. This is about fixed-width fonts and that old fashioned black terminal.

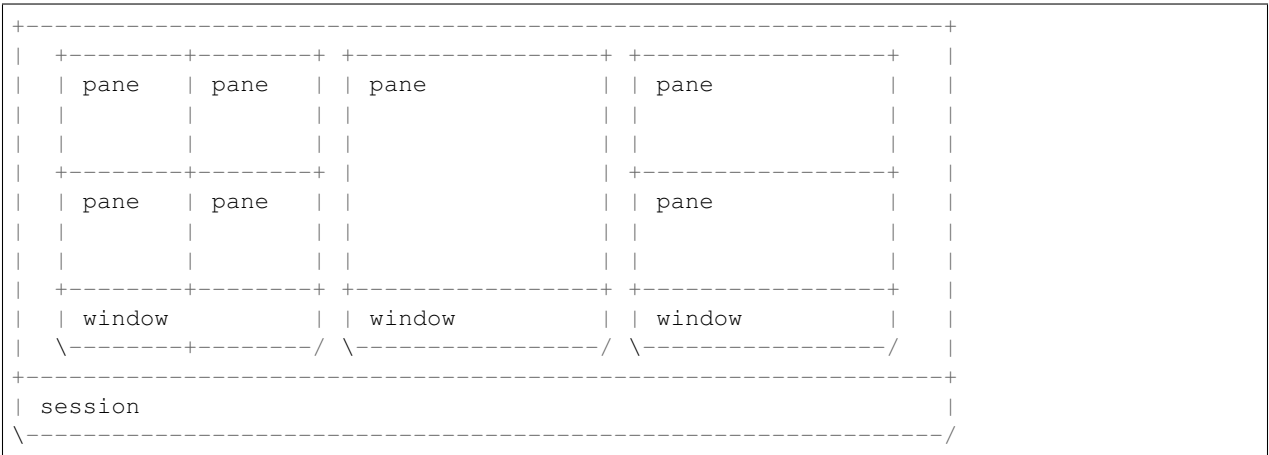
tmux is to the console what a desktop is to gui apps. It's a world inside the text dimension. Inside tmux you can:

- multitask inside the terminal, run multiple applications.
- have multiple command lines (pane) in the same window
- have multiple windows (window) in the workspace (session)
- switch between multiple workspaces, like virtual desktops

Thinking tmux

Text-based window manager

tmux	“Desktop”-Speak	Plain English
Multiplexer	Multi-tasking	Multiple applications simultaneously.
Session	Desktop	Applications are visible here
Window	Virtual Desktop or applications	A desktop that stores it own screen
Pane	Application	Performs operations



- 1 *Server*.
 - has 1 or more *Session*.
 - * has 1 or more *Window*.
 - has 1 or more *Pane*.

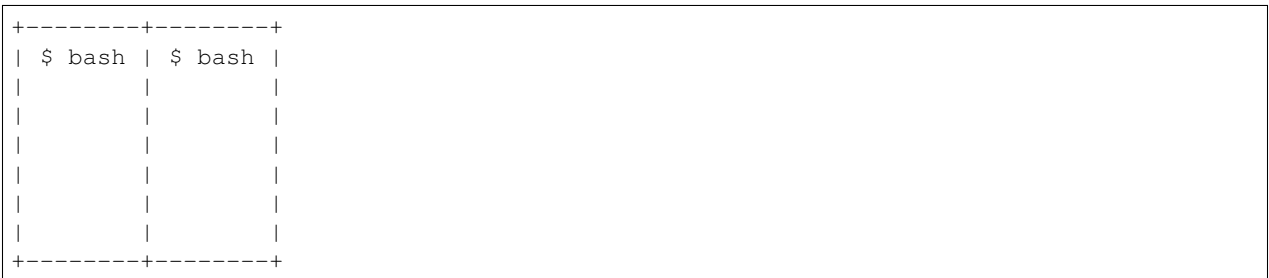
See also:

Glossary has a dictionary of tmux words.

CLI Power Tool

Multiple applications or terminals to run on the same screen by splitting up 1 terminal into multiple.

One screen can be used to edit a file, and another may be used to `$ tail -F a logfile`.



```

+-----+-----+
| $ bash | $ bash |
|         |         |
|         |         |
+-----+-----+
| $ vim  | $ bash  |
|         |         |
|         |         |
+-----+-----+

```

tmux supports as many terminals as you want.

```

+-----+-----+
| $ bash | $ bash |
|         |         |
|         |         | /-----\
+-----+-----+ --> | 'switch-window 2' |
| $ bash | $ bash | \-----/
|         |         |
|         |         |
+-----+-----+
| '1:sys* 2:vim' |
+-----+-----+
|         |         |
|         |         | /-----/
|         |         | |
|         |         | |
|         |         | |
+-----+-----+
| $ vim |
|         |
|         |
+-----+-----+
| $ bash | $ bash |
|         |         |
|         |         |
+-----+-----+
| '1:sys 2:vim*' |
+-----+-----+

```

You can switch between the windows you create.

Resume everything later

You can leave tmux and all applications running (detach), log out, make a sandwich, and re-(attach), all applications are still running!

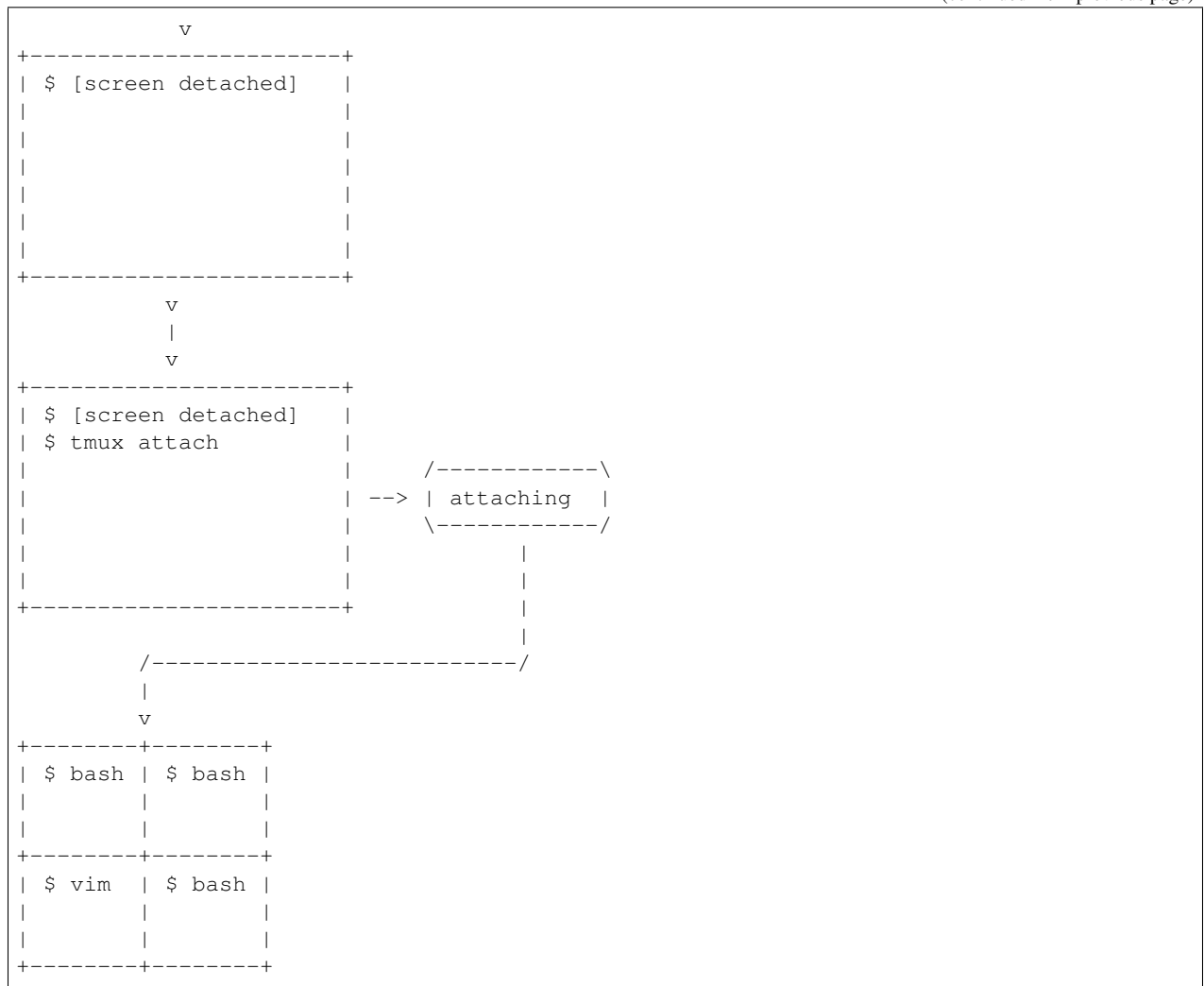
```

+-----+-----+
| $ bash | $ bash |
|         |         |
|         |         | /-----\
+-----+-----+ --> | detach |
| $ vim  | $ bash | | 'Ctrl-b b' |
|         |         | \-----/
|         |         |
|         |         |
+-----+-----+
|         |         |
|         |         | /-----/
|         |         | |
|         |         | |
|         |         | |
+-----+-----+

```

(continues on next page)

(continued from previous page)



Manage workflow

- System administrators monitor logs and services.
- Programmers like to have an editor open with a CLI nearby.

Applications running on a remote server can be launched inside of a tmux session, detached, and reattached next time your “train of thought” and work.

Multitasking. Preserving the thinking you have.

Installing tmux

tmux is packaged on most Linux and BSD systems.

For the freshest results on how to get tmux installed on your system, “How to install tmux on <my distro>” will do, as directions change and are slightly different between distributions.

This documentation is written for version **1.8**. It’s important that you have the latest stable release of tmux. The latest stable version is viewable on the [tmux homepage](#).

Mac OS X users may install that latest stable version of tmux through [MacPorts](#), [fink](#) or [Homebrew](#) (aka brew).

If **compiling from source**, the dependencies are [libevent](#) and [ncurses](#).

Using tmux

Start a new session

```
$ tmux
```

That’s all it takes to launch yourself into a tmux session.

Common pitfall

Running `$ tmux list-sessions` or any other command for listing tmux entities (such as `$ tmux list-windows` or `$ tmux list-panes`). This can generate the error “failed to connect to server”.

This could be because:

- tmux server has killed its’ last session, killing the server.
- tmux server has encountered a crash. (tmux is highly stable, this will rarely happen)
- tmux has not be launched yet at all.

The prefix key

Tmux hot keys have to be pressed in a special way. **Read this carefully**, then try it yourself.

First, you press the *prefix* key. This is `C-b` by default.

Release. Then pause. For less than second. Then type what’s next.

`C-b o` means: Press `Ctrl` and `b` at the same time. Release, Then press `o`.

Remember, prefix + short cut! `C` is `Ctrl` key.

Session Name

Sessions can be *named upon creation*.

```
$ tmux new-session [-s session-name]
```

Sessions can be *renamed after creation*.

Command	<code>\$ tmux rename-session <session-name></code>
Short cut	Prefix + <code>\$</code>

Window Name

Windows can be *named upon creation*.

```
$ tmux new-window [-n window-name]
```

Windows can be *renamed after creation*.

Command	<code>\$ tmux rename-window <new-name></code>
Short cut	Prefix + <code>,</code>

Creating new windows

Command	<code>\$ tmux new-window [-n window-name]</code>
Short cut	Prefix + <code>c</code> You may then rename window.

Traverse windows

By number

```
$ tmux select-window
```

Next

```
$ tmux next-window
```

Previous

```
$ tmux previous-window
```

Last-window

```
$ tmux last-window
```

Short cut	Action
n	Change to the next window.
p	Change to the previous window.
w	Choose the current window interactively.
0 to 9	Select windows 0 to 9.
M-n	Move to the next window with a bell or activity marker.
M-p	Move to the previous window with a bell or activity marker.

Move windows

Move window

```
$ tmux move-window [-t dst-window]
```

Swap the window

```
$ tmux swap-window [-t dst-window]
```

Short cut	Action
.	Prompt for an index to move the current window.

Move panes

```
$ tmux move-pane [-t dst-pane]
```

Short cut	Action
C-o	Rotate the panes in the current window forwards.
{	Swap the current pane with the previous pane.
}	Swap the current pane with the next pane.

Traverse panes

Shortcut to move between panes.

```
$ tmux last-window
```

```
$ tmux next-window
```

Short cut	Action
Up, Down	Change to the pane above, below, to the left, or to
Left, Right	the right of the current pane.

Recipe: tmux conf to hjkl commands, add this to your `~/ .tmux.conf`:

```
# h j k l pane traversal
bind h select-pane -L
bind j select-pane -D
bind k select-pane -U
bind l select-pane -R
```

Kill window

```
$ tmux kill-window [-t target-window]
```

Short cut	Action
&	Kill the current window.

Kill pane

```
$ tmux kill-pane [-t target-pane]
```

Short cut	Action
x	Kill the current pane.

Splitting windows into panes

```
$ tmux split-window [-c start-directory] <shell-command>
```

Tmux windows can be split into multiple panes.

Short cut	Action
%	Split the current pane into two, left and right.
"	Split the current pane into two, top and bottom.

Configuring tmux

Tmux can be configured via a configuration at `~/ .tmux.conf`.

Depending on your tmux version, there is different options available.

Vi-style copy and paste

```
# Vi copypaste mode
set-window-option -g mode-keys vi
bind-key -t vi-copy 'v' begin-selection
bind-key -t vi-copy 'y' copy-selection
```


Aggressive resizing for clients

```
setw -g aggressive-resize on
```

Reload config

<Prefix> + r.

```
bind r source-file ~/.tmux.conf \; display-message "Config reloaded."
```

Status lines

Tmux allows configuring a status line that displays system information, window list, and even pipe in the `stdout` of an application.

You can use `tmux-mem-cpu-load` to get stats (requires compilation) and `basic-cpu-and-memory.tmux`. You can pipe in a bash command to a tmux status line like:

```
$ (shell-command)
```

So if `/usr/local/bin/tmux-mem-cpu-load` outputs stats to `stdout`, then `$(tmux-mem-cpu-load)` is going to output the first line to the status line. The interval is determined by the `status-interval`:

```
set -g status-interval 1
```

Examples

- <https://github.com/tony/tmux-config> - works with tmux 1.5+. Supports screen's `ctrl-a` *The prefix key*. Support for system cpu, memory, uptime stats.
- Add yours, edit this page on github.

Reference

Short cuts

Tip: *The prefix key* is pressed before a short cut!

Short cut	Action
C-b	Send the prefix key (C-b) through to the application.
C-o	Rotate the panes in the current window forwards.
C-z	Suspend the tmux client.
!	Break the current pane out of the window.
"	Split the current pane into two, top and bottom.
#	List all paste buffers.
\$	Rename the current session.

continues on next page

Table 1 – continued from previous page

%	Split the current pane into two, left and right.
&	Kill the current window.
'	Prompt for a window index to select.
,	Rename the current window.
-	Delete the most recently copied buffer of text.
.	Prompt for an index to move the current window.
0 to 9	Select windows 0 to 9.
:	Enter the tmux command prompt.
;	Move to the previously active pane.
=	Choose which buffer to paste interactively from a list.
?	List all key bindings.
D	Choose a client to detach.
[	Enter copy mode to copy text or view the history.
]	Paste the most recently copied buffer of text.
c	Create a new window.
d	Detach the current client.
f	Prompt to search for text in open windows.
i	Display some information about the current window.
l	Move to the previously selected window.
n	Change to the next window.
o	Select the next pane in the current window.
p	Change to the previous window.
q	Briefly display pane indexes.
r	Force redraw of the attached client.
s	Select a new session for the attached client interactively.
L	Switch the attached client back to the last session.
t	Show the time.
w	Choose the current window interactively.
x	Kill the current pane.
{	Swap the current pane with the previous pane.
}	Swap the current pane with the next pane.
~	Show previous messages from tmux, if any.
Page Up	Enter copy mode and scroll one page up.
Up, Down	Change to the pane above, below, to the left, or to
Left, Right	the right of the current pane.
M-1 to M-5	Arrange panes in one of the five preset layouts: even-horizontal, even-vertical, main-horizontal, main-vertical
M-n	Move to the next window with a bell or activity marker.
M-o	Rotate the panes in the current window backwards.
M-p	Move to the previous window with a bell or activity marker.
C-Up, C-Down	Resize the current pane in steps of one cell.
C-Left, C-Right	
M-Up, M-Down	Resize the current pane in steps of five cells.
M-Left, M-Right	

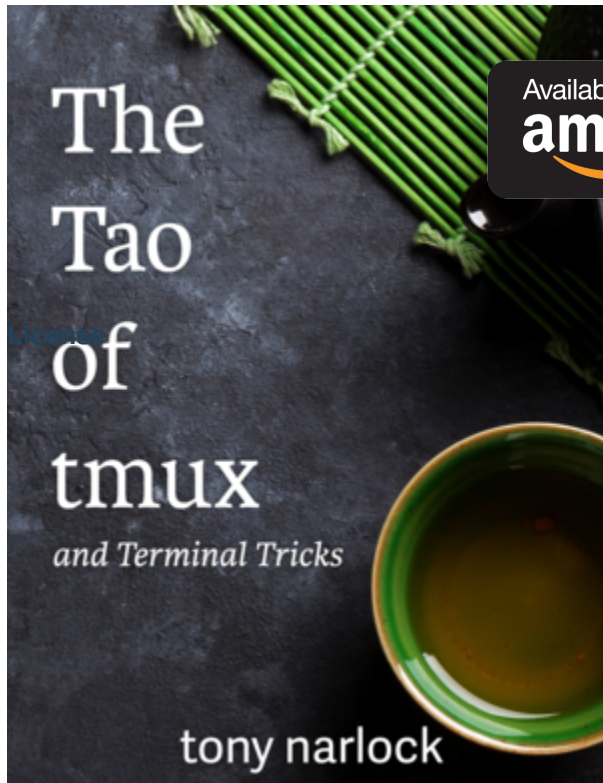
Source: tmux manpage¹.

To get the text documentation of a .1 manual file:

```
$ nroff -mdoc tmux.1|less
```

¹ <http://sourceforge.net/p/tmux/tmux-code/ci/master/tree/tmux.1>

The Book



The Tao of tmux is available on [Leanpub](#) and [Kindle](#) (Amazon).

Read and browse the book for [free on the web](#).

This page is licensed [Creative Commons BY-NC-ND 3.0 US](#).

1.14.9 Glossary

tmuxp A tool to manage workspaces with tmux. A pythonic abstraction of tmux.

tmux(1) The tmux binary. Used internally to distinguish tmuxp is only a layer on top of tmux.

kaptan configuration management library, see [kaptan on github](#).

Server Tmux runs in the background of your system as a process.

The server holds multiple *Session*. By default, tmux automatically starts the server the first time `$ tmux` is ran.

A server contains *session*'s.

tmux starts the server automatically if it's not running.

Advanced cases: multiple can be run by specifying `[-L socket-name]` and `[-S socket-path]`.

Client Attaches to a tmux *server*. When you use tmux through CLI, you are using tmux as a client.

Session Inside a tmux *server*.

The session has 1 or more *Window*. The bottom bar in tmux show a list of windows. Normally they can be navigated with `Ctrl-a [0-9]`, `Ctrl-a n` and `Ctrl-a p`.

Sessions can have a `session_name`.

Uniquely identified by `session_id`.

Window Entity of a *session*.

Can have 1 or more *pane*.

Panes can be organized with a layouts.

Windows can have names.

Pane Linked to a *Window*.

a pseudoterminal.

Target A target, cited in the manual as `[-t target]`
can be a session, window or pane.

PYTHON MODULE INDEX

t

tmuxp, [42](#)

Symbols

`_reattach()` (*tmuxp.cli method*), 35
`_validate_choices()` (*tmuxp.cli method*), 37

B

`BeforeLoadScriptError`, 42
`BeforeLoadScriptNotExists`, 42
`build()` (*tmuxp.workspacebuilder.WorkspaceBuilder method*), 41

C

`Client`, 71
`config_after_window()`
 (*tmuxp.workspacebuilder.WorkspaceBuilder method*), 41
`ConfigError`, 42

E

`EmptyConfigException`, 42
`expand()` (*tmuxp.config method*), 38
`expandshell()` (*tmuxp.config method*), 38

F

`freeze()` (*tmuxp.workspacebuilder method*), 41

G

`get_config_dir()` (*tmuxp.cli method*), 36
`get_teamocil_dir()` (*tmuxp.cli method*), 36
`get_tmuxinator_dir()` (*tmuxp.cli method*), 36

I

`import_teamocil()` (*tmuxp.config method*), 39
`import_tmuxinator()` (*tmuxp.config method*), 40
`in_cwd()` (*tmuxp.config method*), 38
`in_dir()` (*tmuxp.config method*), 38
`inline()` (*tmuxp.config method*), 39
`is_config_file()` (*tmuxp.config method*), 38
`iter_create_panes()`
 (*tmuxp.workspacebuilder.WorkspaceBuilder method*), 41

`iter_create_windows()`
 (*tmuxp.workspacebuilder.WorkspaceBuilder method*), 41

K

`kaptan`, 71

L

`load_workspace()` (*tmuxp.cli method*), 36

M

`module`
 tmuxp, 4, 30, 35, 42

P

`Pane`, 72

R

`run_before_script()` (*tmuxp.util method*), 35

S

`Server`, 71
`Session`, 71

T

`Target`, 72
`tmux(1)`, 71
`tmuxp`, 71
 module, 4, 30, 35, 42
`trickle()` (*tmuxp.config method*), 39

V

`validate_schema()` (*tmuxp.config method*), 38

W

`Window`, 72
`WorkspaceBuilder` (*class* *in*
 tmuxp.workspacebuilder), 40