Qtile Documentation

Release 0.10.5

Aldo Cortesi

March 11, 2016

Contents

CHAPTER 1

Getting started

1.1 Installing Qtile

1.1.1 Distro Guides

Below are the preferred installation methods for specific distros. If you are running something else, please see *Installing From Source*.

Installing on Arch Linux

Qtile is available on the AUR as:

Package Name	Description
qtile	stable branch (release)
qtile-python3-git	development branch

Using an AUR Helper

The preferred way to install Qtile is with an AUR helper. For example, if you use yaourt:

```
yaourt -S <package-name>
```

Using makepkg

The latest version of either package can be obtained by downloading a snapshot or cloning its repository:

```
# snapshot
curl -s https://aur.archlinux.org/cgit/aur.git/snapshot/<package-name>.tar.gz | tar -xvzf -
# or repository
git clone https://aur.archlinux.org/<package-name>.git
```

Next makepkg has to be called in the directory where the files were saved. It installs missing dependencies using pacman, builds the package, installs it and removes obsolete build-time dependencies afterwards:

```
cd <package-name>
makepkg -sri
```

Please see the ArchWiki for more information on installing packages from the AUR.

Installing on Fedora

Stable versions of Qtile are currently packaged for current versions of Fedora. To install this package, run:

```
dnf -y install qtile
```

Installing on Funtoo

Latest versions of Qtile are available on Funtoo with Python 2.7, 3.3, and 3.4 implementations. To install it, run:

```
emerge -av x11-wm/qtile
```

You can also install the development version from GitHub:

```
echo "x11-wm/qtile-9999 **" >> /etc/portage/package.accept_keywords
emerge -av qtile
```

Customize

You can customize your installation with the following useflags:

- dbus
- widget-google-calendar
- widget-imap
- widget-keyboardkbdd
- widget-launchbar
- widget-mpd
- widget-mpris
- widget-wlan

The dbus useflag is enabled by default. Disable it only if you know what it is and know you don't use/need it.

All widget-* useflags are disabled by default because these widgets require additional dependencies while not everyone will use them. Enable only widgets you need to avoid extra dependencies thanks to these useflags.

Visit Funtoo Qtile documentation for more details on Qtile installation on Funtoo.

Installing on Ubuntu

There are no packages for currently released versions of qtile. However, on wily and above (and debian unstable), the dependencies are available via:

```
sudo apt-get install python3-xcffib python3-cairocffi
```

And with those, qtile can be built via a normal `python setup.py install`.

PPA on Launchpad

Packages for old versions are available for 11.10 (Oneiric Ocelot), 12.04 (Precise Pangolin), 12.10 (Quantal Quetzal), 13.04 (Raring Ringtail), 13.10 (Saucy Salamander), 14.04 (Trusty Tahr), and 14.10 (Utopic Unicorn).

```
sudo apt-add-repository ppa:tycho-s/ppa
sudo apt-get update
sudo apt-get install qtile
```

1.1.2 Installing From Source

First, you need to install all of Qtile's dependencies (although some are optional/not needed depending on your Python version, as noted below).

Note that Python 3 versions 3.3 and newer are currently supported and tested. Python 3.2 should still work with Qtile, however the latest versions of pip have dropped support for Qtile, so you will need to either use an older version, or install all the required packages by running the respective `setup.py` scripts.

xcffib

Qtile uses xcffib as an XCB binding, which has its own instructions for building from source. However, if you'd like to skip building it, you can install its dependencies, you will need libxcb and libffi with the associated headers (`libxcb-render0-dev` and `libffi-dev` on Ubuntu), and install it via PyPI:

```
pip install xcffib
```

cairocffi

Qtile uses cairocffi with XCB support via xcffib. You'll need `libcairo2`, the underlying library used by the binding. You should be sure before you install cairocffi that xcffib has been installed, otherwise the needed cairo-xcb bindings will not be built. Once you've got the dependencies installed, you can use the latest version on PyPI:

```
pip install cairocffi
```

pangocairo

You'll also need `libpangocairo`, which on Ubuntu can be installed via `sudo apt-get install libpangocairo-1.0-0`. Qtile uses this to provide text rendering (and binds directly to it via cffi with a small in-tree binding).

asyncio/trollius

Qtile uses the asyncio module as introduced in PEP 3156 for its event loop. Based on your Python version, there are different ways to install this:

- Python >=3.4: The asyncio module comes as part of the standard library, so there is nothing more to install.
- Python 3.3: This has all the infrastructure needed to implement PEP 3156, but the asyncio module must be installed from the Tulip project. This is done by calling:

```
pip install asyncio
```

  Alternatively, you can install trollius (see next point).

- Python 2 and <=3.2 (and 3.3 without asyncio): You will need to install trollius, which backports the asyncio module functionality to work without the infrastructure introduced in PEP 3156. You can install this from PyPI:

```
    pip install trollius
```

dbus/gobject

Until someone comes along and writes an asyncio-based dbus library, qtile will depend on `python-dbus` to interact with dbus. This means that if you want to use things like notification daemon or mpris widgets, you'll need to install python-gobject and python-dbus. Qtile will run fine without these, although it will emit a warning that some things won't work.

Qtile

With the dependencies in place, you can now install qtile:

```
git clone git://github.com/qtile/qtile.git
cd qtile
sudo python setup.py install
```

Stable versions of Qtile can be installed from PyPI:

```
pip install qtile
```

As long as the necessary libraries are in place, this can be done at any point, however, it is recommended that you first install xcffib to ensure the cairo-xcb bindings are built (see above).

1.2 Configuration

Qtile is configured in Python. A script (`~/.config/qtile/config.py` by default) is evaluated, and a small set of configuration variables are pulled from its global namespace.

1.2.1 Configuration lookup order

Qtile looks in the following places for a configuration file, in order:

- The location specified by the `-c` argument.
- `$XDG_CONFIG_HOME/qtile/config.py`, if it is set
- `~/.config/qtile/config.py`
- It reads the module `libqtile.resources.default_config`, included by default with every Qtile installation.

1.2.2 Default Configuration

The default configuration is invoked when qtile cannot find a configuration file. In addition, if qtile is restarted via qsh, qtile will load the default configuration if the config file it finds has some kind of error in it. The documentation below describes the configuration lookup process, as well as what the key bindings are in the default config.

The default config is not intended to be suitable for all users; it's mostly just there so qtile does /something/ when fired up, and so that it doesn't crash and cause you to lose all your work if you reload a bad config.

Key Bindings

The mod key for the default config is `mod4`, which is typically bound to the "Super" keys, which are things like the windows key and the mac command key. The basic operation is:

- `mod + k` or `mod + j`: switch windows on the current stack
- `mod + <space>`: put focus on the other pane of the stack (when in stack layout)
- `mod + <tab>`: switch layouts
- `mod + w`: close window
- `mod + <ctrl> + r`: restart qtile with new config
- `mod + <group name>`: switch to that group
- `mod + <shift> + <group name>`: send a window to that group
- `mod + <enter>`: start xterm
- `mod + r`: start a little prompt in the bar so users can run arbitrary commands

The default config defines one screen and 8 groups, one for each letter in `asdfuiop`. It has a basic bottom bar that includes a group box, the current window name, a little text reminder that you're using the default config, a system tray, and a clock.

The default configuration has several more advanced key combinations, but the above should be enough for basic usage of qtile.

Mouse Bindings

By default, holding your `mod` key and clicking (and holding) a window will allow you to drag it around as a floating window.

1.2.3 Configuration variables

A Qtile configuration consists of a file with a bunch of variables in it, which qtile imports and then runs as a python file to derive its final configuration. The documentation below describes the most common configuration variables; more advanced configuration can be found in the qtile-examples repository, which includes a number of real-world configurations that demonstrate how you can tune Qtile to your liking. (Feel free to issue a pull request to add your own configuration to the mix!)

Lazy objects

The `command.lazy` object is a special helper object to specify a command for later execution. This object acts like the root of the object graph, which means that we can specify a key binding command with the same syntax used to call the command through a script or through qsh.

Example

```
from libqtile.config import Key
from libqtile.command import lazy

keys = [
    Key(
```

```
        ["mod1"], "k",
        lazy.layout.down()
    ),
    Key(
        ["mod1"], "j",
        lazy.layout.up()
    )
]
```

Lazy functions This is overview of the commonly used functions for the key bindings. These functions can be called from commands on the *Qtile* object or on another object in the command tree.

Some examples are given below.

General functions

function	description
`lazy.spawn("application"))`	Run the `application`
`lazy.spawncmd())`	Open command prompt on the bar. See prompt widget.
`lazy.restart())`	Restart Qtile and reload its config. It won't close your windows
`lazy.shutdown())`	Close the whole Qtile

Group functions

function	description
`lazy.next_layout())`	Use next layout on the actual group
`lazy.prev_layout())`	Use previous layout on the actual group
`lazy.screen.next_group())`	Move to the group on the right
`lazy.screen.prev_group())`	Move to the group on the left
`lazy.screen.togglegroup())`	Move to the last visited group
`lazy.group["group_name"].toscreen()`	Move to the group called `group_name`
`lazy.layout.increase_ratio()`	Increase the space for master window at the expense of slave windows
`lazy.layout.decrease_ratio()`	Decrease the space for master window in the advantage of slave windows

Window functions

function	description
`lazy.window.kill())`	Close the focused window
`lazy.layout.next())`	Switch window focus to other pane(s) of stack
`lazy.window.togroup("group_name")`	Move focused window to the group called `group_name`
`lazy.window.toggle_floating()`	Put the focused window to/from floating mode
`lazy.window.toggle_fullscreen()`	Put the focused window to/from fullscreen mode

Groups

A group is a container for a bunch of windows, analogous to workspaces in other window managers. Each client window managed by the window manager belongs to exactly one group. The `groups` config file variable should be

initialized to a list of `DGroup` objects.

`DGroup` objects provide several options for group configuration. Groups can be configured to show and hide themselves when they're not empty, spawn applications for them when they start, automatically acquire certain groups, and various other options.

Example

```
from libqtile.config import Group, Match
groups = [
    Group("a"),
    Group("b"),
    Group("c", matches=[Match(wm_class=["Firefox"])]),
]

# allow mod3+1 through mod3+0 to bind to groups; if you bind your groups
# by hand in your config, you don't need to do this.
from libqtile.dgroups import simple_key_binder
dgroups_key_binder = simple_key_binder("mod3")
```

Reference

Group

class `libqtile.config.`**`Group`**(*name*, *matches=None*, *exclusive=False*, *spawn=None*, *layout=None*, *layouts=None*, *persist=True*, *init=True*, *layout_opts=None*, *screen_affinity=None*, *position=9223372036854775807*)

Represents a "dynamic" group

These groups can spawn apps, only allow certain Matched windows to be on them, hide when they're not in use, etc.

Parameters **name** : string

the name of this group

matches : default `None`

list of `Match` objects whose windows will be assigned to this group

exclusive : boolean

when other apps are started in this group, should we allow them here or not?

spawn : string or list of strings

this will be `exec()` d when the group is created, you can pass either a program name or a list of programs to `exec()`

layout : string

the default layout for this group (e.g. 'max' or 'stack')

layouts : list

the group layouts list overriding global layouts

persist : boolean

should this group stay alive with no member windows?

init : boolean

is this group alive when qtile starts?

position : int

group position

libqtile.dgroups.**simple_key_binder**(*mod*, *keynames=None*)

Bind keys to mod+group position or to the keys specified as second argument

Group Matching

Match

class libqtile.config.**Match**(*title=None*, *wm_class=None*, *role=None*, *wm_type=None*, *wm_instance_class=None*, *net_wm_pid=None*)

Match for dynamic groups

It can match by title, class or role.

`Match` supports both regular expression objects (i.e. the result of `re.compile()`) or strings (match as a "include" match). If a window matches any of the things in any of the lists, it is considered a match.

Parameters **title:**

things to match against the title (WM_NAME)

wm_class:

things to match against the second string in WM_CLASS atom

role:

things to match against the WM_ROLE atom

wm_type:

things to match against the WM_TYPE atom

wm_instance_class:

things to match against the first string in WM_CLASS atom

net_wm_pid:

things to match against the _NET_WM_PID atom (only int allowed in this rule)

Rule

class libqtile.config.**Rule**(*match*, *group=None*, *float=False*, *intrusive=False*, *break_on_match=True*)

How to act on a Match

A Rule contains a Match object, and a specification about what to do when that object is matched.

Parameters **match :**

`Match` object associated with this `Rule`

float :

auto float this window?

intrusive :

override the group's exclusive setting?

break_on_match :

Should we stop applying rules if this rule is matched?

Keys

The `keys` variable defines Qtile's key bindings. Individual key bindings are defined with *libqtile.config.Key* as demonstrated in the following example. Note that you may specify more than one callback functions.

```
from libqtile.config import Key

keys = [
    # Pressing "Meta + Shift + a".
    Key(["mod4", "shift"], "a", callback, ...),

    # Pressing "Control + p".
    Key(["control"], "p", callback, ...),

    # Pressing "Meta + Tab".
    Key(["mod4", "mod1"], "Tab", callback, ...),
]
```

The above may also be written more concisely with the help of the `libqtile.config.EzKey` helper class. The following example is functionally equivalent to the above:

```
from libqtile.config import EzKey as Key

keys = [
    Key("M-S-a", callback, ...),
    Key("C-p",   callback, ...),
    Key("M-A-<Tab>", callback, ...),
]
```

The `EzKey` modifier keys (i.e. `MASC`) can be overwritten through the `EzKey.modifier_keys` dictionary. The defaults are:

```
modifier_keys = {
    'M': 'mod4',
    'A': 'mod1',
    'S': 'shift',
    'C': 'control',
}
```

Modifiers

On most systems `mod1` is the Alt key - you can see which modifiers, which are enclosed in a list, map to which keys on your system by running the `xmodmap` command. This example binds `Alt-k` to the "down" command on the current layout. This command is standard on all the included layouts, and switches to the next window (where "next" is defined differently in different layouts). The matching "up" command switches to the previous window.

Modifiers include: "shift", "lock", "control", "mod1", "mod2", "mod3", "mod4", and "mod5". They can be used in combination by appending more than one modifier to the list:

```
Key(
    ["mod1", "control"], "k",
    lazy.layout.shuffle_down()
)
```

Special keys

These are most commonly used special keys. For complete list please see the code. You can create bindings on them just like for the regular keys. For example `Key(["mod1"], "F4", lazy.window.kill())`.

`Return`
`BackSpace`
`Tab`
`space`
`Home`, `End`
`Left`, `Up`, `Right`, `Down`
`F1`, `F2`, `F3`, ...
`XF86AudioRaiseVolume`
`XF86AudioLowerVolume`
`XF86AudioMute`
`XF86AudioNext`
`XF86AudioPrev`
`XF86MonBrightnessUp`
`XF86MonBrightnessDown`

Reference

Key

class `libqtile.config.`**`Key`**(*modifiers*, *key*, **commands*, ***kwds*)

Defines a keybinding.

Parameters **modifiers:**

A list of modifier specifications. Modifier specifications are one of: “shift”, “lock”, “control”, “mod1”, “mod2”, “mod3”, “mod4”, “mod5”.

key:

A key specification, e.g. “a”, “Tab”, “Return”, “space”.

commands:

A list of lazy command objects generated with the command.lazy helper. If multiple Call objects are specified, they are run in sequence.

kwds:

A dictionary containing “desc”, allowing a description to be added

EzConfig

class `libqtile.config.`**`EzConfig`**

Helper class for defining key and button bindings in an emacs-like format. Inspired by Xmonad’s XMonad.Util.EZConfig.

Layouts

A layout is an algorithm for laying out windows in a group on your screen. Since Qtile is a tiling window manager, this usually means that we try to use space as efficiently as possible, and give the user ample commands that can be bound to keys to interact with layouts.

The `layouts` variable defines the list of layouts you will use with Qtile. The first layout in the list is the default. If you define more than one layout, you will probably also want to define key bindings to let you switch to the next and previous layouts.

See Built-in Layouts for a listing of available layouts.

Example

```
from libqtile import layout
layouts = [
    layout.Max(),
    layout.Stack(stacks=2)
]
```

Mouse

The `mouse` config file variable defines a set of global mouse actions, and is a list of *`Click`* and *`Drag`* objects, which define what to do when a window is clicked or dragged.

Example

```
from libqtile.config import Click, Drag
mouse = [
    Drag([mod], "Button1", lazy.window.set_position_floating(),
        start=lazy.window.get_position()),
    Drag([mod], "Button3", lazy.window.set_size_floating(),
        start=lazy.window.get_size()),
    Click([mod], "Button2", lazy.window.bring_to_front())
]
```

The above example can also be written more concisely with the help of the `EzClick` and `EzDrag` helpers:

```
from libqtile.config import EzClick as EzClick, EzDrag as Drag

mouse = [
    Drag("M-1", lazy.window.set_position_floating(),
        start=lazy.window.get_position()),
    Drag("M-3", lazy.window.set_size_floating(),
        start=lazy.window.get_size()),
    Click("M-2", lazy.window.bring_to_front())
]
```

Reference

Click

class `libqtile.config.`**`Click`**(*modifiers*, *button*, **commands*, ***kwargs*)

Defines binding of a mouse click

It focuses clicked window by default. If you want to prevent it, pass *focus=None* as an argument

Drag
class libqtile.config.**Drag**(*modifiers*, *button*, **commands*, ***kwargs*)
Defines binding of a mouse to some dragging action

On each motion event command is executed with two extra parameters added x and y offset from previous move

It focuses clicked window by default. If you want to prevent it pass, *focus=None* as an argument

Screens

The `screens` configuration variable is where the physical screens, their associated `bars`, and the `widgets` contained within the bars are defined.

See Built-in Widgets for a listing of available widgets.

Example

Tying together screens, bars and widgets, we get something like this:

```
from libqtile.config import Screen
from libqtile import bar, widget

screens = [
    Screen(
        bottom=bar.Bar([
            widget.GroupBox(),
            widget.WindowName()
            ], 30),
        ),
    Screen(
        bottom=bar.Bar([
            widget.GroupBox(),
            widget.WindowName()
            ], 30),
        )
    ]
```

Bars support both solid background colors and gradients by supplying a list of colors that make up a linear gradient. For example, `bar.Bar(..., background="#000000")` will give you a black back ground (the default), while `bar.Bar(..., background=["#000000", "#FFFFFF"])` will give you a background that fades from black to white.

Third-party bars

There might be some reasons to use third-party bars. For instance you can come from another window manager and you have already configured dzen2, xmobar, or something else. They definitely can be used with Qtile too. In fact, any additional configurations aren't needed. Just run the bar and qtile will adapt.

Reference

Screen
class libqtile.config.**Screen**(*top=None*, *bottom=None*, *left=None*, *right=None*, *x=None*, *y=None*, *width=None*, *height=None*)
A physical screen, and its associated paraphernalia.

Define a screen with a given set of Bars of a specific geometry. Note that bar.Bar objects can only be placed at the top or the bottom of the screen (bar.Gap objects can be placed anywhere). Also, `x`, `y`, `width`, and `height` aren't specified usually unless you are using 'fake screens'.

Parameters **top: List of Gap/Bar objects, or None.**

bottom: List of Gap/Bar objects, or None.

left: List of Gap/Bar objects, or None.

right: List of Gap/Bar objects, or None.

x : int or None

y : int or None

width : int or None

height : int or None

Bar

class `libqtile.bar.`**`Bar`**(*widgets*, *size*, ***config*)

A bar, which can contain widgets

Parameters **widgets :**

A list of widget objects.

size :

The "thickness" of the bar, i.e. the height of a horizontal bar, or the width of a vertical bar.

key	default	description
`background`	`'#000000'`	Background colour.
`opacity`	`1`	Bar window opacity.

Gap

class `libqtile.bar.`**`Gap`**(*size*)

A gap placed along one of the edges of the screen

If a gap has been defined, Qtile will avoid covering it with windows. The most probable reason for configuring a gap is to make space for a third-party bar or other static window.

Parameters **size :**

The "thickness" of the gap, i.e. the height of a horizontal gap, or the width of a vertical gap.

Hooks

Qtile provides a mechanism for subscribing to certain events in `libqtile.hook`. To subscribe to a hook in your configuration, simply decorate a function with the hook you wish to subscribe to.

See Built-in Hooks for a listing of available hooks.

Examples

Automatic floating dialogs Let's say we wanted to automatically float all dialog windows (this code is not actually necessary; Qtile floats all dialogs by default). We would subscribe to the `client_new` hook to tell us when a new window has opened and, if the type is "dialog", as can set the window to float. In our configuration file it would look something like this:

```
from libqtile import hook

@hook.subscribe.client_new
def floating_dialogs(window):
    dialog = window.window.get_wm_type() == 'dialog'
    transient = window.window.get_wm_transient_for()
    if dialog or transient:
        window.floating = True
```

A list of available hooks can be found in the Built-in Hooks reference.

Autostart If you want to run commands or spawn some applications when Qtile starts, you'll want to look at the `startup` and `startup_once` hooks. `startup` is emitted every time Qtile starts (including restarts), whereas `startup_once` is only emitted on the very first startup.

Let's create a file `~/.config/qtile/autostart.sh` that will set our desktop wallpaper and start a few programs when Qtile first runs.

```
#!/bin/sh
feh --bg-scale ~/images/wallpaper.jpg &
pidgin &
dropbox start &
```

We can then subscribe to `startup_once` to run this script:

```
import os
import subprocess

@hook.subscribe.startup_once
def autostart():
    home = os.path.expanduser('~/.config/qtile/autostart.sh')
    subprocess.call([home])
```

In addition to the above variables, there are several other boolean configuration variables that control specific aspects of Qtile's behavior:

variable	default	description
follow_mouse_focus	False	Controls whether or not focus follows the mouse around as it moves across windows in a layout.
bring_front_click	False	When clicked, should the window be brought to the front or not. (This sets the X Stack Mode to Above.)
cursor_warp	False	If true, the cursor follows the focus as directed by the keyboard, warping to the center of the focused window.
auto_fullscreen	True	If a window requests to be fullscreen, it is automatically fullscreened. Set this to false if you only want windows to be fullscreen if you ask them to be.

1.2.4 Testing your configuration

The best way to test changes to your configuration is with the provided Xephyr script. This will run Qtile with your `config.py` inside a nested X server and prevent your running instance of Qtile from crashing if something goes wrong.

See Hacking Qtile for more information on using Xephyr.

1.2.5 Starting Qtile

There are several ways to start Qtile. The most common way is via an entry in your X session manager's menu. The default Qtile behavior can be invoked by creating a qtile.desktop file in `/usr/share/xsessions`.

A second way to start Qtile is a custom X session. This way allows you to invoke Qtile with custom arguments, and also allows you to do any setup you want (e.g. special keyboard bindings like mapping caps lock to control, setting your desktop background, etc.) before Qtile starts. If you're using an X session manager, you still may need to create a `custom.desktop` file similar to the `qtile.desktop` file above, but with `Exec=/etc/X11/xsession`. Then, create your own `~/.xsession`. There are several examples of user defined `xsession` s in the qtile-examples repository.

Finally, if you're a gnome user, you can start integrate Qtile into Gnome's session manager and use gnome as usual:

Running Inside Gnome

Add the following snippet to your Qtile configuration. As per this page, it registers Qtile with gnome-session. Without it, a "Something has gone wrong!" message shows up a short while after logging in. dbus-send must be on your $PATH.

```
import subprocess
import os

@hook.subscribe.startup
def dbus_register():
    x = os.environ['DESKTOP_AUTOSTART_ID']
    subprocess.Popen(['dbus-send',
                      '--session',
                      '--print-reply=string',
                      '--dest=org.gnome.SessionManager',
                      '/org/gnome/SessionManager',
                      'org.gnome.SessionManager.RegisterClient',
                      'string:qtile',
                      'string:' + x])
```

This adds a new entry "Qtile GNOME" to GDM's login screen.

```
$ cat /usr/share/xsessions/qtile_gnome.desktop
[Desktop Entry]
Name=Qtile GNOME
Comment=Tiling window manager
TryExec=/usr/bin/gnome-session
Exec=gnome-session --session=qtile
Type=XSession
```

The custom session for gnome-session.

```
$ cat /usr/share/gnome-session/sessions/qtile.session
[GNOME Session]
Name=Qtile session
RequiredComponents=qtile;gnome-settings-daemon;
```

So that Qtile starts automatically on login.

```
$ cat /usr/share/applications/qtile.desktop
[Desktop Entry]
Type=Application
Encoding=UTF-8
Name=Qtile
Exec=qtile
NoDisplay=true
X-GNOME-WMName=Qtile
X-GNOME-Autostart-Phase=WindowManager
X-GNOME-Provides=windowmanager
X-GNOME-Autostart-Notify=false
```

The above does not start gnome-panel. Getting gnome-panel to work requires some extra Qtile configuration, mainly making the top and bottom panels static on panel startup and leaving a gap at the top (and bottom) for the panel window.

You might want to add keybindings to log out of the GNOME session.

```
Key([mod, 'control'], 'l', lazy.spawn('gnome-screensaver-command -l')),
Key([mod, 'control'], 'q', lazy.spawn('gnome-session-quit --logout --no-prompt')),
Key([mod, 'shift', 'control'], 'q', lazy.spawn('gnome-session-quit --power-off')),
```

The above apps need to be in your path (though they are typically installed in `/usr/bin`, so they probably are if they're installed at all).

CHAPTER 2

Commands and scripting

2.1 Commands API

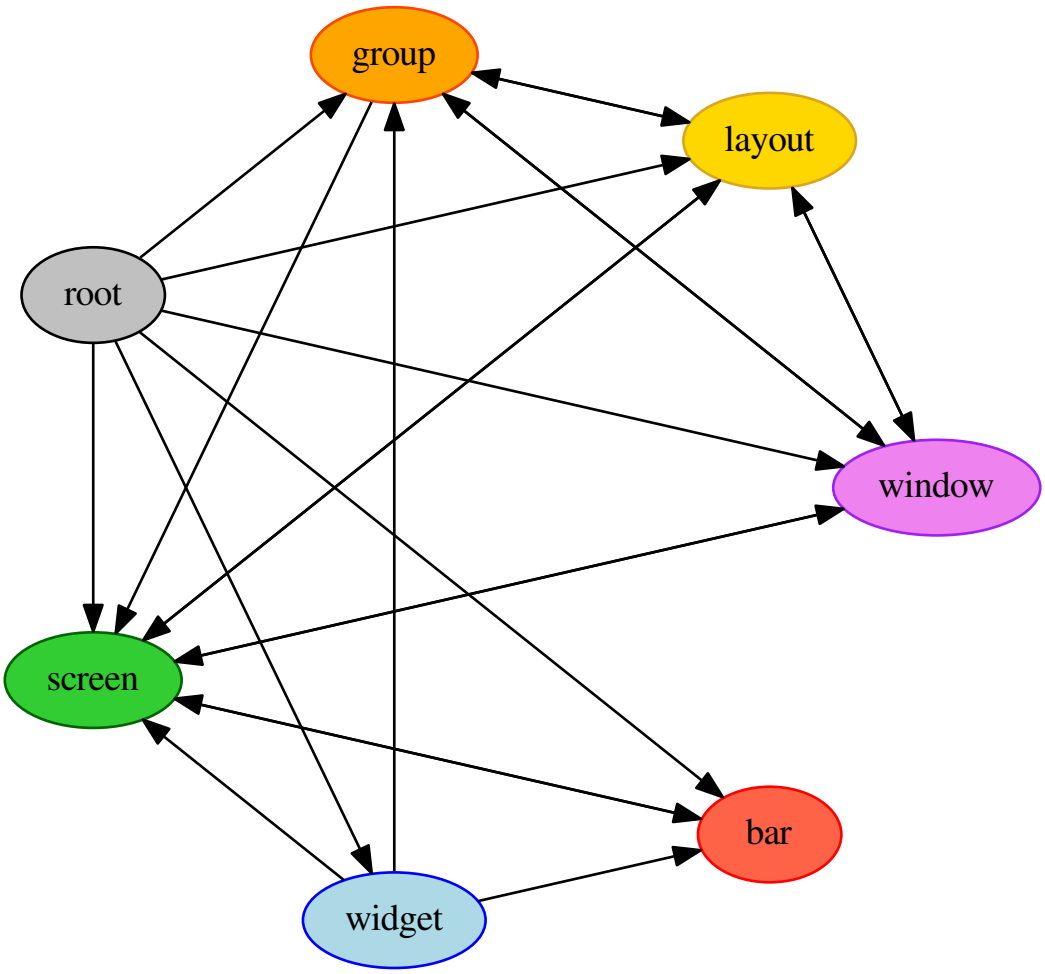
Qtile's command API is based on a graph of objects, where each object has a set of associated commands. The graph and object commands are used in a number of different places:

- Commands can be bound to keys in the Qtile configuration file.
- Commands can be called through qsh, the Qtile shell.
- The qsh can also be hooked into a Jupyter kernel called iqsh.
- Commands can be called from a script to interact with Qtile from Python.

If the explanation below seems a bit complex, please take a moment to explore the API using the `qsh` command shell. Command lists and detailed documentation can be accessed from its built-in help command.

2.1.1 Object Graph

The objects in Qtile's object graph come in seven flavours, matching the seven basic components of the window manager: `layouts`, `windows`, `groups`, `bars`, `widgets`, `screens`, and a special `root` node. Objects are addressed by a path specification that starts at the root, and follows the edges of the graph. This is what the graph looks like:



Each arrow can be read as "holds a reference to". So, we can see that a `widget` object *holds a reference to* objects of type `bar`, `screen` and `group`. Lets start with some simple examples of how the addressing works. Which particular objects we hold reference to depends on the context - for instance, widgets hold a reference to the screen that they appear on, and the bar they are attached to.

Lets look at an example, starting at the root node. The following script runs the `status` command on the root node, which, in this case, is represented by the Client object:

```
from libqtile.command import Client
c = Client()
print c.status()
```

From the graph, we can see that the root node holds a reference to `group` nodes. We can access the "info" command on the current group like so:

```
c.group.info()
```

To access a specific group, regardless of whether or not it is current, we use the Python containment syntax. This

command sends group “b” to screen 1 (by the `libqtile.config.Group.to_screen()` method):

```
c.group["b"].to_screen(1)
```

The current `group`, `layout`, `screen` and `window` can be accessed by simply leaving the key specifier out. The key specifier is mandatory for `widget` and `bar` nodes.

We can now drill down deeper in the graph. To access the screen currently displaying group “b”, we can do this:

```
c.group["b"].screen.info()
```

Be aware, however, that group “b” might not currently be displayed. In that case, it has no associated screen, the path resolves to a non-existent node, and we get an exception:

```
libqtile.command.CommandError: No object screen in path 'group['b'].screen'
```

The graph is not a tree, since it can contain cycles. This path (redundantly) specifies the group belonging to the screen that belongs to group “b”:

```
c.group["b"].screen.group
```

2.1.2 Keys

The key specifier for the various object types are as follows:

Object	Key	Optional?	Example
bar	“top”, “bottom”	No	c.screen.bar[”bottom”]
group	Name string	Yes	c.group[”one”] c.group
layout	Integer index	Yes	c.layout[2] c.layout
screen	Integer index	Yes	c.screen[1] c.screen
widget	Widget name	No	c.widget[”textbox”]
window	Integer window ID	Yes	c.window[123456] c.window

2.2 Scripting

2.2.1 Client-Server Scripting Model

Qtile has a client-server control model - the main Qtile instance listens on a named pipe, over which marshalled command calls and response data is passed. This allows Qtile to be controlled fully from external scripts. Remote interaction occurs through an instance of the *libqtile.command.Client* class. This class establishes a connection to the currently running instance of Qtile, and sources the user's configuration file to figure out which commands should be exposed. Commands then appear as methods with the appropriate signature on the `Client` object. The object hierarchy is described in the Commands API section of this manual. Full command documentation is available through the Qtile Shell.

2.2.2 Example

Below is a very minimal example script that inspects the current qtile instance, and returns the integer offset of the current screen.

```
from libqtile.command import Client
c = Client()
print c.screen.info()["index"]
```

2.2.3 Reference

Client

class `libqtile.command.`**`Client`**(*fname=None*, *is_json=False*)

Exposes a command tree used to communicate with a running instance of Qtile

2.3 qsh

The Qtile command shell is a command-line shell interface that provides access to the full complement of Qtile command functions. The shell features command name completion, and full command documentation can be accessed from the shell itself. The shell uses GNU Readline when it's available, so the interface can be configured to, for example, obey VI keybindings with an appropriate `.inputrc` file. See the GNU Readline documentation for more information.

2.3.1 Navigating the Object Graph

The shell presents a filesystem-like interface to the object graph - the builtin "cd" and "ls" commands act like their familiar shell counterparts:

```
> ls
layout/  widget/  screen/  bar/     window/  group/

> cd bar

bar> ls
bottom/

bar> cd bottom
```

```
bar['bottom']> ls
screen/

bar['bottom']> cd ../..

> ls
layout/  widget/  screen/  bar/     window/  group/
```

Note that the shell provides a "short-hand" for specifying node keys (as opposed to children). The following is a valid shell path:

```
> cd group/4/window/31457314
```

The command prompt will, however, always display the Python node path that should be used in scripts and key bindings:

```
group['4'].window[31457314]>
```

2.3.2 Live Documentation

The shell `help` command provides the canonical documentation for the Qtile API:

```
> cd layout/1

layout[1]> help
help command   -- Help for a specific command.

Builtins
========
cd    exit  help  ls    q     quit

Commands for this object
========================
add           commands      current       delete        doc
down          get info      items         next          previous
rotate        shuffle_down  shuffle_up    toggle_split  up

layout[1]> help previous
previous()
Focus previous stack.
```

2.3.3 Reference

Qsh

class `libqtile.sh.`**`QSh`**(*client*, *completekey='tab'*)

Qtile shell instance

`do_cd`(*arg*)

Change to another path.

Examples

cd layout/0

cd ../layout

do_exit(*args*)
Exit qsh

do_ls(*arg*)
List contained items on a node.

Examples

> ls > ls ../layout

do_pwd(*arg*)
Returns the current working location

This is the same information as presented in the qsh prompt, but is very useful when running iqsh.

Examples

> pwd / > cd bar/top bar['top']> pwd bar['top']

do_help(*arg*)
Give help on commands and builtins

When invoked without arguments, provides an overview of all commands. When passed as an argument, also provides a detailed help on a specific command or builtin.

Examples

> help

> help command

2.4 iqsh

In addition to the standard `qsh` shell interface, we provide a kernel capable of running through Jupyter that hooks into the qsh client. The command structure and syntax is the same as qsh, so it is recommended you read that for more information about that.

2.4.1 Dependencies

In order to run iqsh, you must have ipykernel and jupyter_console. You can install the dependencies when you are installing qtile by running:

```
$ pip install qtile[ipython]
```

Otherwise, you can just install these two packages separately, either through PyPI or through your distribution package manager.

2.4.2 Installing and Running the Kernel

Once you have the required dependencies, you can run the kernel right away by running:

```
$ python -m libqtile.interactive.iqsh_kernel
```

However, this will merely spawn a kernel instance, you will have to run a separate frontend that connects to this kernel.

A more convenient way to run the kernel is by registering the kernel with Jupyter. To register the kernel itself, run:

```
$ python -m libqtile.interactive.iqsh_install
```

If you run this as a non-root user, or pass the `--user` flag, this will install to the user Jupyter kernel directory. You can now invoke the kernel directly when starting a Jupyter frontend, for example:

```
$ jupyter console --kernel qsh
```

The `iqsh` script will launch a Jupyter terminal console with the qsh kernel.

2.4.3 iqsh vs qsh

One of the main drawbacks of running through a Jupyter kernel is the frontend has no way to query the current node of the kernel, and as such, there is no way to set a custom prompt. In order to query your current node, you can call `pwd`.

This, however, enables many of the benefits of running in a Jupyter frontend, including being able to save, run, and re-run code cells in frontends such as the Jupyter notebook.

The Jupyter kernel also enables more advanced help, text completion, and introspection capabilities (however, these are currently not implemented at a level much beyond what is available in the standard qsh).

CHAPTER 3

Getting involved

3.1 Contributing

3.1.1 Reporting bugs

Perhaps the easiest way to contribute to Qtile is to report any bugs you run into on the github issue tracker.

Useful bug reports are ones that get bugs fixed. A useful bug report normally has two qualities:

1. **Reproducible.** If your bug is not reproducible it will never get fixed. You should clearly mention the steps to reproduce the bug. Do not assume or skip any reproducing step. Described the issue, step-by-step, so that it is easy to reproduce and fix.
2. **Specific.** Do not write a essay about the problem. Be Specific and to the point. Try to summarize the problem in minimum words yet in effective way. Do not combine multiple problems even they seem to be similar. Write different reports for each problem.

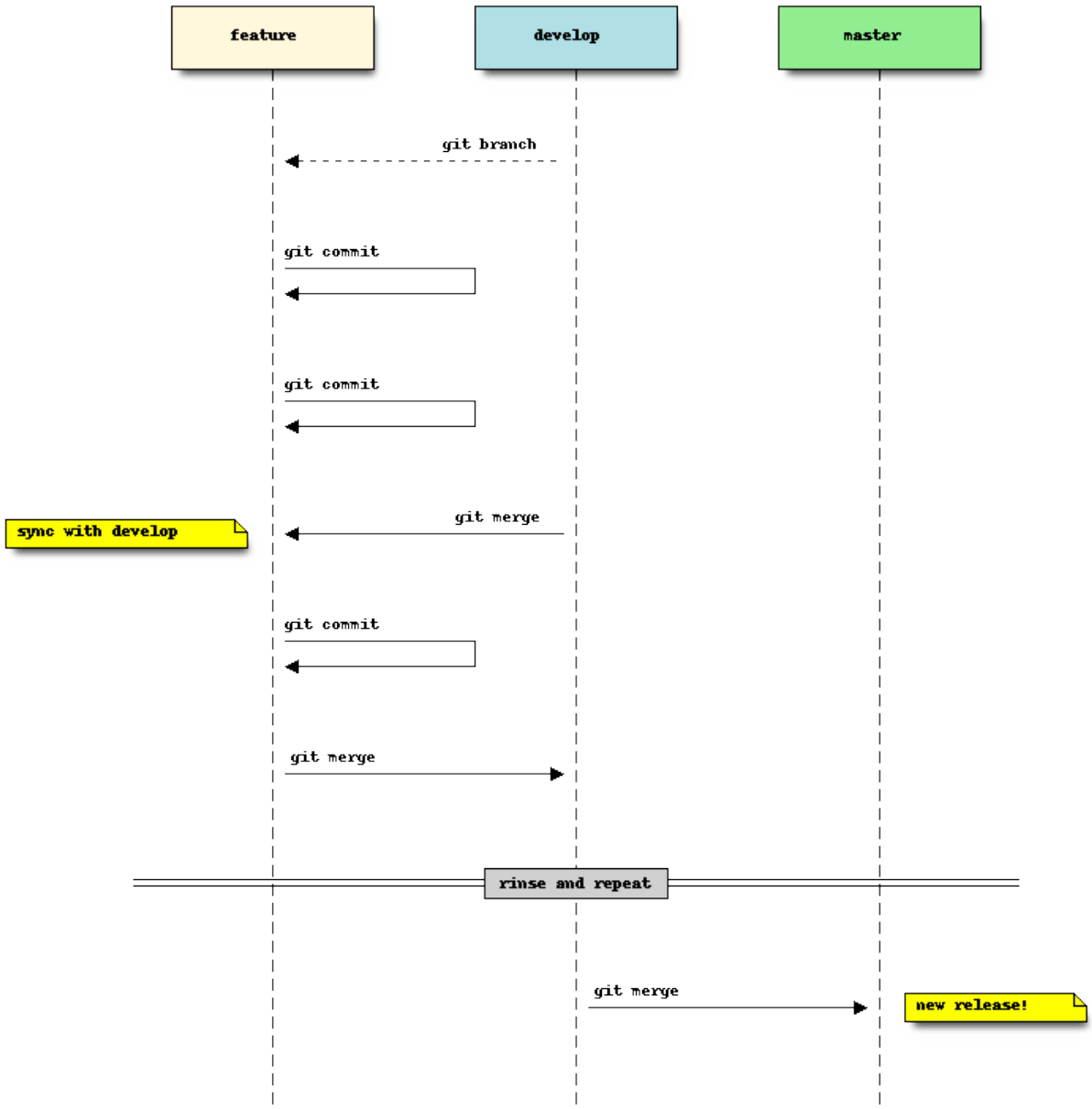
3.1.2 Writing code

To get started writing code for Qtile, check out our guide to Hacking on Qtile.

Git workflow

Our workflow is based on Vincent Driessen's successful git branching model:

- The `master` branch is our current release
- The `develop` branch is what all pull requests should be based against
- Feature branches are where new features, both major and minor, should be developed.



git-flow is a git plugin that helps facilitate this branching strategy. It's not required, but can help make things a bit easier to manage. There is also a good write up on using git-flow.

We also request that git commit messages follow the standard format.

Submit a pull request

You've done your hacking and are ready to submit your patch to Qtile. Great! Now it's time to submit a pull request to our issue tracker on Github.

Important: Pull requests are not considered complete until they include all of the following:

- **Code** that conforms to PEP8.

- **Unit tests** that pass locally and in our CI environment.
- **Documentation** updates on an as needed basis.

Feel free to add your contribution (no matter how small) to the appropriate place in the CHANGELOG as well!

3.2 Hacking on Qtile

3.2.1 Requirements

Any reasonably recent version of these should work, so you can probably just install them from your package manager.

- Nose
- Xephyr
- `xeyes` and `xclock`

On ubuntu, this can be done with `sudo apt-get install python-nose xserver-xephyr x11-apps`.

3.2.2 Building cffi module

Qtile ships with a small in-tree pangocairo binding built using cffi, `pangocffi.py`, and also binds to xcursor with cffi. The bindings are not built at run time and will have to be generated manually when the code is downloaded or when any changes are made to the cffi library. This can be done by calling:

```
python libqtile/ffi_build.py
```

3.2.3 Using Xephyr and the test suite

Qtile has a very extensive test suite, using the Xephyr nested X server. When tests are run, a nested X server with a nested instance of Qtile is fired up, and then tests interact with the Qtile instance through the client API. The fact that we can do this is a great demonstration of just how completely scriptable Qtile is. In fact, Qtile is designed expressly to be scriptable enough to allow unit testing in a nested environment.

The Qtile repo includes a tiny helper script to let you quickly pull up a nested instance of Qtile in Xephyr, using your current configuration. Run it from the top-level of the repository, like this:

```
./scripts/xephyr
```

In practice, the development cycle looks something like this:

1. make minor code change
2. run appropriate test: `nosetests --tests=test_module`
3. GOTO 1, until hackage is complete
4. run entire test suite: `nosetests`
5. commit

3.2.4 Second X Session

Some users prefer to test Qtile in a second, completely separate X session: Just switch to a new tty and run `startx` normally to use the `~/.xinitrc` X startup script.

It's likely though that you want to use a different, customized startup script for testing purposes, for example `~/.config/qtile/xinitrc`. You can do so by launching X with:

```
startx ~/.config/qtile/xinitrc
```

`startx` deals with multiple X sessions automatically. If you want to use `xinit` instead, you need to first copy `/etc/X11/xinit/xserverrc` to `~/.xserverrc`; when launching it, you have to specify a new session number:

```
xinit ~/.config/qtile/xinitrc -- :1
```

Examples of custom X startup scripts are available in qtile-examples.

3.2.5 Capturing an `xtrace`

Occasionally, a bug will be low level enough to require an `xtrace` of Qtile's conversations with the X server. To capture one of these, create an `xinitrc` or similar file with:

```
exec xtrace qtile >> ~/.qtile.log
```

This will put the xtrace output in Qtile's logfile as well. You can then demonstrate the bug, and paste the contents of this file into the bug report.

3.2.6 Coding style

While not all of our code follows PEP8, we do try to adhere to it where possible. All new code should be PEP8 compliant.

The `make lint` command will run a linter with our configuration over libqtile to ensure your patch complies with reasonable formatting constraints. We also request that git commit messages follow the standard format.

3.2.7 Deprecation policy

When a widget API is changed, you should deprecate the change using `libqtile.widget.base.deprecated` to warn users, in addition to adding it to the appropriate place in the changelog. We will typically remove deprecated APIs one tag after they are deprecated.

3.2.8 Testing

Of course, your patches should also pass the unit tests as well (i.e. `make check`). These will be run by travis-ci on every pull request so you can see whether or not your contribution passes.

3.2.9 Resources

Here are a number of resources that may come in handy:

- Inter-Client Conventions Manual

- Extended Window Manager Hints
- A reasonable basic Xlib Manual

CHAPTER 4

Miscellaneous

4.1 Reference

4.1.1 Scripting Commands

Here is documented some of the commands available on objects in the command tree when running qsh or scripting commands to qtile. Note that this is an incomplete list, some objects, such as *layouts* and *widgets*, may implement their own set of commands beyond those given here.

Qtile

class libqtile.manager.**Qtile**(*config*, *displayName=None*, *fname=None*, *no_spawn=False*, *state=None*)

This object is the *root* of the command graph

cmd_add_rule(*match_args*, *rule_args*, *min_priorty=False*)

Add a dgroup rule, returns rule_id needed to remove it

Parameters match_args :

config.Match arguments

rule_args :

config.Rule arguments

min_priorty :

If the rule is added with minimum prioriry (last) (default: False)

cmd_addgroup(*group*)

Add a group with the given name

cmd_commands()

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

cmd_critical()

Set log level to CRITICAL

cmd_debug()

Set log level to DEBUG

cmd_delgroup(*group*)
Delete a group with the given name

cmd_display_kb(**args*)
Display table of key bindings

cmd_doc(*name*)
Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

cmd_error()
Set log level to ERROR

cmd_eval(*code*)
Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

cmd_execute(*cmd*, *args*)
Executes the specified command, replacing the current process

cmd_findwindow(*prompt='window'*, *widget='prompt'*)
Launch prompt widget to find a window of the given name

Parameters **prompt :**

Text with which to prompt user (default: "window")

widget :

Name of the prompt widget (default: "prompt")

cmd_focus_by_click(*e*)
Bring a window to the front

Parameters **e** : xcb event

Click event used to determine window to focus

cmd_function(*function*, **args*, ***kwargs*)
Call a function with current object as argument

cmd_get_info()
Prints info for all groups

cmd_get_state()
Get pickled state for restarting qtile

cmd_groups()
Return a dictionary containing information for all groups

Examples

groups()

cmd_hide_show_bar(*position='all'*)
Toggle visibility of a given bar

Parameters **position :**

one of: "top", "bottom", "left", "right", or "all" (default: "all")

cmd_info()
Set log level to INFO

cmd_internal_windows()
Return info for each internal window (bars, for example)

cmd_items(*name*)
Returns a list of contained items for the specified name

Used by __qsh__ to allow navigation of the object graph.

cmd_list_widgets()
List of all addressible widget names

cmd_next_layout(*group=None*)
Switch to the next layout.

Parameters group :

Group name. If not specified, the current group is assumed

cmd_next_screen()
Move to next screen

cmd_next_urgent()
Focus next window with urgent hint

cmd_pause()
Drops into pdb

cmd_prev_layout(*group=None*)
Switch to the previous layout.

Parameters group :

Group name. If not specified, the current group is assumed

cmd_prev_screen()
Move to the previous screen

cmd_qtile_info()
Returns a dictionary of info on the Qtile instance

cmd_qtilecmd(*prompt='command'*, *widget='prompt'*, *messenger='xmessage'*)
Execute a Qtile command using the client syntax

Tab completeion aids navigation of the command tree

Parameters prompt :

Text to display at the prompt (default: "command: ")

widget :

Name of the prompt widget (default: "prompt")

messenger :

Command to display output, set this to None to disable (default: "xmessage")

cmd_remove_rule(*rule_id*)
Remove a dgroup rule by rule_id

cmd_restart()
Restart qtile using the execute command

cmd_run_external(*full_path*)
Run external Python script

cmd_screens()
Return a list of dictionaries providing information on all screens

cmd_shutdown()
Quit Qtile

cmd_simulate_keypress(*modifiers*, *key*)
Simulates a keypress on the focused window.

Parameters modifiers :

A list of modifier specification strings. Modifiers can be one of "shift", "lock", "control" and "mod1" - "mod5".

key :

Key specification.

Examples

simulate_keypress(["control", "mod2"], "k")

cmd_spawn(*cmd*)
Run cmd in a shell.

cmd may be a string, which is parsed by shlex.split, or a list (similar to subprocess.Popen).

Examples

spawn("firefox")

spawn(["xterm", "-T", "Temporary terminal"])

cmd_spawncmd(*prompt='spawn'*, *widget='prompt'*, *command='%s'*, *complete='cmd'*)
Spawn a command using a prompt widget, with tab-completion.

Parameters prompt :

Text with which to prompt user (default: "spawn: ").

widget :

Name of the prompt widget (default: "prompt").

command :

command template (default: "%s").

complete :

Tab completion function (default: "cmd")

cmd_status()
Return "OK" if Qtile is running

cmd_switch_groups(*groupa*, *groupb*)
Switch position of groupa to groupb

cmd_switchgroup(*prompt='group'*, *widget='prompt'*)
Launch prompt widget to switch to a given group to the current screen

Parameters prompt :

Text with which to prompt user (default: "group")

widget :

Name of the prompt widget (default: "prompt")

cmd_sync()
Sync the X display. Should only be used for development

cmd_to_layout_index(*index*, *group=None*)
Switch to the layout with the given index in self.layouts.

Parameters index :

Index of the layout in the list of layouts.

group :

Group name. If not specified, the current group is assumed.

cmd_to_screen(*n*)
Warp focus to screen n, where n is a 0-based screen number

Examples

to_screen(0)

cmd_togroup(*prompt='group'*, *widget='prompt'*)
Launch prompt widget to move current window to a given group

Parameters prompt :

Text with which to prompt user (default: "group")

widget :

Name of the prompt widget (default: "prompt")

cmd_tracemalloc_dump()
Dump tracemalloc snapshot

cmd_tracemalloc_toggle()
Toggle tracemalloc status

Running tracemalloc is required for qtile-top

cmd_warning()
Set log level to WARNING

cmd_windows()
Return info for each client window

Bar

class libqtile.bar.**Bar**(*widgets*, *size*, ***config*)
A bar, which can contain widgets

Parameters widgets :

A list of widget objects.

size :

The "thickness" of the bar, i.e. the height of a horizontal bar, or the width of a vertical bar.

key	default	description
`background`	`'#000000'`	Background colour.
`opacity`	`1`	Bar window opacity.

cmd_commands()
Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

cmd_doc(*name*)
Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

cmd_eval(*code*)
Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

cmd_fake_button_press(*screen*, *position*, *x*, *y*, *button=1*)
Fake a mouse-button-press on the bar. Co-ordinates are relative to the top-left corner of the bar.

:screen The integer screen offset :position One of "top", "bottom", "left", or "right"

cmd_function(*function*, **args*, ***kwargs*)
Call a function with current object as argument

cmd_info()
Info for this object.

cmd_items(*name*)
Returns a list of contained items for the specified name

Used by __qsh__ to allow navigation of the object graph.

Group

class libqtile.config.**Group**(*name*, *matches=None*, *exclusive=False*, *spawn=None*, *layout=None*, *layouts=None*, *persist=True*, *init=True*, *layout_opts=None*, *screen_affinity=None*, *position=9223372036854775807*)
Represents a "dynamic" group

These groups can spawn apps, only allow certain Matched windows to be on them, hide when they're not in use, etc.

Parameters **name** : string

the name of this group

matches : default `None`

list of `Match` objects whose windows will be assigned to this group

exclusive : boolean

when other apps are started in this group, should we allow them here or not?

spawn : string or list of strings

this will be `exec()` d when the group is created, you can pass either a program name or a list of programs to `exec()`

layout : string

the default layout for this group (e.g. 'max' or 'stack')

layouts : list

the group layouts list overriding global layouts

persist : boolean

should this group stay alive with no member windows?

init : boolean

is this group alive when qtile starts?

position : int

group position

Screen

class libqtile.config.**Screen**(*top=None*, *bottom=None*, *left=None*, *right=None*, *x=None*, *y=None*, *width=None*, *height=None*)

A physical screen, and its associated paraphernalia.

Define a screen with a given set of Bars of a specific geometry. Note that bar.Bar objects can only be placed at the top or the bottom of the screen (bar.Gap objects can be placed anywhere). Also, `x`, `y`, `width`, and `height` aren't specified usually unless you are using 'fake screens'.

Parameters **top: List of Gap/Bar objects, or None.**

bottom: List of Gap/Bar objects, or None.

left: List of Gap/Bar objects, or None.

right: List of Gap/Bar objects, or None.

x : int or None

y : int or None

width : int or None

height : int or None

cmd_commands()

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

cmd_doc(*name*)

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

cmd_eval(*code*)

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

cmd_function(*function*, **args*, ***kwargs*)

Call a function with current object as argument

cmd_info()
Returns a dictionary of info for this screen.

cmd_items(*name*)
Returns a list of contained items for the specified name

Used by __qsh__ to allow navigation of the object graph.

cmd_next_group(*skip_empty=False*, *skip_managed=False*)
Switch to the next group

cmd_prev_group(*skip_empty=False*, *skip_managed=False*)
Switch to the previous group

cmd_resize(*x=None*, *y=None*, *w=None*, *h=None*)
Resize the screen

cmd_togglegroup(*groupName=None*)
Switch to the selected group or to the previously active one

Window

class libqtile.window.**Window**(*window*, *qtile*)

cmd_bring_to_front()

cmd_commands()
Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

cmd_disable_floating()

cmd_disable_fullscreen()

cmd_disable_maximize()

cmd_disable_minimize()

cmd_doc(*name*)
Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

cmd_down_opacity()

cmd_enable_floating()

cmd_enable_fullscreen()

cmd_enable_maximize()

cmd_enable_minimize()

cmd_eval(*code*)
Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

cmd_function(*function*, **args*, ***kwargs*)
Call a function with current object as argument

cmd_get_position()

cmd_get_size()

cmd_info()
Returns a dictionary of info for this object

cmd_inspect()
Tells you more than you ever wanted to know about a window

cmd_items(*name*)
Returns a list of contained items for the specified name

Used by __qsh__ to allow navigation of the object graph.

cmd_kill()
Kill this window

Try to do this politely if the client support this, otherwise be brutal.

cmd_match(**args*, ***kwargs*)

cmd_move_floating(*dx*, *dy*, *curx*, *cury*)
Move window by dx and dy

cmd_opacity(*opacity*)

cmd_resize_floating(*dw*, *dh*, *curx*, *cury*)
Add dw and dh to size of window

cmd_set_position(*dx*, *dy*, *curx*, *cury*)

cmd_set_position_floating(*x*, *y*, *curx*, *cury*)
Move window to x and y

cmd_set_size_floating(*w*, *h*, *curx*, *cury*)
Set window dimensions to w and h

cmd_static(*screen*, *x*, *y*, *width*, *height*)

cmd_toggle_floating()

cmd_toggle_fullscreen()

cmd_toggle_maximize()

cmd_toggle_minimize()

cmd_togroup(*groupName=None*)
Move window to a specified group.

If groupName is not specified, we assume the current group

Examples

Move window to current group:

```
togroup()
```

Move window to group “a”:

```
togroup("a")
```

cmd_up_opacity()

4.1.2 Built-in Hooks

subscribe.**selection_notify** (*func*)
Called on selection notify

subscribe.**startup_once** (*func*)
Called when Qtile has initialized, exactly once (i.e. not on each lazy.restart()).

subscribe.**client_killed** (*func*)
Called after a client has been unmanaged.

Arguments:

window.Window object of the killed window.

subscribe.**float_change** (*func*)
Called when a change in float state is made

subscribe.**setgroup** (*func*)
Called when group is changed

subscribe.**delgroup** (*func*)
Called when group is deleted

subscribe.**group_window_add** (*func*)
Called when a new window is added to a group

subscribe.**changegroup** (*func*)
Called whenever a group change occurs

subscribe.**addgroup** (*func*)
Called when group is added

subscribe.**net_wm_icon_change** (*func*)
Called on _NET_WM_ICON chance

subscribe.**selection_change** (*func*)
Called on selection chance

subscribe.**layout_change** (*func*)
Called on layout change

subscribe.**current_screen_change** (*func*)
Called when the current screen (i.e. the screen with focus) changes

No arguments.

subscribe.**client_mouse_enter** (*func*)
Called when the mouse enters a client

subscribe.**client_urgent_hint_changed** (*func*)
Called when the client urgent hint changes

subscribe.**client_new** (*func*)
Called before Qtile starts managing a new client

Use this hook to declare windows static, or add them to a group on startup. This hook is not called for internal windows.

Arguments:

window.Window object

Examples

```
    def func(c):
        if c.name == "xterm":
            c.togroup("a")
        elif c.name == "dzen":
            c.static(0)

    libqtile.hook.subscribe.client_new(func)
```

subscribe.**client_managed**(*func*)
Called after Qtile starts managing a new client

That is, after a window is assigned to a group, or when a window is made static. This hook is not called for internal windows.

Arguments:

window.Window object

subscribe.**client_name_updated**(*func*)
Called when the client name changes

subscribe.**startup**(*func*)
Called each time qtile is started (including the first time qtile starts)

subscribe.**client_focus**(*func*)
Called whenever focus changes

Arguments: window.Window object of the new focus.

subscribe.**focus_change**(*func*)
Called when focus is changed

subscribe.**client_type_changed**(*func*)
Called whenever window type changes

subscribe.**screen_change**(*func*)
Called when a screen is added or screen configuration is changed (via xrandr)

The hook should take two arguments: the root qtile object and the `xproto.randr.ScreenChangeNotify` event. Common usage is simply to call `qtile.cmd_restart()` on each event (to restart qtile when there is a new monitor):

Example:

```
    def restart_on_randr(qtile, ev):
        qtile.cmd_restart()
```

subscribe.**client_state_changed**(*func*)
Called whenever client state changes

subscribe.**window_name_change**(*func*)
Called whenever a windows name changes

4.1.3 Built-in Layouts

Floating

class libqtile.layout.floating.**Floating**(*float_rules=None*, ***config*)
Floating layout, which does nothing with windows but handles focus order

key	default	description
border_focus	'#0000ff'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	1	Border width.
max_border_width	0	Border width for maximize.
fullscreen_border_width	0	Border width for fullscreen.
name	'floating'	Name of this layout.
auto_float_types	{'utility', 'notification', 'splash', 'dialog', 'toolbar'}	default wm types to automatically float

Columns

class libqtile.layout.columns.**Columns**(***config*)
Extension of the Stack layout.

The screen is split into columns, which can be dynamically added or removed. Each column displays either a sigle window at a time from a stack of windows or all of them simultaneously, spliting the column space. Columns and windows can be resized and windows can be shuffled around. This layout can also emulate "Wmii", "Verical", and "Max", depending on the default parameters.

An example key configuration is:

```
    Key([mod], "j", lazy.layout.down()),
    Key([mod], "k", lazy.layout.up()),
    Key([mod], "h", lazy.layout.left()),
    Key([mod], "l", lazy.layout.right()),
    Key([mod, "shift"], "j", lazy.layout.shuffle_down()),
    Key([mod, "shift"], "k", lazy.layout.shuffle_up()),
    Key([mod, "shift"], "h", lazy.layout.shuffle_left()),
    Key([mod, "shift"], "l", lazy.layout.shuffle_right()),
    Key([mod, "control"], "j", lazy.layout.grow_down()),
    Key([mod, "control"], "k", lazy.layout.grow_up()),
    Key([mod, "control"], "h", lazy.layout.grow_left()),
    Key([mod, "control"], "l", lazy.layout.grow_right()),
    Key([mod], "Return", lazy.layout.toggle_split()),
    Key([mod], "n", lazy.layout.normalize()),
```

key	default	description
name	'columns'	Name of this layout.
border_focus	'#881111'	Border colour for the focused window.
border_normal	'#220000'	Border colour for un-focused windows.
border_width	2	Border width.
margin	0	Margin of the layout.
autosplit	True	Autosplit newly created columns.
num_columns	2	Preferred number of columns.
grow_amount	10	Amount by which to grow a window/column.
fair	False	Add new windows to the column with least windows.

Matrix

class libqtile.layout.matrix.**Matrix**(*columns=2*, ***config*)

This layout divides the screen into a matrix of equally sized cells and places one window in each cell. The number of columns is configurable and can also be changed interactively.

key	default	description
border_focus	'#0000ff'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	1	Border width.
name	'matrix'	Name of this layout.
margin	0	Margin of the layout

Max

class libqtile.layout.max.**Max**(***config*)

Maximized layout

A simple layout that only displays one window at a time, filling the screen. This is suitable for use on laptops and other devices with small screens. Conceptually, the windows are managed as a stack, with commands to switch to next and previous windows in the stack.

key	default	description
name	'max'	Name of this layout.

MonadTall

class libqtile.layout.xmonad.**MonadTall**(***config*)

Emulate the behavior of XMonad's default tiling scheme

Main-Pane:

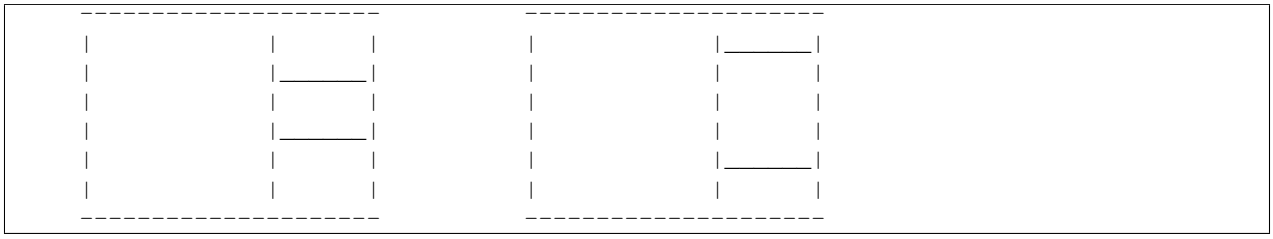
A main pane that contains a single window takes up a vertical portion of the screen based on the ratio setting. This ratio can be adjusted with the cmd_grow_main and cmd_shrink_main or, while the main pane is in focus, cmd_grow and cmd_shrink.

```
---------------------
|             |       |
|             |       |
|             |       |
|             |       |
|             |       |
|             |       |
---------------------
```

Using the `cmd_flip` method will switch which horizontal side the main pane will occupy. The main pane is considered the "top" of the stack.

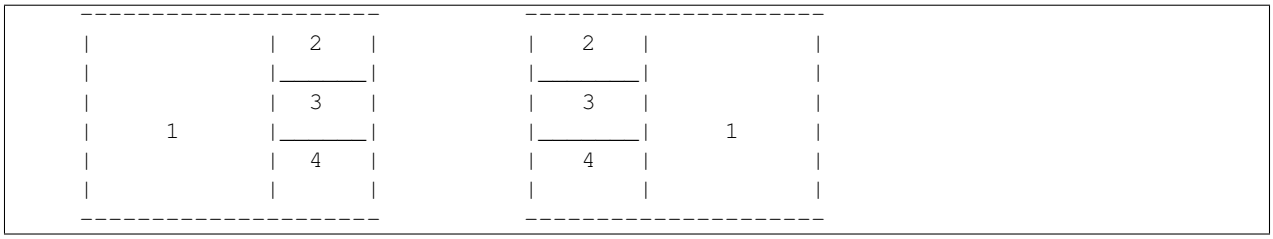
Secondary-panes:

Occupying the rest of the screen are one or more secondary panes. The secondary panes will share the vertical space of the screen however they can be resized at will with the `cmd_grow` and `cmd_shrink` methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.



Panes can be moved with the `cmd_shuffle_up` and `cmd_shuffle_down` methods. As mentioned the main pane is considered the top of the stack; moving up is counter-clockwise and moving down is clockwise.

The opposite is true if the layout is "flipped".



Normalizing:

To restore all client windows to their default size ratios simply use the `cmd_normalize` method.

Maximizing:

To toggle a client window between its minimum and maximum sizes simply use the `cmd_maximize` on a focused client.

Suggested Bindings:

```
Key([modkey], "h", lazy.layout.left()),
Key([modkey], "l", lazy.layout.right()),
Key([modkey], "j", lazy.layout.down()),
Key([modkey], "k", lazy.layout.up()),
Key([modkey, "shift"], "h", lazy.layout.swap_left()),
Key([modkey, "shift"], "l", lazy.layout.swap_right()),
Key([modkey, "shift"], "j", lazy.layout.shuffle_down()),
Key([modkey, "shift"], "k", lazy.layout.shuffle_up()),
Key([modkey], "i", lazy.layout.grow()),
Key([modkey], "m", lazy.layout.shrink()),
Key([modkey], "n", lazy.layout.normalize()),
```

```
    Key([modkey], "o", lazy.layout.maximize()),
    Key([modkey, "shift"], "space", lazy.layout.flip()),
```

key	default	description
border_focus	'#ff0000'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	2	Border width.
single_border_width	None	Border width for single window
name	'xmonad-tall'	Name of this layout.
margin	0	Margin of the layout
ratio	0.5	The percent of the screen-space the master pane should occupy by default.
align	0	Which side master plane will be placed (one of MonadTall._left or MonadTall._right)
change_ratio	0.05	Resize ratio
change_size	20	Resize change in pixels

RatioTile

class libqtile.layout.ratiotile.**RatioTile**(***config*)

Tries to tile all windows in the width/height ratio passed in

key	default	description
border_focus	'#0000ff'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	1	Border width.
name	'ratiotile'	Name of this layout.
margin	0	Margin of the layout
ratio	1.618	Ratio of the tiles
ratio_increment	0.1	Amount to increment per ratio increment
fancy	False	Use a different method to calculate window sizes.

Slice

class libqtile.layout.slice.**Slice**(***config*)

Slice layout

This layout cuts piece of screen and places a single window on that piece, and delegates other window placement to other layout

key	default	description
width	256	Slice width
side	'left'	Side of the slice (left, right, top, bottom)
name	'max'	Name of this layout.
wname	None	WM_NAME to match
wmclass	None	WM_CLASS to match
role	None	WM_WINDOW_ROLE to match
fallback	<libqtile.layout.max.Max object at 0x7f72a0860dd8>	Fallback layout

Stack

class libqtile.layout.stack.**Stack**(***config*)
A layout composed of stacks of windows

The stack layout divides the screen horizontally into a set of stacks. Commands allow you to switch between stacks, to next and previous windows within a stack, and to split a stack to show all windows in the stack, or unsplit it to show only the current window. At the moment, this is the most mature and flexible layout in Qtile.

key	default	description
border_focus	'#0000ff'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	1	Border width.
name	'stack'	Name of this layout.
autosplit	False	Auto split all new stacks.
num_stacks	2	Number of stacks.
fair	False	Add new windows to the stacks in a round robin way.
margin	0	Margin of the layout

Tile

class libqtile.layout.tile.**Tile**(*ratio=0.618*, *masterWindows=1*, *expand=True*, *ratio_increment=0.05*, *add_on_top=True*, *shift_windows=False*, *master_match=None*, ***config*)

key	default	description
border_focus	'#0000ff'	Border colour for the focused window.
border_normal	'#000000'	Border colour for un-focused windows.
border_width	1	Border width.
name	'tile'	Name of this layout.
margin	0	Margin of the layout

TreeTab

class libqtile.layout.tree.**TreeTab**(***config*)
Tree Tab Layout

This layout works just like Max but displays tree of the windows at the left border of the screen, which allows you to overview all opened windows. It's designed to work with uzbl-browser but works with other windows too.

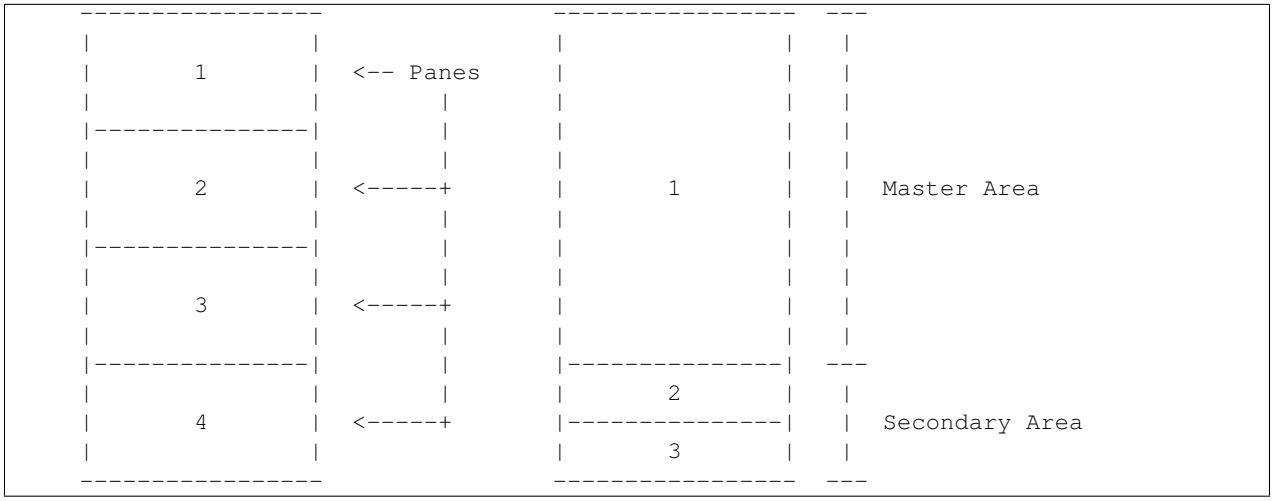
key	default	description
bg_color	'000000'	Background color of tabs
active_bg	'000080'	Background color of active tab
active_fg	'ffffff'	Foreground color of active tab
inactive_bg	'606060'	Background color of inactive tab
inactive_fg	'ffffff'	Foreground color of inactive tab
margin_left	6	Left margin of tab panel
margin_y	6	Vertical margin of tab panel
padding_left	6	Left padding for tabs
padding_x	6	Left padding for tab label
padding_y	2	Top padding for tab label
border_width	2	Width of the border
vspace	2	Space between tabs
level_shift	8	Shift for children tabs
font	'Arial'	Font
fontsize	14	Font pixel size.
fontshadow	None	font shadow color, default is None (no shadow)
section_fontsize	11	Font pixel size of section label
section_fg	'ffffff'	Color of section label
section_top	4	Top margin of section label
section_bottom	6	Bottom margin of section
section_padding	4	Bottom of margin section label
section_left	4	Left margin of section label
panel_width	150	Width of the left panel
sections	['Default']	Foreground color of inactive tab
name	'treetab'	Name of this layout.
previous_on_rm	False	Focus previous window on close instead of first.

VerticalTile

class libqtile.layout.verticaltile.**VerticalTile**(**_config_)

Tiling layout that works nice on vertically mounted monitors

The available height gets divided by the number of panes, if no pane is maximized. If one pane has been maximized, the available height gets split in master- and secondary area. The maximized pane (master pane) gets the full height of the master area and the other panes (secondary panes) share the remaining space. The master area (at default 75%) can grow and shrink via keybindings.



Normal behavior. No One maximized pane in the master area maximized pane. No and two secondary panes in the specific areas. secondary area.

```
-----------------------------------  In some cases VerticalTile can be
|                                 |  useful on horizontal mounted
|                1                |  monitors two.
|                                 |  For example if you want to have a
|---------------------------------|  webbrowser and a shell below it.
|                                 |
|                2                |
|                                 |
-----------------------------------
```

Suggested keybindings:

```
Key([modkey], 'j', lazy.layout.down()),
Key([modkey], 'k', lazy.layout.up()),
Key([modkey], 'Tab', lazy.layout.next()),
Key([modkey, 'shift'], 'Tab', lazy.layout.next()),
Key([modkey, 'shift'], 'j', lazy.layout.shuffle_down()),
Key([modkey, 'shift'], 'k', lazy.layout.shuffle_up()),
Key([modkey], 'm', lazy.layout.maximize()),
Key([modkey], 'n', lazy.layout.normalize()),
```

key	default	description
border_focus	'#FF0000'	Border color for the focused window.
border_normal	'#FFFFFF'	Border color for un-focused windows.
border_width	1	Border width.
margin	0	Border margin.
name	'VerticalTile'	Name of this layout.

Wmii

class libqtile.layout.wmii.**Wmii**(***config*)

This layout emulates wmii layouts

The screen it split into columns, always starting with one. A new window is created in the active window's column. Windows can be shifted left and right. If there is no column when shifting, a new one is created. Each column can be stacked or divided (equally split).

This layout implements something akin to wmii's semantics.

Each group starts with one column. The first window takes up the whole screen. Next window splits the column in half. Windows can be moved to the column to the left or right. If there is no column in the direction being moved into, a new column is created.

Each column can be either stacked (each window takes up the whole vertical real estate) or split (the windows are split equally vertically in the column) Columns can be grown horizontally (cmd_grow_left/right).

My config.py has the following added:

```
Key(
    [mod, "shift", "control"], "l",
    lazy.layout.grow_right()
),
Key(
    [mod, "shift"], "l",
    lazy.layout.shuffle_right()
),
```

```
    Key(
        [mod, "shift", "control"], "h",
        lazy.layout.grow_left()
    ),
    Key(
        [mod, "shift"], "h",
        lazy.layout.shuffle_left()
    ),
    Key(
        [mod], "s",
        lazy.layout.toggle_split()
    ),
```

key	default	description
border_focus	'#881111'	Border colour for the focused window.
border_normal	'#220000'	Border colour for un-focused windows.
border_focus_stack	'#0000ff'	Border colour for un-focused windows.
border_normal_stack	'#000022'	Border colour for un-focused windows.
grow_amount	5	Amount by which to grow/shrink a window.
border_width	2	Border width.
name	'wmii'	Name of this layout.
margin	0	Margin of the layout

Zoomy

class libqtile.layout.zoomy.**Zoomy**(***config*)

A layout with single active windows, and few other previews at the right

key	default	description
columnwidth	150	Width of the right column
property_name	'ZOOM'	Property to set on zoomed window
property_small	'0.1'	Property value to set on zoomed window
property_big	'1.0'	Property value to set on normal window
margin	0	Margin of the layout

4.1.4 Built-in Widgets

AGroupBox

class libqtile.widget.**AGroupBox**(***config*)

A widget that graphically displays the current group

Supported bar orientations: horizontal only

key	default	description
border	'000000'	group box border color

Backlight

class libqtile.widget.**Backlight**(***config*)

A simple widget to show the current brightness of a monitor

Supported bar orientations: horizontal only

key	default	description
backlight_name	'acpi_video0'	ACPI name of a backlight device
brightness_file	'brightness'	Name of file with the current brightness in /sys/class/backlight/backlight_name
max_brightness_file	'max_brightness'	Name of file with the maximum brightness in /sys/class/backlight/backlight_name
update_interval	0.2	The delay in seconds between updates

Battery

class libqtile.widget.**Battery**(***config*)
A simple but flexible text-based battery widget

Supported bar orientations: horizontal only

key	default	description
charge_char	'^'	Character to indicate the battery is charging
discharge_char	'V'	Character to indicate the battery is discharging
error_message	'Error'	Error message if something is wrong
format	'{char} {percent:2.0%} {hour:d}:{min:02d}'	Display format
hide_threshold	None	Hide the text when there is enough energy
low_percentage	0.1	Indicates when to use the low_foreground color 0 < x < 1
low_foreground	'FF0000'	Font color on low battery

BatteryIcon

class libqtile.widget.**BatteryIcon**(***config*)
Battery life indicator widget.

Supported bar orientations: horizontal only

key	default	description
theme_path	'/home/docs/checkouts/readthedocs.org/user_builds/qtile/checkouts/v0.10.5/libq	Path of the icons
custom_icons	{}	dict containing key->filename icon map

BitcoinTicker

class libqtile.widget.**BitcoinTicker**(***config*)
A bitcoin ticker widget, data provided by the btc-e.com API. Defaults to displaying currency in whatever the current locale is. Examples:

```
    # display the average price of bitcoin in local currency
    widget.BitcoinTicker(format="BTC: {avg}")

    # display the average price of litecoin in local currency
    widget.BitcoinTicker(format="LTC: {avg}", source_currency='ltc')
```

```
    # display the average price of litecoin in bitcoin
    widget.BitcoinTicker(format="BTC: Ƀ{avg}", source_currency='ltc', currency='btc', round=False)
```

Supported bar orientations: horizontal only

key	default	description
currency	''	The currency the value that bitcoin is displayed in
source_currency	'btc'	The source currency to convert from
round	True	whether or not to use locale.currency to round the values
format	'BTC Buy: {buy}, Sell: {sell}'	Display format, allows buy, sell, high, low, avg, vol, vol_cur, last, variables.

CPUGraph

class libqtile.widget.**CPUGraph**(***config*)
Display CPU usage graph

Supported bar orientations: horizontal only

key	default	description
core	'all'	Which core to show (all/0/1/2/...)

Canto

class libqtile.widget.**Canto**(***config*)
Display RSS feeds updates using the canto console reader

Supported bar orientations: horizontal only

key	default	description
fetch	False	Whether to fetch new items on update
feeds	[]	List of feeds to display, empty for all
one_format	'{name}: {number}'	One feed display format
all_format	'{number}'	All feeds display format

CheckUpdates

class libqtile.widget.**CheckUpdates**(***config*)
Shows number of pending updates in different unix systems

Supported bar orientations: horizontal only

key	default	description
distro	'Arch'	Name of your distribution
update_interval	60	Update interval in seconds.
execute	None	Command to execute on click
display_format	'Updates: {updates}'	Display format if updates available
colour_no_updates	'ffffff'	Colour when there's no updates.
colour_have_updates	'ffffff'	Colour when there are updates.

Clipboard

class libqtile.widget.**Clipboard**(*width=CALCULATED*, ***config*)
Display current clipboard contents

Supported bar orientations: horizontal only

key	default	description
selection	'CLIPBOARD'	Which selection to display(CLIPBOARD or PRIMARY)
max_width	10	maximum number of characters to display (None for all, useful when width is bar.STRETCH)
timeout	10	Default timeout (seconds) for display text, None to keep forever
blacklist	['keepassx']	list with blacklisted wm_class, sadly not every clipboard window sets them, keepassx does.Clipboard contents from blacklisted wm_classes will be replaced by the value of blacklist_text.
blacklist_text	'********'	text to display when the wm_class is blacklisted

Clock

class libqtile.widget.**Clock**(***config*)
A simple but flexible text-based clock

Supported bar orientations: horizontal only

key	default	description
format	'%H:%M'	A Python datetime format string
update_interval	1	Update interval for the clock
timezone	None	The timezone to use for this clock, e.g. "US/Central" (or anything in /usr/share/zoneinfo). None means the default timezone.

Cmus

class libqtile.widget.**Cmus**(***config*)
A simple Cmus widget.

Show the artist and album of now listening song and allow basic mouse control from the bar:

- toggle pause (or play if stopped) on left click;
- skip forward in playlist on scroll up;
- skip backward in playlist on scroll down.

Cmus (https://cmus.github.io) should be installed.

Supported bar orientations: horizontal only

key	default	description
play_color	'00ff00'	Text colour when playing.
noplay_color	'cecece'	Text colour when not playing.
max_chars	0	Maximum number of characters to display in widget.
update_interval	0.5	Update Time in seconds.

Countdown

class libqtile.widget.**Countdown**(**config*)
A simple countdown timer text widget

Supported bar orientations: horizontal only

key	default	description
format	'{D}d {H}h {M}m {S}s'	Format of the displayed text. Available variables:{D} == days, {H} == hours, {M} == minutes, {S} seconds.
update_interval	1.0	Update interval in seconds for the clock
date	datetime.datetime(2016, 3, 11, 13, 50, 18, 413068)	The datetime for the endo of the countdown

CurrentLayout

class libqtile.widget.**CurrentLayout**(*width=CALCULATED*, **config*)
Display the name of the current layout of the current group of the screen, the bar containing the widget, is on.

Supported bar orientations: horizontal only

key	default	description
font	'Arial'	Default font
fontsize	None	Font size. Calculated if None.
padding	None	Padding. Calculated if None.
foreground	'ffffff'	Foreground colour
fontshadow	None	font shadow color, default is None(no shadow)
markup	False	Whether or not to use pango markup

CurrentScreen

class libqtile.widget.**CurrentScreen**(*width=CALCULATED*, **config*)
Indicates whether the screen this widget is on is currently active or not

Supported bar orientations: horizontal only

key	default	description
active_text	'A'	Text displayed when the screen is active
inactive_text	'I'	Text displayed when the screen is inactive
active_color	'00ff00'	Color when screen is active
inactive_color	'ff0000'	Color when screen is inactive

DF

class libqtile.widget.**DF**(**config*)
Disk Free Widget

By default the widget only displays if the space is less than warn_space.

Supported bar orientations: horizontal only

key	default	description
partition	'/'	the partition to check space
warn_color	'ff0000'	Warning color
warn_space	2	Warning space in scale defined by the measure option.
visible_on_warn	True	Only display if warning
measure	'G'	Measurement (G, M, B)
format	'{p} ({uf}{m})'	String format (p: partition, s: size, f: free space, uf: user free space, m: measure)
update_interval	60	The update interval.

DebugInfo

class libqtile.widget.**DebugInfo**(**config*)
Displays debugging infos about selected window

Supported bar orientations: horizontal only

key	default	description
font	'Arial'	Default font
fontsize	None	Font size. Calculated if None.
padding	None	Padding. Calculated if None.
foreground	'ffffff'	Foreground colour
fontshadow	None	font shadow color, default is None(no shadow)
markup	False	Whether or not to use pango markup

GenPollText

class libqtile.widget.**GenPollText**(**config*)
A generic text widget that polls using poll function to get the text

Supported bar orientations: horizontal only

key	default	description
func	None	Poll Function

GenPollUrl

class libqtile.widget.**GenPollUrl**(**config*)
A generic text widget that polls an url and parses it using parse function

Supported bar orientations: horizontal only

key	default	description
url	None	Url
data	None	Post Data
parse	None	Parse Function
json	True	Is Json?
user_agent	'Qtile'	Set the user agent
headers	{}	Extra Headers

GmailChecker

class libqtile.widget.**GmailChecker**(**config*)
A simple gmail checker

Supported bar orientations: horizontal only

key	default	description
update_interval	30	Update time in seconds.
username	None	username
password	None	password
email_path	'INBOX'	email_path
fmt	'inbox[%s],unseen[%s]'	fmt
status_only_unseen	False	Only show unseen messages

GroupBox

class libqtile.widget.**GroupBox**(**config*)
A widget that graphically displays the current group

Supported bar orientations: horizontal only

key	default	description
active	'FFFFFF'	Active group font colour
inactive	'404040'	Inactive group font colour
highlight_method	'border'	Method of highlighting ('border', 'block', 'text', or 'line')Uses *_border color settings
rounded	True	To round or not to round box borders
this_current_screen_border	'215578'	Border or line colour for group on this screen when focused.
this_screen_border	'215578'	Border or line colour for group on this screen when unfocused.
other_screen_border	'404040'	Border or line colour for group on other screen.
highlight_color	['000000', '282828']	Active group highlight color when using 'line' highlight method.
urgent_alert_method	'border'	Method for alerting you of WM urgent hints (one of 'border', 'text', 'block', or 'line')
urgent_text	'FF0000'	Urgent group font color
urgent_border	'FF0000'	Urgent border or line color
disable_drag	False	Disable dragging and dropping of group names on widget
invert_mouse_wheel	False	Whether to invert mouse wheel group movement
visible_groups	None	Groups that will be visible (if set to None or [], all groups will be visible)

HDDBusyGraph

class libqtile.widget.**HDDBusyGraph**(**config*)
Display HDD busy time graph

Parses /sys/block/<dev>/stat file and extracts overall device IO usage, based on io_ticks's value. See https://www.kernel.org/doc/Documentation/block/stat.txt

Supported bar orientations: horizontal only

key	default	description
device	'sda'	Block device to display info for

HDDGraph

class libqtile.widget.**HDDGraph**(***config*)
Display HDD free or used space graph

Supported bar orientations: horizontal only

key	default	description
path	'/'	Partition mount point.
space_type	'used'	free/used

Image

class libqtile.widget.**Image**(*length=CALCULATED*, *width=None*, ***config*)
Display a PNG image on the bar

Supported bar orientations: horizontal and vertical

key	default	description
scale	True	Enable/Disable image scaling
filename	None	PNG Image filename. Can contain '~'

ImapWidget

class libqtile.widget.**ImapWidget**(***config*)
Email IMAP widget

This widget will scan one of your imap email boxes and report the number of unseen messages present. I've configured it to only work with imap with ssl. Your password is obtained from the Gnome Keyring.

Writing your password to the keyring initially is as simple as (changing out <userid> and <password> for your userid and password):

1.create the file ~/.local/share/python_keyring/keyringrc.cfg with the following contents:

[backend] default-keyring=keyring.backends.Gnome.Keyring keyring-path=/home/<userid>/.local/share/keyring/

2.Execute the following python shell script once:

#!/usr/bin/env python3 import keyring user = <userid> password = <password> keyring.set_password('imapwidget', user, password)

mbox names must include the path to the mbox (except for the default INBOX). So, for example if your mailroot is ~/Maildir, and you want to look at the mailbox at HomeMail/fred, the mbox setting would be: mbox="'~/Maildir/HomeMail/fred"'. Note the nested sets of quotes! Labels can be whatever you choose, of course.

Supported bar orientations: horizontal only

key	default	description
mbox	'"INBOX"'	mailbox to fetch
label	'INBOX'	label for display
user	None	email username
server	None	email server name

KeyboardKbdd

class libqtile.widget.**KeyboardKbdd**(***config*)
Widget for changing keyboard layouts per window, using kbdd

kbdd should be installed and running, you can get it from: https://github.com/qnikst/kbdd

Supported bar orientations: horizontal only

key	default	description
update_interval	1	Update interval in seconds.
configured_keyboards	['us', 'ir']	your predefined list of keyboard layouts.example: ['us', 'ir', 'es']
colours	None	foreground colour for each layouteither 'None' or a list of colours.example: ['ffffff', 'E6F0AF'].

KeyboardLayout

class libqtile.widget.**KeyboardLayout**(***config*)
Widget for changing and displaying the current keyboard layout

It requires setxkbmap to be available in the system.

Supported bar orientations: horizontal only

key	default	description
update_interval	1	Update time in seconds.
configured_keyboards	['us']	A list of predefined keyboard layouts represented as strings. For example: ['us', 'us colemak', 'es', 'fr'].

KhalCalendar

class libqtile.widget.**KhalCalendar**(***config*)
Khal calendar widget

This widget will display the next appointment on your Khal calendar in the qtile status bar. Appointments within the "reminder" time will be highlighted.

Supported bar orientations: horizontal only

key	default	description
reminder_color	'FF0000'	color of calendar entries during reminder time
foreground	'FFFF33'	default foreground color
remindertime	10	reminder time in minutes
lookahead	7	days to look ahead in the calendar

LaunchBar

class libqtile.widget.**LaunchBar**(*progs=None*, *width=CALCULATED*, ***config*)
A widget that display icons to launch the associated command

Parameters progs :

a list of tuples (software_name, command_to_execute, comment), for example:

```
                ('thunderbird', 'thunderbird -safe-mode', 'launch thunderbird in safe mode')
                ('logout', 'qsh:self.qtile.cmd_shutdown()', 'logout from qtile')
```

Supported bar orientations: horizontal only

key	default	description
padding	2	Padding between icons
default_icon	'/usr/share/icons/oxygen/256x256/mimetypes/application-x-executable.png'	Default icon not found

Maildir

class libqtile.widget.**Maildir**(***config*)
A simple widget showing the number of new mails in maildir mailboxes

Supported bar orientations: horizontal only

key	default	description
maildirPath	'~/Mail'	path to the Maildir folder
subFolders	[]	The subfolders to scan (e.g. [{"path": "INBOX", "label": "Home mail"}, {"path": "spam", "label": "Home junk"}]
separator	' '	the string to put between the subfolder strings.

Memory

class libqtile.widget.**Memory**(***config*)
Displays memory usage

Supported bar orientations: horizontal only

key	default	description
fmt	'{MemUsed}M/{MemTotal}M'	see /proc/meminfo for field names

MemoryGraph

class libqtile.widget.**MemoryGraph**(***config*)
Displays a memory usage graph

Supported bar orientations: horizontal only

key	default	description
graph_color	'18BAEB'	Graph color
fill_color	'1667EB.3'	Fill color for linefill graph
border_color	'215578'	Widget border color
border_width	2	Widget border width
margin_x	3	Margin X
margin_y	3	Margin Y
samples	100	Count of graph samples.
frequency	1	Update frequency in seconds
type	'linefill'	'box', 'line', 'linefill'
line_width	3	Line width
start_pos	'bottom'	Drawer starting position ('bottom'/'top')

Moc

class libqtile.widget.**Moc**(***config*)
A simple MOC widget.

Show the artist and album of now listening song and allow basic mouse control from the bar:

- toggle pause (or play if stopped) on left click;
- skip forward in playlist on scroll up;
- skip backward in playlist on scroll down.

MOC (http://moc.daper.net) should be installed.

Supported bar orientations: horizontal only

key	default	description
play_color	'00ff00'	Text colour when playing.
noplay_color	'cecece'	Text colour when not playing.
max_chars	0	Maximum number of characters to display in widget.
update_interval	0.5	Update Time in seconds.

Mpd

class libqtile.widget.**Mpd**(*host='localhost'*, *port=6600*, *password=False*, *fmt_playing='%a - %t [%v%%]'*, *fmt_stopped='Stopped [%v%%]'*, *msg_nc='Mpd off'*, *do_color_progress=True*, ***config*)
A widget for the Music Player Daemon (MPD)

Initialize the widget with the following parameters

Parameters **host :**

host to connect to

port :

port to connect to

password :

password to use

fmt_playing :

format string to display when playing/paused

fmt_stopped :

format strings to display when stopped

msg_nc :

which message to show when we're not connected

do_color_progress :

whether to indicate progress in song by altering message color

width :

A fixed width, or bar.CALCULATED to calculate the width automatically (which is recommended).

Supported bar orientations: horizontal only

key	default	description
foreground_progress	'ffffff'	Foreground progress colour
reconnect	False	attempt to reconnect if initial connection failed
reconnect_interval	1	Time to delay between connection attempts.
update_interval	0.5	Update Time in seconds.

Mpris

class libqtile.widget.**Mpris**(*name='clementine'*, *width=CALCULATED*, *objname='org.mpris.clementine'*, ***config*)

MPRIS player widget

A widget which displays the current track/artist of your favorite MPRIS player. It should work with all players which implement a reasonably correct version of MPRIS, though I have only tested it with clementine.

Supported bar orientations: horizontal only

key	default	description
font	'Arial'	Default font
fontsize	None	Font size. Calculated if None.
padding	None	Padding. Calculated if None.
foreground	'ffffff'	Foreground colour
fontshadow	None	font shadow color, default is None(no shadow)
markup	False	Whether or not to use pango markup

Mpris2

class libqtile.widget.**Mpris2**(***config*)

An MPRIS 2 widget

A widget which displays the current track/artist of your favorite MPRIS player. It should work with all MPRIS 2 compatible players which implement a reasonably correct version of MPRIS, though I have only tested it with audacious. This widget scrolls the text if neccessary and information that is displayed is configurable.

Supported bar orientations: horizontal only

key	default	description
name	'audacious'	Name of the MPRIS widget.
objname	'org.mpris.MediaPlayer2.audacious'	DBUS MPRIS 2 compatible player identifier- Find it out with dbus-monitor - Also see: http://specifications.freedesktop.org/mpris-spec/latest/#Bus-Name-Policy
display_metadata	['xesam:title', 'xesam:album', 'xesam:artist']	Which metadata identifiers to display. See http://www.freedesktop.org/wiki/Specifications/mpris-spec/metadata/#index5h3 for available values
scroll_chars	30	How many chars at once to display.
scroll_interval	0.5	Scroll delay interval.
scroll_wait_intervals	8	Wait x scroll_interval beforescrolling/removing text

Net

class libqtile.widget.**Net**(**config*)
Displays interface down and up speed

Supported bar orientations: horizontal only

key	default	description
interface	'wlan0'	The interface to monitor
update_interval	1	The update interval.

NetGraph

class libqtile.widget.**NetGraph**(**config*)
Display a network usage graph

Supported bar orientations: horizontal only

key	default	description
interface	'auto'	Interface to display info for ('auto' for detection)
bandwidth_type	'down'	down(load)/up(load)

Notify

class libqtile.widget.**Notify**(*width=CALCULATED*, **config*)
A notify widget

Supported bar orientations: horizontal only

key	default	description
foreground_urgent	'ff0000'	Foreground urgent priority colour
foreground_low	'dddddd'	Foreground low priority colour
default_timeout	None	Default timeout (seconds) for notifications
audiofile	None	Audiofile played during notifications

Pacman

class libqtile.widget.**Pacman**(**config*)
Shows number of available updates

Needs the pacman package manager installed. So will only work in Arch Linux installation.

Supported bar orientations: horizontal only

key	default	description
unavailable	'ffffff'	Unavailable Color - no updates.
execute	None	Command to execute on click
update_interval	60	The update interval.

Prompt

class libqtile.widget.**Prompt**(*name='prompt'*, **config*)
A widget that prompts for user input

Input should be started using the `.startInput()` method on this class.

Supported bar orientations: horizontal only

key	default	description
cursor	True	Show a cursor
cursorblink	0.5	Cursor blink rate. 0 to disable.
cursor_color	'bef098'	Color for the cursor and text over it.
prompt	'{prompt}: '	Text displayed at the prompt
record_history	True	Keep a record of executed commands
max_history	100	Commands to keep in history. 0 for no limit.
bell_style	'audible'	Alert at the begin/end of the command history. Possible values: 'audible', 'visual' and None.
visual_bell_color	'ff0000'	Color for the visual bell (changes prompt background).
visual_bell_time	0.2	Visual bell duration (in seconds).

Sep

class libqtile.widget.**Sep**(*height_percent=None*, ***config*)
A visible widget separator

Supported bar orientations: horizontal and vertical

key	default	description
padding	2	Padding on either side of separator.
linewidth	1	Width of separator line.
foreground	'888888'	Separator line colour.
size_percent	80	Size as a percentage of bar size (0-100).

She

class libqtile.widget.**She**(***config*)
Widget to display the Super Hybrid Engine status

Can display either the mode or CPU speed on eeepc computers.

Supported bar orientations: horizontal only

key	default	description
device	'/sys/devices/platform/eeepc/cpufv'	sys path to cpufv
format	'speed'	Type of info to display "speed" or "name"
update_interval	0.5	Update Time in seconds.

Spacer

class libqtile.widget.**Spacer**(*length=STRETCH*, *width=None*)
Just an empty space on the bar

Often used with length equal to bar.STRETCH to push bar widgets to the right or bottom edge of the screen.

Parameters **length :**

Length of the widget. Can be either bar.STRETCH or a length in pixels.

width :

DEPRECATED, same as `length`.

Supported bar orientations: horizontal and vertical

key	default	description
`background`	`None`	Widget background color

SwapGraph

class `libqtile.widget.`**`SwapGraph`**`(**config)`
Display a swap info graph

Supported bar orientations: horizontal only

key	default	description
`graph_color`	`'18BAEB'`	Graph color
`fill_color`	`'1667EB.3'`	Fill color for linefill graph
`border_color`	`'215578'`	Widget border color
`border_width`	`2`	Widget border width
`margin_x`	`3`	Margin X
`margin_y`	`3`	Margin Y
`samples`	`100`	Count of graph samples.
`frequency`	`1`	Update frequency in seconds
`type`	`'linefill'`	'box', 'line', 'linefill'
`line_width`	`3`	Line width
`start_pos`	`'bottom'`	Drawer starting position ('bottom'/'top')

Systray

class `libqtile.widget.`**`Systray`**`(**config)`
A widget that manages system tray

Supported bar orientations: horizontal only

key	default	description
`icon_size`	`20`	Icon width
`padding`	`5`	Padding between icons

TaskList

class `libqtile.widget.`**`TaskList`**`(**config)`
Displays the icon and name of each window in the current group

Contrary to WindowTabs this is an interactive widget. The window that currently has focus is highlighted.

Supported bar orientations: horizontal only

key	default	description
font	'Arial'	Default font
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
fontshadow	None	font shadow color, default is None(no shadow)
borderwidth	2	Current group border width
border	'215578'	Border colour
rounded	True	To round or not to round borders
highlight_method	'border'	Method of highlighting (one of 'border' or 'block') Uses *_border color settings
urgent_border	'FF0000'	Urgent border color
urgent_alert_method	'border'	Method for alerting you of WM urgent hints (one of 'border' or 'text')
max_title_width	200	size in pixels of task title

TextBox

class libqtile.widget.**TextBox**(*text=' '*, *width=CALCULATED*, ***config*)
A flexible textbox that can be updated from bound keys, scripts and qsh

Supported bar orientations: horizontal only

key	default	description
font	'Arial'	Text font
fontsize	None	Font pixel size. Calculated if None.
fontshadow	None	font shadow color, default is None(no shadow)
padding	None	Padding left and right. Calculated if None.
foreground	'#ffffff'	Foreground colour.

ThermalSensor

class libqtile.widget.**ThermalSensor**(***config*)
Widget to display temperature sensor information

For using the thermal sensor widget you need to have lm-sensors installed. You can get a list of the tag_sensors executing "sensors" in your terminal. Then you can choose which you want, otherwise it will display the first available.

Supported bar orientations: horizontal only

key	default	description
metric	True	True to use metric/C, False to use imperial/F
show_tag	False	Show tag sensor
update_interval	2	Update interval in seconds
tag_sensor	None	Tag of the temperature sensor. For example: "temp1" or "Core 0"
threshold	70	If the current temperature value is above, then change to foreground_alert colour
foreground_alert	'ff0000'	Foreground colour alert

Volume

class libqtile.widget.**Volume**(***config*)
Widget that display and change volume

If theme_path is set it draw widget as icons.

Supported bar orientations: horizontal only

key	default	description
cardid	None	Card Id
device	'default'	Device Name
channel	'Master'	Channel
padding	3	Padding left and right. Calculated if None.
theme_path	None	Path of the icons
update_interval	0.2	Update time in seconds.
emoji	False	Use emoji to display volume states, only if theme_path is not set.The specified font needs to contain the correct unicode characters.
mute_command	None	Mute command
volume_up_command	None	Volume up command
volume_down_command	None	Volume down command
get_volume_command	None	Command to get the current volume

Wallpaper

class libqtile.widget.**Wallpaper**(***config*)

Supported bar orientations: horizontal only

key	default	description
directory	'/home/docs/Pictures/wallpapers/'	Wallpaper Directory
wallpaper	None	Wallpaper
wallpaper_command	None	Wallpaper command

WindowName

class libqtile.widget.**WindowName**(*width=STRETCH*, ***config*)

Displays the name of the window that currently has focus

Supported bar orientations: horizontal only

key	default	description
show_state	True	show window status before window name

WindowTabs

class libqtile.widget.**WindowTabs**(***config*)

Displays the name of each window in the current group. Contrary to TaskList this is not an interactive widget. The window that currently has focus is highlighted.

Supported bar orientations: horizontal only

key	default	description
separator	' | '	Task separator text.
selected	('<', '>')	Selected task indicator

Wlan

class libqtile.widget.**Wlan**(***config*)

Displays Wifi ssid and quality

Supported bar orientations: horizontal only

key	default	description
`interface`	`'wlan0'`	The interface to monitor
`update_interval`	`1`	The update interval.
`disconnected_message`	`'Disconnected'`	String to show when the wlan is diconnected.
`format`	`'{essid} {quality}/70'`	Display format. For percents you can use "{essid} {percent:2.0%}"

YahooWeather

class `libqtile.widget.`**`YahooWeather`**(***config*)

A weather widget, data provided by the Yahoo! Weather API.

Format options:

- astronomy_sunrise
- astronomy_sunset
- atmosphere_humidity
- atmosphere_visibility
- atmosphere_pressure
- atmosphere_rising
- condition_text
- condition_code
- condition_temp
- condition_date
- location_city
- location_region
- location_country
- units_temperature
- units_distance
- units_pressure
- units_speed
- wind_chill

Supported bar orientations: horizontal only

key	default	description
location	None	Location to fetch weather for. Ignored if woeid is set.
woeid	None	Where On Earth ID. Auto-calculated if location is set.
format	'{location_city}: {condition_temp} °{units_temperature}'	Display format
metric	True	True to use metric/C, False to use imperial/F
up	'^'	symbol for rising atmospheric pressure
down	'v'	symbol for falling atmospheric pressure
steady	's'	symbol for steady atmospheric pressure

4.2 Frequently Asked Questions

4.2.1 Why the name Qtile?

Users often wonder, why the Q? Does it have something to do with Qt? No. Below is an IRC excerpt where cortesi explains the great trial that ultimately brought Qtile into existence, thanks to the benevolence of the Open Source Gods. Praise be to the OSG!

```
ramnes:  what does Qtile mean?
ramnes:  what's the Q?
@tych0:  ramnes: it doesn't :)
@tych0:  cortesi was just looking for the first letter that wasn't registered
         in a domain name with "tile" as a suffix
@tych0:  qtile it was :)
cortesi: tych0, dx: we really should have something more compelling to
         explain the name. one day i was swimming at manly beach in sydney,
         where i lived at the time. suddenly, i saw an enormous great white
         right beside me. it went for my leg with massive, gaping jaws, but
         quick as a flash, i thumb-punched it in both eyes. when it reared
         back in agony, i saw that it had a jagged, gnarly scar on its
         stomach... a scar shaped like the letter "Q".
cortesi: while it was distracted, i surfed a wave to shore. i knew that i
         had to dedicate my next open source project to the ocean gods, in
         thanks for my lucky escape. and thus, qtile got its name...
```

4.2.2 When I first start xterm/urxvt/rxvt containing an instance of Vim, I see text and layout corruption. What gives?

Vim is not handling terminal resizes correctly. You can fix the problem by starting your xterm with the "-wf" option, like so:

```
xterm -wf -e vim
```

Alternatively, you can just cycle through your layouts a few times, which usually seems to fix it.

4.2.3 How do I know which modifier specification maps to which key?

To see a list of modifier names and their matching keys, use the `xmodmap` command. On my system, the output looks like this:

```
$ xmodmap
xmodmap:  up to 3 keys per modifier, (keycodes in parentheses):

shift       Shift_L (0x32),  Shift_R (0x3e)
lock        Caps_Lock (0x9)
control     Control_L (0x25),  Control_R (0x69)
mod1        Alt_L (0x40),  Alt_R (0x6c),  Meta_L (0xcd)
mod2        Num_Lock (0x4d)
mod3
mod4        Super_L (0xce),  Hyper_L (0xcf)
mod5        ISO_Level3_Shift (0x5c),  Mode_switch (0xcb)
```

4.2.4 My "pointer mouse cursor" isn't the one I expect it to be!

Qtile should set the default cursor to left_ptr, you must install xcb-util-cursor if you want support for themed cursors.

4.3 License

- genindex

A

B

C

D

E

F