Qtile Documentation

Release 0.7.0

Aldo Cortesi

January 15, 2015

Contents

Everything you need to know about Qtile.

CHAPTER 1

Installing

- **Getting started**: *Overview* | *From source*
- **Distro guides**: *Arch Linux* | *Funtoo* | *Gentoo* | *Ubuntu* | *Debian*

CHAPTER 2

Configuration

- **The basics**: *Default Configuration* | *Overview* | *Starting Qtile* | *Hooks* | *Running inside Gnome*
- **Config variables**: *groups* | *keys* | *layouts* | *mouse* | *screens*
- **Examples**: Default config | qtile-examples

CHAPTER 3

Commands and scripting

- **The basics**: *Overview*
- **Diving in**: *Scripting* | *qsh*

CHAPTER 4

Reference

CHAPTER 5

Miscellaneous

5.1 Installing Qtile

5.1.1 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*
- *Installing on Funtoo*
- *Installing on Gentoo*
- *Installing on Ubuntu*
- *Installing on Debian*

5.1.2 Installing From Source

Qtile relies on some cutting-edge features in PyCairo, XCB, and xpyb. Until the latest versions of these projects make it into distros, it's best to use recent checkouts from their repositories.

Here is a brief step-by-step guide to installing Qtile and its dependencies from source:

Installing from Source

5.2 Installing on Arch Linux

Qtile is available on the AUR as qtile-git.

5.2.1 Using an AUR Helper

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

```
yaourt -S qtile-git
```

5.2.2 Using pacman

Packages in Core:

- pygtk (32-Bit / 64-Bit)
- python2 (32-Bit / 64-Bit)
- cairo (32-Bit / 64-Bit)

If you don't have these already, they can be installed with:

```
sudo pacman -S pygtk python2 cairo
```

Packages in the AUR:

- pycairo-xcb-git
- xorg-xpyb-git
- qtile-git

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

5.3 Installing on Debian

Packages are available from Tycho's Ubuntu PPA repository. These are known to work with Debian Wheezy without any observed side effects.

Tych0's ppa is at https://launchpad.net/~tycho-s/+archive/ppa

create file /etc/apt/sources.d/tycho.list with the following:

```
deb http://ppa.launchpad.net/tycho-s/ppa/ubuntu precise main
deb-src http://ppa.launchpad.net/tycho-s/ppa/ubuntu precise main
```

Then to install tycho's key from the ppa above:

```
curl "http://keyserver.ubuntu.com:11371/pks/lookup?op=get&search=0x8516D5EEF453E809" > tyco.key
sudo apt-key add tyco.key
```

and finally install with:

```
sudo aptitude update
sudo aptitude install qtile
```

To get up and running copy the default config file.

```
mkdir -p ~/.config/qtile
cp /usr/lib/pymodules/python2.7/libqtile/resources/default-config.py ~/.config/qtile/config.py
```

5.4 Installing on Funtoo

5.4.1 Portage

The ebuild in portage is broken for now, as of missing xpyb support for pycairo, but will be fixed in some future releases.

5.4.2 Manual (Github)

This section is taken from the documents from Qtile. [1].

Dependencies

USE flags and keyword changes may have to be made for the packages taken from portage.

libxcb

libxcb can be emerged from portage.

```
# emerge -avt libxcb
```

xpyb

xpyb can be emerged from portage. Make sure that you are emerging xpyb-1.3.1 or above.

```
# emerge -avt xpyb
```

cairo

cairo can be emerged from portage. Make sure you have your USE flags set to:

```
X glib opengl svg xcb
```

and then emerge cairo:

```
# emerge -avt cairo
```

pygtk

pygtk can be merged from portage.

[1] Installation on Gentoo

```
# emerge pygtk
```

py2cairo

Needs to be build manually cause of reason above.

```
# git clone git://git.cairographics.org/git/py2cairo
# cd py2cairo
# ./configure --prefix=/path/to/virtualenv
# make
# sudo make install
```

As an alternative to virtualenv, you can

```
./configure --prefix=/usr
```

But the virtualenv is the recommended option in installation if you are an advanced user with python, else use the systemwide alternative.

qtile

```
# git clone git://github.com/qtile/qtile
# cd qtile
# sudo python setup.py install --record files_uninstall.txt
```

5.4.3 Setup

Copy either a config from the examples directory in the cloned qtile **(including a default config)**, a config you have found elsewhere, or create your own config.

```
# install -d ~/.config/qtile
# cp /path/to/cloned-qtile/examples/config/cortesi-config.py ~/.config/qtile/config.py
# cp /path/to/cloned-qtile/examples/config/dgroups.py ~/.config/qtile/config.py
# cp /path/to/cloned-qtile/examples/config/roger-config.py ~/.config/qtile/config.py
# cp /path/to/cloned-qtile/examples/config/tailhook-config.py ~/.config/qtile/config.py
```

Another config is config.py, this is based on dmpayton's config.py.

5.4.4 Testing Qtile Installation

If you have a running DE/WM already you can test your qtile config with the following steps:

Examples:

```
# Xephyr :1 -screen 800x600 -a -v -noreset
# Display=:1
# /path/to/qtile/qtile
```

or using the build in code: [2]

```
# echo "exec qtile" > .start_qtile ; xinit .start_qtile -- :1
```

For further information, see the Documentation section.

[2] https://groups.google.com/group/qtile-dev/browse_thread/thread/26191253a8190568_qtile-dev_Google_Group

5.4.5 dmenu

Qtile uses dmenu as the application launcher

```
# emerge dmenu
```

5.4.6 xinitrc

An example of preparing Qtile to start with the startup-session script for autostarting apps in the ~/.xinitrc:

```
#!/bin/zsh
xrdb -merge ~/.Xresources
xcompmgr &
if [[ $1 == "i3" ]]; then
    exec ck-launch-session dbus-launch --sh-syntax --exit-with-session i3 -V -d all > ~/.i3/i3log-$(
elif [[ $1 == "razor" ]]; then
    exec ck-launch-session dbus-launch startrazor
elif [[ $1 == "awesome" ]]; then
    exec ck-launch-session dbus-launch awesome
elif [[ $1 == "qtile" ]]; then
    exec ck-launch-session dbus-launch ~/.qtile-session
else
    echo "Choose a window manager"
fi
```

and the connected ~/.qtile-session

```
conky -c ~/.conky/conkyrc_grey &
sh ~/.fehbg &
dropbox &
```

5.4.7 X and RandR

NOTE: RandR and Xinerama do not play together. Use one or the other. I use an AMD HD 6870 with 3 monitors (2 DVI and 1 with an AMD validated Mini DisplayPort™ to DVI dongle).

Install xrandr:

```
# emerge x11-apps/xrandr
```

and if you want a GUI with xrandr:

```
# emerge x11-misc/arandr
```

If you do not have X configured yet, follow the link on the Gentoo Wiki.

My xorg.conf.d folder for example: 30-screen.conf.

Since the names of the monitors are already known in xrandr, I just use those names in my 30-screen.conf configuration. It doesn't matter what you use in your X configuration however.

Once you have X configured however you like, start qtile with either:

```
# startx
```

or, in a case similar to mine,

```
# xinit qtile
```

5.4.8 Starting with CDM

Another good tool for starting qtile is **CDM** (short for Console Display Manager). To make it work, just merge cdm

```
# emerge -avt cdm
```

and add it to autostart with

```
# cp /usr/share/cdm/zzz-cdm-profile.sh /etc/profile.d/zzz-cdm-profile.sh
```

Now add to /etc/X11/cdm/cdmrc the following lines:

```
binlist=(
    "/usr/bin/xinit ${HOME}/.start_qtile --:0"
    "/bin/bash --login"
    "/bin/zsh"
)
namelist=(qtile "Console bash" "Console zsh")
flaglist=(C C C)
consolekit=no
```

and check that ${HOME}/.start_qtile contains just the following

```
exec qtile
```

5.5 Installing on Gentoo

5.5.1 Prepare Dependencies

You may apply these USE-Flags:

```
echo "dev-python/pycairo xcb" >> /etc/portage/package.use
echo "x11-libs/cairo X glib opengl svg xcb" >> /etc/portage/package.use
```

5.5.2 Install

Simply unmask and emerge:

```
echo "x11-wm/qtile ~amd64" >> /etc/portage/package.keywords
emerge -av qtile
```

Don't forget the config.py:

```
mkdir ~/.config/qtile
cp build/lib/libqtile/resources/default_config.py ~/.config/qtile/config.py
```

where build is i.e

```
/usr/lib64/python2.7/site-packages
```

5.5.3 Test Installation

You can test your installation in Xephyr. If you don't have Xephyr you need to set the kdrive USE-Flag for xorg-server

```
echo "x11-base/xorg-server kdrive" >> /etc/portage/package.use
emerge -1 xorg-server
```

You can run Xephyr with

```
Xephyr :1 -screen 800x600 -av -noreset
```

In another term you set DISPLAY to :1

```
DISPLAY=:1
```

You start qtile simply with:

```
qtile
```

5.6 Installing from Source

Qtile relies on some cutting-edge features in PyCairo, XCB, and xpyb. Until the latest versions of these projects make it into distros, it's best to use recent checkouts from their repositories. You'll need python's `setuptools` installed. Here's a brief step-by-step guide:

5.6.1 xpyb

Either `xpyb-ng` or `xpyb` versions >= 1.3.1 should work. The `xpyb` build itself has historically had some package config issues, so we provide xpyb-ng for people who want to use setuptools. (The implementations are also slightly different, but users have reported that qtile is stable on either fork.) For users with a system version of `xcb-proto` < 1.7, xpyb will not build correctly (you get an `AttributeError: 'ListType' object has no attribute 'parent'`). However, xpyb-ng provides a branch called `pre-1.7-xproto` which has a hack to fix this issue.

```
git clone git://anongit.freedesktop.org/xcb/xpyb
cd xpyb && ./autogen.sh
./configure
make install

git clone git@github.com:tych0/xpyb-ng.git
cd xpyb-ng
python setup.py install
```

5.6.2 cairo

The latest cairo release works, but recompiling with xcb support is needed.

```
wget http://cairographics.org/releases/cairo-1.10.0.tar.gz
tar xvzf cairo-1.10.0.tar.gz
cd cairo-1.10.0
./autogen.sh --enable-xcb
make
sudo make install
```

5.6.3 py2cairo

```
git clone git://git.cairographics.org/git/py2cairo
cd py2cairo
./autogen.sh --enable-xcb
```

Check the configure output to make sure that XPYB is correctly detected.

```
make
sudo make install
```

5.6.4 PyGTK

We also require a reasonably recent version of the Python GTK bindings, in particular, the pango module. You should just be able to install this using your chosen distribution's package manager.

5.6.5 Qtile

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

5.7 Installing on Ubuntu

5.7.1 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), and 14.04 (Trusty Tahr).

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

5.7.2 Manual Installation Guides

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

5.8 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.

5.8.1 Configuration variables

groups A list of `libqtile.config.Group` objects which defines the group names. 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.

keys A list of `libqtile.config.Key` objects which defines the keybindings. At a minimum, this will probably include bindings to switch between windows, groups and layouts.

layouts A list of layout objects, configuring the layouts you want to use.

mouse A list of `libqtile.config.Drag` and `libqtile.config.Click` objects defining mouse operations.

screens A list of `libqtile.config.Screen` objects, which defines the physical screens you want to use, and the bars and widgets associated with them. Most of the visible "look and feel" configuration will happen in this section.

main() A function that executes after the window manager is initialized, but before groups, screens and other components are set up.

5.8.2 Putting it all together

The qtile-examples repository 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!)

5.8.3 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.

5.9 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.

5.9.1 Configuration Lookup

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.

5.9.2 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, a system tray, and a clock. you're using the default config.

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

5.10 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.

```python
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=gnome-settings-daemon;
RequiredProviders=windowmanager;notifications;
DefaultProvider-windowmanager=qtile
DefaultProvider-notifications=notification-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).

5.11 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.

5.11.1 Example

```python
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")
```

5.12 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.

Let's say we wanted to automatically float all dialog windows. 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.

5.13 Keys

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

5.13.1 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*.

5.13.2 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()
)
```

5.14 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.

A list of available layouts can be found in the *Built-in Layouts* reference.

5.14.1 Example

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

5.15 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.

5.15.1 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())
]
```

5.16 Screens

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

Please see *Widgets Reference* for a listing of built-in widgets.

5.16.1 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.

5.17 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.

5.18 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:

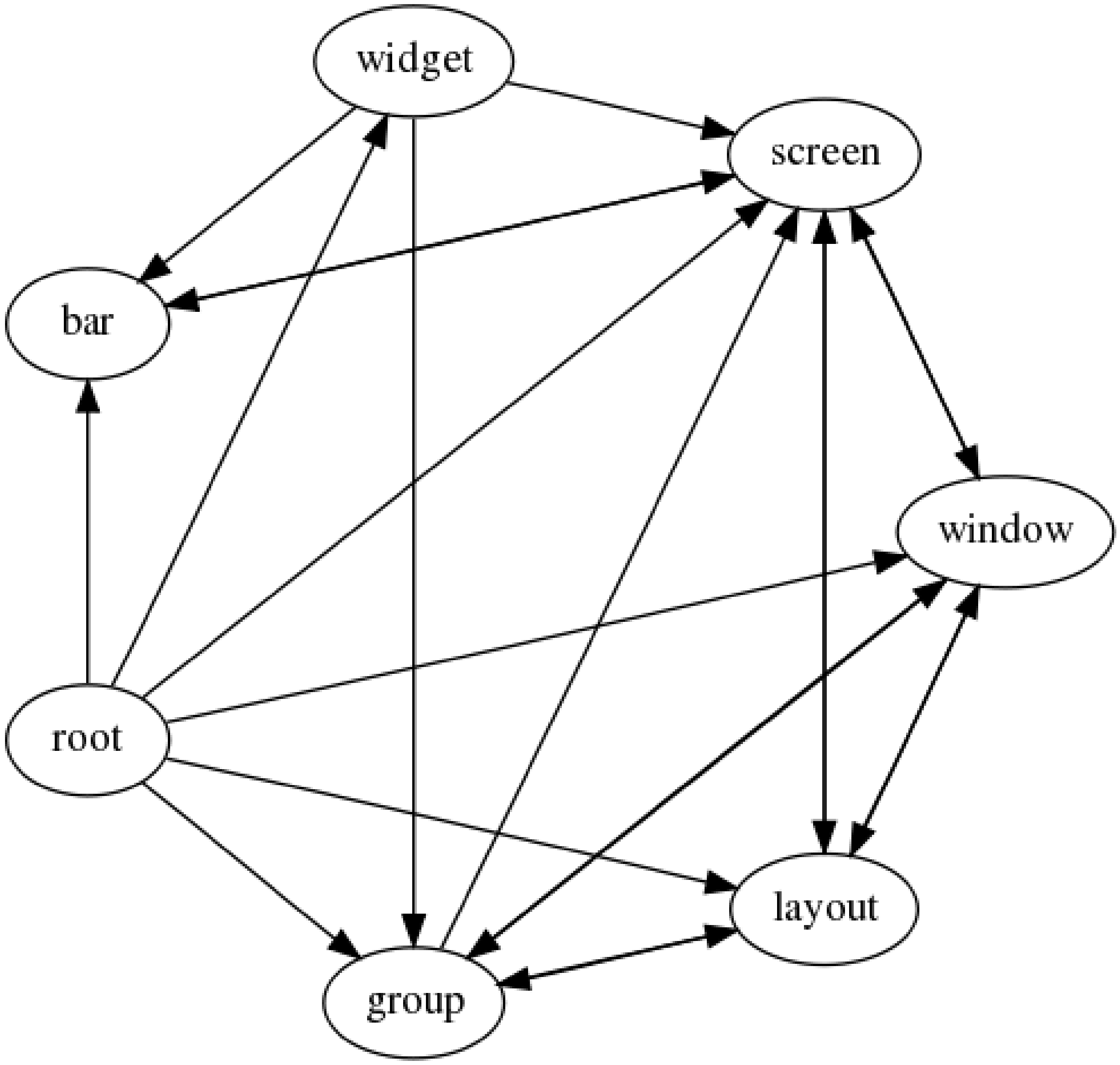
- Commands can be *bound to keys* in the Qtile configuration file.
- Commands can be *called through qsh*, the Qtile shell.

- 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.

5.19 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 exeption:

```
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()
```

5.20 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

5.21 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.

5.21.1 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]>
```

5.21.2 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.
```

5.22 Scripting

5.22.1 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*.

5.22.2 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"]
```

5.23 Qtile Reference

Built-in Hooks

Built-in Layouts

Built-in Widgets

5.24 Built-in Hooks

5.24.1 startup

Called when Qtile has initialized

5.24.2 client_name_updated

Called when the client name changes.

5.24.3 client_focus

Called whenver focus changes.

Arguments:

- `window.Window` object of the new focus.

5.24.4 addgroup

Called when group is added.

5.24.5 delgroup

Called when group is deleted.

5.24.6 group_window_add

Called when a new window is added to a group.

5.24.7 client_managed

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

5.24.8 client_new

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 of the newly created window

Example:

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

5.24.9 client_urgent_hint_changed

Called when the client urgent hint changes.

5.24.10 focus_change

Called when focus is changed.

5.24.11 float_change

Called when a change in float state is made

5.24.12 client_killed

Called after a client has been unmanaged.

Arguments:

- `window.Window` object of the killed window.

5.24.13 setgroup

Called when group is changed.

5.24.14 layout_change

Called on layout change.

5.24.15 client_state_changed

Called whenever client state changes.

5.24.16 window_name_change

Called whenever a windows name changes.

5.24.17 client_mouse_enter

Called when the mouse enters a client.

5.24.18 client_type_changed

Called whenever window type changes.

5.24.19 screen_change

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()
```

5.25 Built-in Layouts

5.25.1 Floating

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

5.25.2 Max

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.

5.25.3 MonadTall

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], "k", lazy.layout.down()),
Key([modkey], "j", lazy.layout.up()),
Key([modkey, "shift"], "k", lazy.layout.shuffle_down()),
Key([modkey, "shift"], "j", 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	`"monadtall"`	Name of this layout

5.25.4 RatioTile

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.

5.25.5 Slice

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.

5.25.6 Stack

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`	Autosplit all new stacks.
num_stacks	`2`	Number of stacks

5.25.7 Tile

<missing doc string>

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.

5.25.8 TreeTab

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.
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	`"max"`	Name of this layout.

5.25.9 Zoomy

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

5.26 Built-in Widgets

5.26.1 Applications

Canto

<missing doc string>

key	default	description
font	`"Arial"`	Font
fontsize	`None`	Pixel size. Calculated if None.
padding	`None`	Padding. Calculated if None.
background	`"000000"`	Background colour
foreground	`"ffffff"`	Foreground colour
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
update_delay	`600`	The delay in seconds between updates

Maildir

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

key	default	description
font	`"Arial"`	Font
fontsize	`None`	Maildir widget font size. Calculated if None.
padding	`None`	Maildir widget padding. Calculated if None.
background	`"000000"`	Background colour
foreground	`"ffffff"`	Foreground colour

Mpris

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.

key	default	description
font	`"Arial"`	Mpd widget font
fontsize	`None`	Mpd widget pixel size. Calculated if None.
padding	`None`	Mpd widget padding. Calculated if None.
background	`"000000"`	Background colour
foreground	`"ffffff"`	Foreground colour

YahooWeather

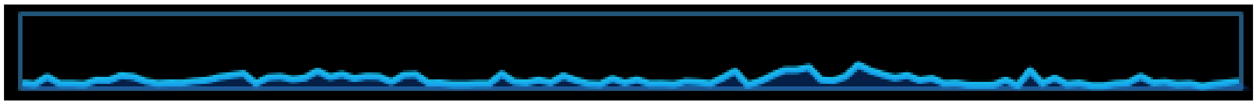
A weather widget, data provided by the Yahoo! Weather API

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
- wind_direction
- wind_speed

key	default	description
font	`"Arial"`	Font
fontsize	`None`	Pixel size, calculated if None.
padding	`None`	Padding, calculated if None.
background	`"000000"`	Background colour
foreground	`"ffffff"`	Foreground colour
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
update_interval	`600`	Update interval in seconds

5.26.2 Graphs



CPUGraph

<missing doc string>

key	default	description
graph_color	"18BAEB"	Graph color
fill_color	"1667EB.3"	Fill color for linefill graph
background	"000000"	Widget background
border_color	"215578"	Widget border color
border_width	2	Widget background
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')

HDDGraph

<missing doc string>

key	default	description
graph_color	"18BAEB"	Graph color
fill_color	"1667EB.3"	Fill color for linefill graph
background	"000000"	Widget background
border_color	"215578"	Widget border color
border_width	2	Widget background
margin_x	3	Margin X
margin_y	3	Margin Y
samples	100	Count of graph samples.
frequency	60	Update frequency in seconds
type	"linefill"	'box', 'line', 'linefill'
line_width	3	Line width
start_pos	"bottom"	Drawer starting position ('bottom'/'top')
path	"sda1"	Path at which parition is MOUNTED.
space_type	"used"	free/used

MemoryGraph

<missing doc string>

key	default	description
graph_color	"18BAEB"	Graph color
fill_color	"1667EB.3"	Fill color for linefill graph
background	"000000"	Widget background
border_color	"215578"	Widget border color
border_width	2	Widget background
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’)

NetGraph

<missing doc string>

key	default	description
graph_color	"18BAEB"	Graph color
fill_color	"1667EB.3"	Fill color for linefill graph
background	"000000"	Widget background
border_color	"215578"	Widget border color
border_width	2	Widget background
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
interface	"eth0"	Interface to display info for
bandwidth_type	"down"	down(load)/up(load)
start_pos	"bottom"	Drawer starting position (‘bottom’/’top’)

SwapGraph

<missing doc string>

key	default	description
graph_color	"18BAEB"	Graph color
fill_color	"1667EB.3"	Fill color for linefill graph
background	"000000"	Widget background
border_color	"215578"	Widget border color
border_width	2	Widget background
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’)

5.26.3 Misc

Notify

An notify widget

key	default	description
font	"Arial"	Mpd widget font
fontsize	None	Mpd widget pixel size. Calculated if None.
padding	None	Mpd widget padding. Calculated if None.
background	"000000"	Background colour
foreground	"ffffff"	Foreground normal priority colour
foreground_urgent	"ff0000"	Foreground urgent priority colour
foreground_low	"dddddd"	Foreground low priority colour

Prompt

A widget that prompts for user input. Input should be started using the .startInput method on this class.

key	default	description
font	"Arial"	Font
fontsize	None	Font pixel size. Calculated if None.
padding	None	Padding. Calculated if None.
background	"000000"	Background colour
foreground	"ffffff"	Foreground colour
cursorblink	0.5	Cursor blink rate. 0 to disable.

Sep

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.
background	"000000"	Background colour.
height_percent	80	Height as a percentage of bar height (0-100).

Spacer

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.

5.26.4 System

Battery

A simple but flexible text-based battery widget.

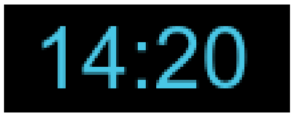
key	default	description
low_foreground	"FF0000"	font color when battery is low
format	"{char} {percent:2.0%} {hour:d}:{min:02d}"	Display format
charge_char	"^"	Character to indicate the battery is charging
discharge_char	"V"	Character to indicate the battery is discharging
font	"Arial"	Text font
fontsize	None	Font pixel size. Calculated if None.
padding	3	Padding left and right. Calculated if None.
background	None	Background colour.
foreground	"#ffffff"	Foreground colour.
battery_name	"BAT0"	ACPI name of a battery, usually BAT0
status_file	"status"	Name of status file in /sys/class/power_supply/battery_name
energy_now_file	"energy_now"	Name of file with the current energy in /sys/class/power_supply/battery_name
energy_full_file	"energy_full"	Name of file with the maximum energy in /sys/class/power_supply/battery_name
power_now_file	"power_now"	Name of file with the current power draw in /sys/class/power_supply/battery_name
update_delay	1	The delay in seconds between updates

BatteryIcon

Battery life indicator widget

key	default	description
theme_path	"libqtile/resources/battery-icons"	Path of the icons
custom_icons	{}	dict containing key->filename icon map
font	"Arial"	Text font
fontsize	None	Font pixel size. Calculated if None.
padding	3	Padding left and right. Calculated if None.
background	None	Background colour.
foreground	"#ffffff"	Foreground colour.
battery_name	"BAT0"	ACPI name of a battery, usually BAT0
status_file	"status"	Name of status file in /sys/class/power_supply/battery_name
energy_now_file	"energy_now"	Name of file with the current energy in /sys/class/power_supply/battery_name
energy_full_file	"energy_full"	Name of file with the maximum energy in /sys/class/power_supply/battery_name
power_now_file	"power_now"	Name of file with the current power draw in /sys/class/power_supply/battery_name
update_delay	1	The delay in seconds between updates

Clock

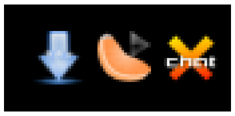


A simple but flexible text-based clock.

key	default	description
font	`"Arial"`	Clock font
fontsize	`None`	Clock pixel size. Calculated if None.
padding	`None`	Clock padding. Calculated if None.
background	`"000000"`	Background colour
foreground	`"ffffff"`	Foreground colour

Systray

A widget that manage system tray



key	default	description
icon_size	`20`	Icon width
padding	`5`	Padding between icons
background	`None`	Background colour

Volume

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
font	`"Arial"`	Text font
fontsize	`None`	Font pixel size. Calculated if None.
padding	`3`	Padding left and right. Calculated if None.
background	`None`	Background colour.
foreground	`"#ffffff"`	Foreground colour.
theme_path	`None`	Path of the icons
update_interval	`0.2`	Update time in seconds.

5.26.5 Window Management

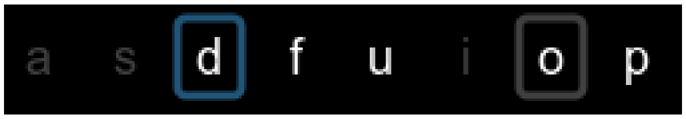
CurrentLayout

<missing doc string>

key	default	description
font	"Arial"	Text font
fontsize	None	Font pixel size. Calculated if None.
padding	None	Padding left and right. Calculated if None.
background	None	Background colour.
foreground	"#ffffff"	Foreground colour.

GroupBox

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
margin_y	3	Y margin outside the box
margin_x	3	X margin outside the box
borderwidth	3	Current group border width
font	"Arial"	Font face
fontsize	None	Font pixel size - calculated if None
background	"000000"	Widget background
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.
this_screen_border	"113358"	Border colour for group on this screen.
other_screen_border	"404040"	Border colour for group on other screen.
padding	5	Padding inside the box
urgent_border	"FF0000"	Urgent border color
urgent_alert_method	"border"	Method for alerting you of WM urgent hints (one of 'border' or 'text')

WindowName

Displays the name of the window that currently has focus.

key	default	description
font	"Arial"	Font face.
fontsize	None	Font pixel size. Calculated if None.
padding	None	Padding left and right.
background	"000000"	Background colour.
foreground	"ffffff"	Foreground colour.

5.27 Frequently Asked Questions

5.27.1 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.

5.27.2 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)
```

5.27.3 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.

5.28 Hacking Qtile

5.28.1 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`.

5.28.2 Running the test suite

```
$ cd test
$ nosetests
```

Note: nose runs the tests against the first version of qtile it finds on your `$PYTHONPATH`; for most people this is the currently installed version of qtile.

5.28.3 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

5.28.4 Second X Session

Some users prefer to test Qtile in it's own brand new X session. If you'd like to run a second X session, you can switch to a new tty and start a new one with `xinit second_xsession`, where `second_xsession` is a file invoking your development version of qtile (and doing any other setup you want). Examples of custom `xsession` files are available in qtile-examples.

5.28.5 Contributing to Qtile

Now you've got a patch you want to submit to be merged to Qtile. Typically, this is done via github pull requests. Qtile uses a well known branching model; master is our current release, and the `develop` branch is what all pull requests should be based against.

While not all of our code follows PEP8, we do try to adhere to it where possible, and ideally any new code would be PEP8 compliant. Perhaps the biggest issue is tabs vs. spaces: only 4 space tabs, please. We also request that git commit messages follow the standard format.

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

5.28.6 Reporting Bugs

Please report any bugs you find to the github issue tracker.

5.28.7 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

5.29 License

This project is distributed under the MIT license.

5.30 Release Notes

Release notes for official Qtile releases.

- *Qtile 0.6 release notes*
- *Qtile 0.5 release notes*
- *Qtile 0.7.0 release notes*

5.31 Qtile 0.5 release notes

November 11th, 2012

Download qtile-0.5.tar.gz

This marks the first officially community managed release.

5.31.1 Changelog

Features

- Test framework is now nose
- Documentation is now in sphinx
- Several install guides for various OSes
- New widgets: battery based icon, MPRIS1, canto, current layout, yahoo weather, sensors, screen brightness, notifiy, pacman, windowtabs, she, crashme, wifi.
- Several improvements to old widgets (e.g. battery widget displays low battery in red, GroupBox now has a better indication of which screen has focus in multi-screen setups, improvements to Prompt, etc.)
- Desktop notification service.
- More sane way to handle configuration files
- Promote dgroups to a first class entity in libqtile
- Allow layouts to be named on an instance level, so you can:

```
layouts = [
    # a layout just for gimp
    layout.Slice('left', 192, name='gimp', role='gimp*toolbox',
        fallback=layout.Slice('right', 256, role='gimp*dock',
        fallback=layout.Stack(stacks=1, **border_args)))
]
...

dynamic_groups = { 'gimp': {'layout': 'gimp'} }

Dgroups(..., dynamic_groups, ...)
```

- New Layout: Zoomy
- Add a session manager to re-exec qtile if things go south
- Support for WM_TAKE_FOCUS protocol
- Basic .desktop file for support in login managers
- Qsh reconnects after qtile is restarted from within it
- Textbox supports pango markup
- Examples moved to qtile-examples repository.

Bug fixes

- Fix several classes of X races in a more sane way
- Minor typo fixes to most widgets
- Fix several crashes when drawing systray icons too early
- Create directories for qtile socket as necessary
- PEP8 formatting updates (though we're not totally there yet)
- All unit tests pass

- Lots of bugfixes to MonadTall
- Create IPC socket directory if necessary
- Better error if two widgets have STRETCH width
- Autofloat window classes can now be overridden
- xkeysyms updated

5.32 Qtile 0.6 release notes

May 11th, 2013

Download qtile-0.6.tar.gz

5.32.1 Config Breakage!

This release breaks your config file in several ways:

- The Textbox widget no longer takes a `name` positional parameter, since it was redundant; you can use the `name` kwarg to define it.
- manager.Group (now _Group) is not used to configure groups any more; config.Group replaces it. For simple configurations (i.e. Group("a") type configs), this should be a drop in replacement. config.Group also provides many more options for showing and hiding groups, assigning windows to groups by default, etc.
- The Key, Screen, Drag, and Click objects have moved from the manager module to the config module.
- The Match object has moved from the dgroups module to the config module.
- The addgroup hook now takes two parameters: the qtile object and the name of the group added:

  ```
  @hook.subscribe
  def addgroup_hook(qtile, name):
      pass
  ```

- The nextgroup and prevgroup commands are now on Screen instead of Group.

For most people with basic configs, you should be able to just:

```
sed -i -e 's/libqtile.manager/libqtile.config/' config.py
```

...dgroups users will need to go to a bit more work, but hopefully configuration will be much simpler now for new users.

5.32.2 Changelog

Features

- New widgets: task list,
- New layout: Matrix
- Added ability to drag and drop groups on GroupBox
- added "next urgent window" command
- added font shadowing on widgets

- maildir widget supports multiple folders
- new config option log_level to set logging level (any of logging.{DEBUG, INFO, WARNING, ERROR, CRITICAL})
- add option to battery widget to hide while level is above a certain amount
- vastly simplify configuration of dynamic groups
- MPD widget now supports lots of metadata options

Bug fixes

- don't crash on restart when the config has errors
- save layout and selected group state on restart
- varous EWMH properties implemented correctly
- fix non-black systray icon backgrounds
- drastically reduce the number of timeout_add calls in most widgets
- restart on RandR attach events to allow for new screens
- log level defaults to ERROR
- default config options are no longer initialized when users define their corresponding option (preventing duplicate widgets, etc.)
- don't try to load config in qsh (not used)
- fix font alignment across Textbox based widgets

5.33 Qtile 0.7.0 release notes

March 30, 2014

5.33.1 Changelog

Features

- new disk free percentage widget
- new widget to display static image
- per core CPU graphs
- add "screen affinity" in dynamic groups
- volume widget changes volume linear-ly instead of log-ly
- only draw bar when idle, vastly reducing the number of bar draws and speeding things up
- new Gmail widget
- Tile now supports automatically managing master windows via the `master_match` parameter.
- include support for minimum height, width, size increment hints

Bug fixes

- don't crash on any exception in main loop
- don't crash on exceptions in hooks
- fix a ZeroDivisionError in CPU graph
- remove a lot of duplicate and unused code
- Steam windows are placed more correctly
- Fixed several crashes in qsh
- performance improvements for some layouts
- keyboard layout widget behaves better with multiple keyboard configurations

Config Breakage!

- Tile's shuffleMatch is renamed to resetMaster