Qtile Documentation

Release 0.9.1

Aldo Cortesi

July 08, 2015

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:

- qtile-git development branch of qtile.
- qtile-python3-git development branch of qtile for python3.
- qtile stable branch(release) of qtile.

Using an AUR Helper

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

```
# for release
yaourt -S qtile
# or for develop
yaourt -S qtile-git
# or for develop python3
yaourt -S qtile-python3-git
```

Using pacman

You can choose python3 or python2

```
# for python3
sudo pacman -S python pango
# or for python2
sudo pacman -S python2 pango
```

Also you need these packages from AUR:

For python3:

- qtile-python3-git
- python-xcffib
- python-cairocffi

For python2:

- qtile-git
- python2-xcffib
- python2-cairocffi
- trollius

Installing AUR packages without helper

To install these packages, download the .tar.gz's from the AUR and run the following commands for each:

```
tar -xvzf <packagename>-<vernum>.tar.gz
cd <packagename>-<vernum>
makepkg -s
sudo pacman -U <packagename>
```

Please see the Arch Wiki for more information on installing packages from the AUR:

http://wiki.archlinux.org/index.php/AUR#Installing_packages

Installing on Ubuntu

PPA on Launchpad

Packages 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
```

Manual Installation Guides

- Installing Qtile on Ubuntu 11.10
- Installing Qtile on Ubuntu 10.10

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).

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 (`sudo apt-get install libxcb-render0-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. Once you've got that 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 infastructure 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 infastructure introduced in PEP 3156. You can install this from PyPi:

  ```
  pip install trollius
  ```

importlib

- Python <=2.6 you will need to install importlib from PyPi:

  ```
  pip install importlib
  ```

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
```

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 `-f` 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 sutiable 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 control 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 `qweruiop`. 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!)

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.

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.

`__init__`(*title=None*, *wm_class=None*, *role=None*, *wm_type=None*, *wm_instance_class=None*, *net_wm_pid=None*)

`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)

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.

`__init__`(*name*, *matches=None*, *exclusive=False*, *spawn=None*, *layout=None*, *layouts=None*, *persist=True*, *init=True*, *layout_opts=None*, *screen_affinity=None*, *position=9223372036854775807*)

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*) – this will be `exec()` d when the group is created
- **`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.

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")
```

Keys

The `keys` variable defines Qtile's key bindings.

The command.lazy object

`command.lazy` 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()
    )
]
```

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()
)
```

Lazy functions

This is overview of the commonly used functions for the key bindings.

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.nextlayout())`	Use next layout on the actual group
`lazy.prevlayout())`	Use previous layout on the actual group
`lazy.screen.nextgroup())`	Move to the group on the right
`lazy.screen.prevgroup())`	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()`	Incrase 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

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`

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())
]
```

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 background colors and gradients, e.g. `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.

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('~')
    subprocess.call([home + '/.config/qtile/autostart.sh'])
```

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

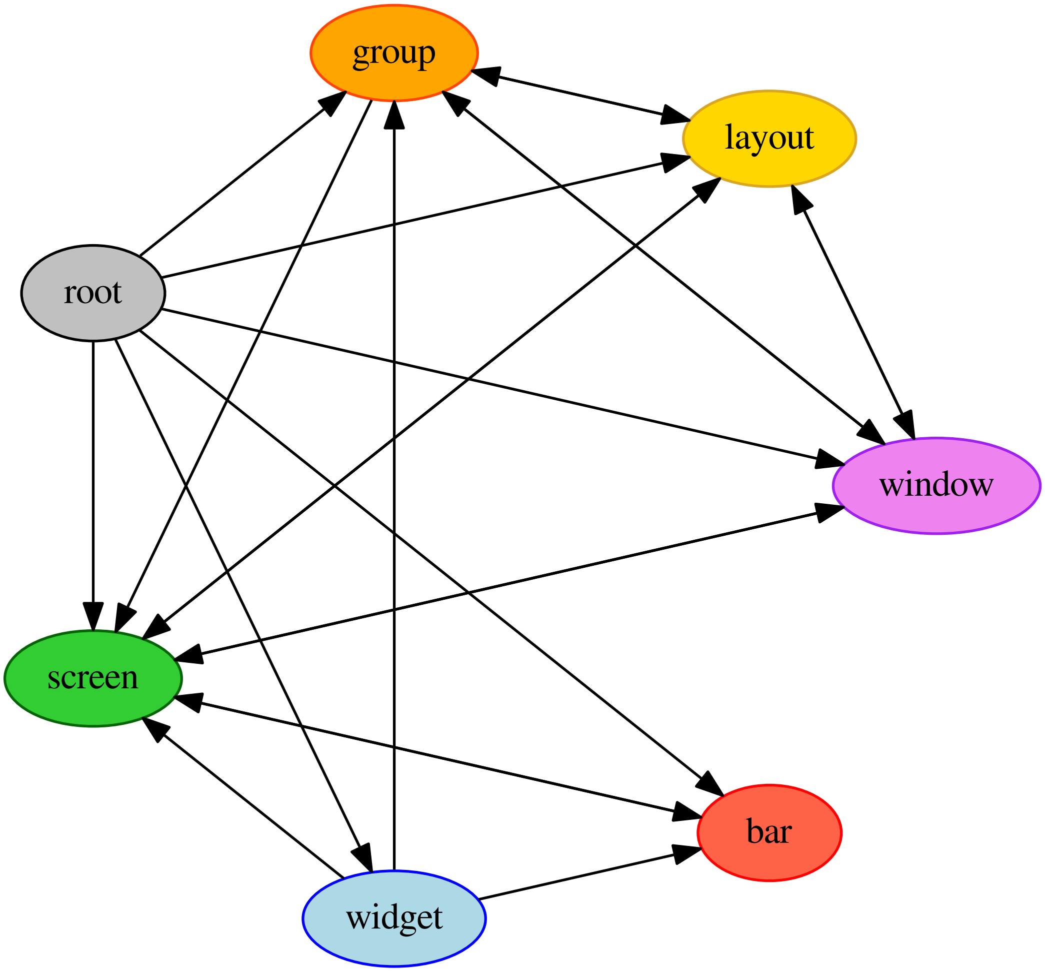
Qtile's command API is based on a graph of objects, where each object has a set of associated commands. The graph and object commands are used in a number of different places:

- Commands can be bound to keys in the Qtile configuration file.
- Commands can be called through qsh, the Qtile shell.
- 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:

```
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 offset	Yes	c.layout[2] c.layout
screen	Integer offset	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"]
```

2.3 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" commmands 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 Documentation

The shell help 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.
```

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.

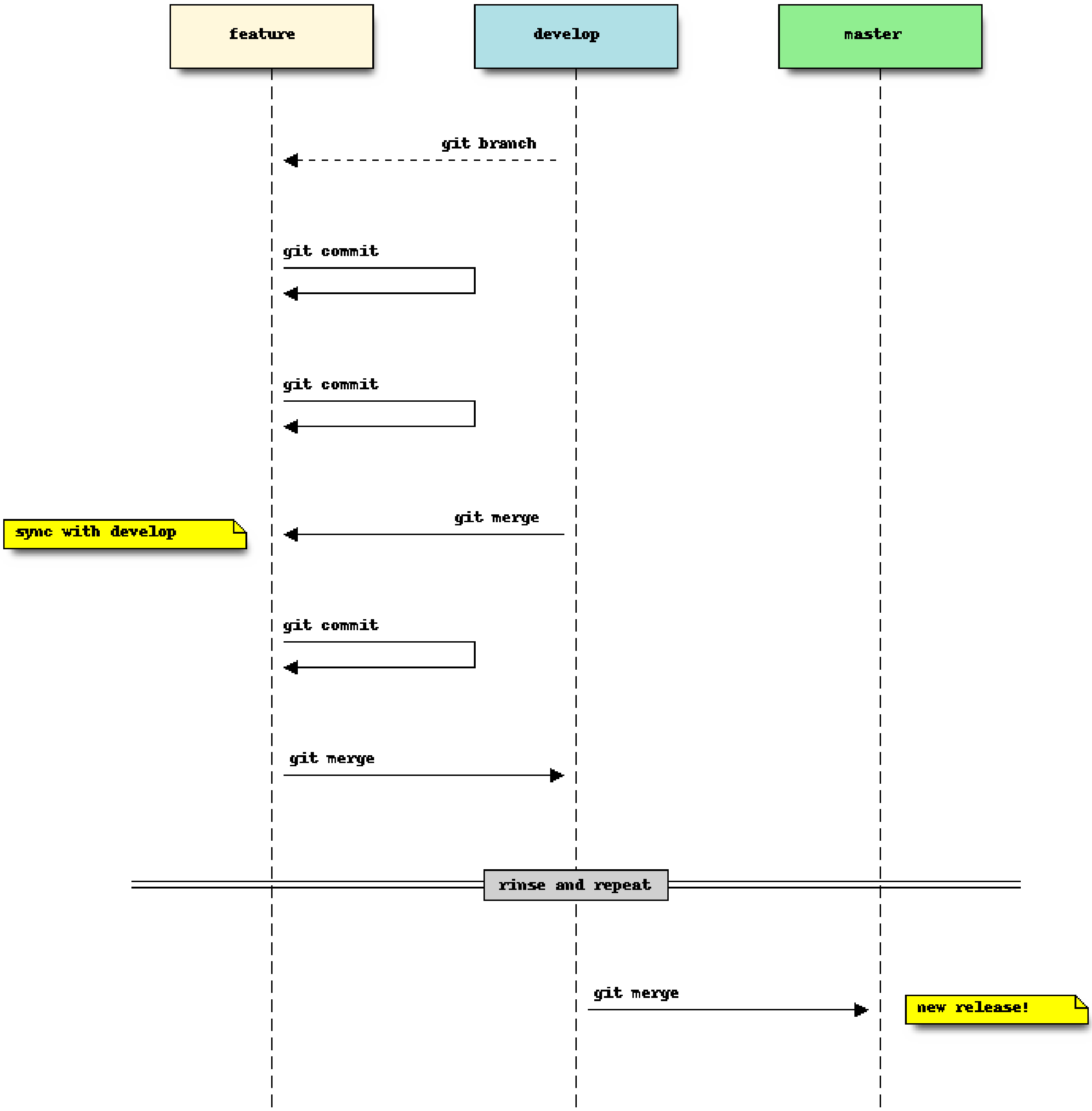
3.1.2 Writing code

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 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

3.2.3 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.

3.2.4 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.

3.2.5 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.

3.2.6 Deprecation policy

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

3.2.7 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.

3.2.8 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 Built-in Hooks

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

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

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

- arguments: window.Window object of the killed window.

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

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

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

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

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

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

`subscribe.`**`client_focus`**(*func*)
Called whenver focus changes.

- arguments: window.Window object of the new focus.

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

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

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

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

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

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

Example:

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

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

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

subscribe.**current_screen_change**(*func*)
Called when the current screen (i.e. the screen with focus) changes; no arguments.

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

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

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

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

4.1.2 Built-in Layouts

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 winows.
`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`	`set(['dialog', 'notification', 'splash', 'toolbar', 'utility'])`	default wm types to automatically float

`__init__`(*float_rules=None*, ***config*)

If you have certain apps that you always want to float you can provide `float_rules` to do so. `float_rules` is a list of dictionaries containing:

{wname: WM_NAME, wmclass: WM_CLASS role: WM_WINDOW_ROLE}

The keys must be specified as above. You only need one, but you need to provide the value for it. When a new window is opened it's `match` method is called with each of these rules. If one matches, the window will float. The following will float gimp and skype:

float_rules=[dict(wmclass="skype"), dict(wmclass="gimp")]

Specify these in the `floating_layout` in your config.

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 winows.
`border_width`	`1`	Border width.
`name`	`'matrix'`	Name of this layout.
`margin`	`0`	Margin of the layout

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

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.

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

This layout attempts to 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` and `cmd_shrink` methods while the main pane is in focus.

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

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".

```
---------------------          ---------------------
|            |  2   |          |   2   |           |
|            |______|          |_______|           |
|            |  3   |          |   3   |           |
|     1      |______|          |_______|     1     |
|            |  4   |          |   4   |           |
|            |      |          |       |           |
---------------------          ---------------------
```

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 winows.
`border_width`	`2`	Border width.
`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

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 winows.
`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 inrement per ratio increment
`fancy`	`False`	Use a different method to calculate window sizes.

class `libqtile.layout.slice.`**`Slice`**(*side*, *width*, ***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 0x7f4a9724ba10>`	Fallback layout

class `libqtile.layout.stack.`**`Stack`**(***config*)

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 winows.
`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

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 winows.
`border_width`	`1`	Border width.
`name`	`'tile'`	Name of this layout.
`margin`	`0`	Margin of the layout

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 magin 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.

class `libqtile.layout.verticaltile.`**`VerticalTile`**`(`***config*`)`

VerticalTile implements a 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.

```
-----------------                -----------------  ---
|               |                |               |   |
|       1       |  <-- Panes     |               |   |
|               |        |       |               |   |
|---------------|        |       |               |   |
|               |        |       |               |   |
|       2       |  <-----+       |       1       |   |  Master Area
|               |        |       |               |   |
|---------------|        |       |               |   |
|               |        |       |               |   |
|       3       |  <-----+       |               |   |
|               |        |       |               |   |
|---------------|        |       |---------------|  ---
|               |        |       |       2       |   |
|       4       |  <-----+       |---------------|   |  Secondary Area
|               |                |       3       |   |
-----------------                -----------------  ---
```

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 winows.
`border_width`	`1`	Border width.
`margin`	`0`	Border margin.
`name`	`'VerticalTile'`	Name of this layout.

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

4.1.3 Built-in Widgets

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

A widget that graphically displays the current group.

key	default	description
`border`	`'000000'`	group box border color

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

A simple widget to show the current brightness of a monitor.

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

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

A simple but flexible text-based battery widget.

key	default	description
`low_foreground`	`'FF0000'`	font color when battery is low
`format`	`'{char} {percent:2.0%} {hour:d}:{min:02d}'`	Display format
`charge_char`	`'^'`	Character to indicate the battery is charging
`discharge_char`	`'V'`	Character to indicate the battery is discharging
`low_percentage`	`0.1`	0 < x < 1 at which to indicate battery is low with low_foreground
`hide_threshold`	`None`	Hide the text when there is enough energy

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

Battery life indicator widget

key	default	description
`theme_path`	`'/home/docs/checkouts/readthedocs.org/user_builds/qtile/checkouts/v0.9.1/libqt`	Path of the icons
`custom_icons`	`{}`	dict containing key->filename icon map

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.

key	default	description
`currency`	`''`	The currency the value of bitcoin is displayed in
`format`	`'BTC Buy:  {buy}, Sell: {sell}'`	Display format, allows buy, sell, high, low, avg, vol, vol_cur, last, variables.

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

Display CPU usage graph.

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

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

Display RSS feeds updates using the canto console reader.

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

class `libqtile.widget.`**`Clipboard`**`(`*width=CALCULATED*, ***config*`)`

Display current clipboard contents.

key	default	description
`selection`	`'CLIPBOARD'`	the 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

class `libqtile.widget.Clock`(*fmt=None*, ***config*)

A simple but flexible text-based clock.

key	default	description
`format`	`'%H:%M'`	A Python datetime format string
`update_interval`	`1.0`	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.

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

A simple countdown timer text widget.

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(2015, 7, 8, 17, 36, 55, 980875)`	The datetime for the endo of the countdown

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.

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

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

Disk Free Widget

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

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 inteval.

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

Displays debugging infos about selected window

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

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

A generic text widget that polls using poll function to get the text

key	default	description
`poll`	`None`	Poll Function

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

A generic text widget that polls an url and parses it using parse function

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

class `libqtile.widget.`**`GmailChecker`**`(`*settings=None*, ***config*`)`

A simple gmail checker.

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

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

A widget that graphically displays the current group.

key	default	description
`active`	`'FFFFFF'`	Active group font colour
`inactive`	`'404040'`	Inactive group font colour
`urgent_text`	`'FF0000'`	Urgent group font color
`highlight_method`	`'border'`	Method of highlighting (one of ‘border’ or ‘block’) Uses *_border color settings
`rounded`	`True`	To round or not to round borders
`this_current_screen_border`	`'215578'`	Border colour for group on this screen when focused.
`urgent_alert_method`	`'border'`	Method for alerting you of WM urgent hints (one of ‘border’, ‘text’ or ‘block’)
`disable_drag`	`False`	Disable dragging and dropping of group names on widget
`this_screen_border`	`'215578'`	Border colour for group on this screen.
`other_screen_border`	`'404040'`	Border colour for group on other screen.
`urgent_border`	`'FF0000'`	Urgent border color
`invert_mouse_wheel`	`False`	Whether to invert mouse wheel group movement

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

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

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

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

Display HDD free or used space graph.

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

class `libqtile.widget.`**`Image`**(*width=CALCULATED*, ***config*)

Display a PNG image on the bar.

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

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

Widget for changing and displaying the current keyboard layout. It requires setxkbmap to be available in the sytem.

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'].

class `libqtile.widget.`**`Maildir`**(*maildirPath=None*, *subFolders=None*, *separator=' '*, ***config*)

A simple widget showing the number of new mails in maildir mailboxes.

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.

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

Displays a memory usage graph.

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')

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

Displays interface down and up speed.

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

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

Display a network usage graph.

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

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

A notify widget

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

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

Shows number of available updates. Needs the pacman package manager installed. So will only work in Arch Linux installation.

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

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.

key	default	description
`cursorblink`	`0.5`	Cursor blink rate. 0 to disable.
`prompt`	`'{prompt}: '`	Text displayed at the prompt

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

A visible widget separator.

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

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

Widget to display the Super Hybrid Engine status. can display either the mode or CPU speed on eeepc computers.

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.

class `libqtile.widget.`**`Spacer`**(*width=STRETCH*)

Just an empty space on the bar. Often used with width equal to bar.STRETCH to push bar widgets to the right edge of the screen.

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

`__init__`(*width=STRETCH*)

- width: Width of the widget. Can be either `bar.STRETCH` or a width in pixels.

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

Display a swap info graph.

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')

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

A widget that manage system tray

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

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

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

class libqtile.widget.**TextBox**(*text=' '*, *width=CALCULATED*, ***config*)

A flexible textbox that can be updated from bound keys, scripts and qsh.

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.

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

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.

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

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

Widget that display and change volume if theme_path is set it draw widget as icons

key	default	description
`cardid`	`0`	Card Id
`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

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

Displays the name of the window that currently has focus.

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

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.

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

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

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} \xc2\xb0{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 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.

4.2.2 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)
```

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

Append the following to your `~/.config/qtile/config.py` file:

```
from libqtile import hook
@hook.subscribe.startup
def runner():
    import subprocess
    subprocess.Popen(['xsetroot', '-cursor_name', 'left_ptr'])
```

This will change your pointer cursor to the standard "Left Pointer" cursor you chose in your `~/.Xresources` file on Qtile startup.

4.3 License

This project is distributed under the MIT license.

Copyright (c) 2008, Aldo Cortesi All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- genindex

Index