i3

i3 User's Guide

Michael Stapelberg

<michael@i3wm.org>

Table of Contents

This document contains all the information you need to configure and use the i3 window manager. If it does not, please check https://www.reddit.com/r/i3wm/ first, then contact us on IRC (preferred) or post your question(s) on the mailing list.

1. Default keybindings

For the "too long; didn't read" people, here is an overview of the default keybindings (click to see the full-size image):

Keys to use with $mod (Alt):

Keys to use with Shift+$mod:

The red keys are the modifiers you need to press (by default), the blue keys are your homerow.
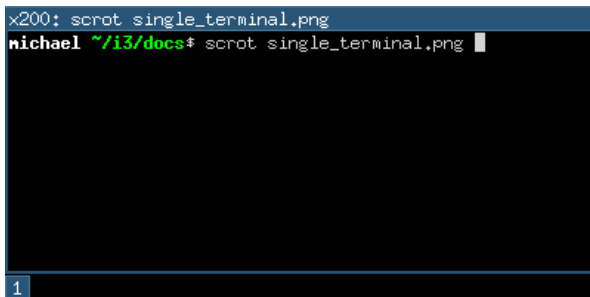
Note that when starting i3 without a config file, i3-config-wizard will offer you to create a config file in which the key positions (!) match what you see in the image above, regardless of the keyboard layout you are using. If you prefer to use a config file where the key letters match what you are seeing above, just decline i3-config-wizard's offer and base your config on `/etc/i3/config`.

2. Using i3

Throughout this guide, the keyword `$mod` will be used to refer to the configured modifier. This is the Alt key (`Mod1`) by default, with the Windows key (`Mod4`) being a popular alternative that largely prevents conflicts with application-defined shortcuts.
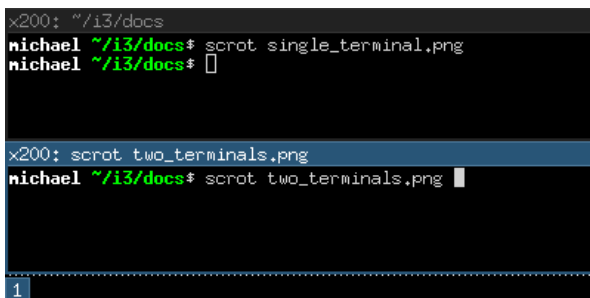
2.1. Opening terminals and moving around

One very basic operation is opening a new terminal. By default, the keybinding for this is `$mod+Enter`, that is Alt+Enter (`Mod1+Enter`) in the default configuration. By pressing `$mod+Enter`, a new terminal will be opened. It will fill the whole space available on your screen.

If you now open another terminal, i3 will place it next to the current one, splitting the screen size in half. Depending on your monitor, i3 will put the created window beside the existing window (on wide displays) or below the existing window (rotated displays).

To move the focus between the two terminals, you can use the direction keys which you might know from the editor `vi`. However, in i3, your homerow is used for these keys (in `vi`, the keys are shifted to the left by one for compatibility with most keyboard layouts). Therefore, `$mod+j` is left, `$mod+k` is down, `$mod+l` is up and `$mod+;` is right. So, to switch between the terminals, use `$mod+k` or `$mod+l`. Of course, you can also use the arrow keys.

At the moment, your workspace is split (it contains two terminals) in a specific direction (horizontal by default). Every window can be split horizontally or vertically again, just like the workspace. The terminology is "window" for a container that actually contains an X11 window (like a terminal or browser) and "split container" for containers that consist of one or more windows.

TODO: picture of the tree

To split a window vertically, press $mod+v before you create the new window. To split it horizontally, press $mod+h.

2.2. Changing the container layout

A split container can have one of the following layouts:

splith/splitv

Windows are sized so that every window gets an equal amount of space in the container. splith distributes the windows horizontally (windows are right next to each other), splitv distributes them vertically (windows are on top of each other).

stacking

Only the focused window in the container is displayed. You get a list of windows at the top of the container.

tabbed

The same principle as `stacking`, but the list of windows at the top is only a single line which is vertically split.

To switch modes, press $mod+e for splith/splitv (it toggles), $mod+s for stacking and $mod+w for tabbed.

2.3. Toggling fullscreen mode for a window

To display a window in fullscreen mode or to go out of fullscreen mode again, press $mod+f.

There is also a global fullscreen mode in i3 in which the client will span all available outputs (the command is `fullscreen toggle global`).

2.4. Opening other applications

Aside from opening applications from a terminal, you can also use the handy dmenu which is opened by pressing $mod+d by default. Just type the name (or a part of it) of the application which you want to open. The corresponding application has to be in your $PATH for this to work.

Additionally, if you have applications you open very frequently, you can create a keybinding for starting the application directly. See the section [configuring] for details.

2.5. Closing windows

If an application does not provide a mechanism for closing (most applications provide a menu, the escape key or a shortcut like `Control+w` to close), you can press $mod+Shift+q to kill a window. For applications which support the WM_DELETE protocol, this will correctly close the application (saving any modifications or doing other cleanup). If the application doesn't support the WM_DELETE protocol your X server will kill the window and the behaviour depends on the application.

2.6. Using workspaces

Workspaces are an easy way to group a set of windows. By default, you are on the first workspace, as the bar on the bottom left indicates. To switch to another workspace, press $mod+num where num is the number of the workspace you want to use. If the workspace does not exist yet, it will be created.

A common paradigm is to put the web browser on one workspace, communication applications (`mutt`, `irssi`, ...) on another one, and the ones with which you work, on the third one. Of course, there is no need to follow this approach.

If you have multiple screens, a workspace will be created on each screen at startup. If you open a new workspace, it will be bound to the screen you created it on. When you switch to a workspace on another screen, i3 will set focus to that screen.

2.7. Moving windows to workspaces

To move a window to another workspace, simply press `$mod+Shift+num` where num is (like when switching workspaces) the number of the target workspace. Similarly to switching workspaces, the target workspace will be created if it does not yet exist.

2.8. Resizing

The easiest way to resize a container is by using the mouse: Grab the border and move it to the wanted size.

You can also use [binding_modes] to define a mode for resizing via the keyboard. To see an example for this, look at the default config provided by i3.

2.9. Restarting i3 inplace

To restart i3 in place (and thus get into a clean state if there is a bug, or to upgrade to a newer version of i3) you can use `$mod+Shift+r`.

2.10. Exiting i3

To cleanly exit i3 without killing your X server, you can use `$mod+Shift+e`. By default, a dialog will ask you to confirm if you really want to quit.

2.11. Floating

Floating mode is the opposite of tiling mode. The position and size of a window are not managed automatically by i3, but manually by you. Using this mode violates the tiling paradigm but can be useful for some corner cases like "Save as" dialog windows, or toolbar windows (GIMP or similar). Those windows usually set the appropriate hint and are opened in floating mode by default.

You can toggle floating mode for a window by pressing `$mod+Shift+Space`. By dragging the window's titlebar with your mouse you can move the window around. By grabbing the borders and moving them you can resize the window. You can also do that by using the [floating_modifier]. Another way to resize floating windows using the mouse is to right-click on the titlebar and drag.

For resizing floating windows with your keyboard, see the resizing binding mode provided by the i3 default config.

Floating windows are always on top of tiling windows.

3. Tree

i3 stores all information about the X11 outputs, workspaces and layout of the windows on them in a tree. The root node is the X11 root window, followed by the X11 outputs, then dock areas and a content container, then workspaces and finally the windows themselves. In previous versions of i3 we had multiple lists (of outputs, workspaces) and a table for each workspace. That approach turned out to be complicated to use (snapping), understand and implement.

3.1. The tree consists of Containers

The building blocks of our tree are so-called `Containers`. A `Container` can host a window (meaning an X11 window, one that you can actually see and use, like a browser). Alternatively, it could contain one or more `Containers`. A simple example is the workspace: When you start i3 with a single monitor, a single workspace and you open two terminal windows, you will end up with a tree like this:
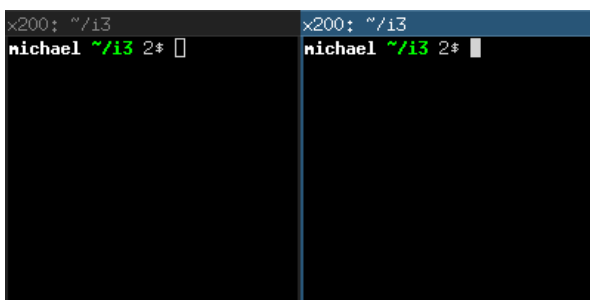
Figure 1. Two terminals on standard workspace
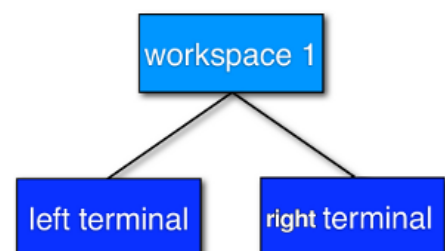
3.2. Orientation and Split Containers

It is only natural to use so-called `Split Containers` in order to build a layout when using a tree as data structure. In i3, every `Container` has an orientation (horizontal, vertical or unspecified) and the orientation depends on the layout the container is in (vertical for splitv and stacking, horizontal for splith and tabbed). So, in our example with the workspace, the default layout of the workspace `Container` is splith (most monitors are widescreen nowadays). If you change the layout to splitv (`$mod+v` in the default config) and **then** open two terminals, i3 will configure your windows like this:
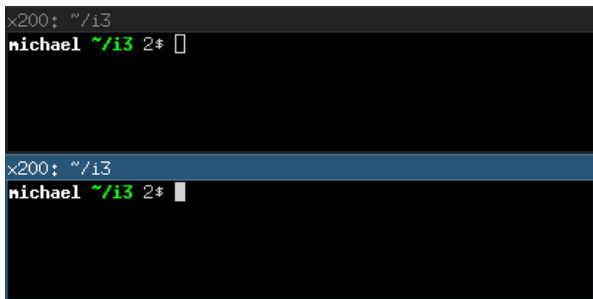
Figure 2. Vertical Workspace Orientation

An interesting new feature of i3 since version 4 is the ability to split anything: Let's assume you have two terminals on a workspace (with splith layout, that is horizontal orientation), focus is on the right terminal. Now you want to open another terminal window below the current one. If you would just open a new terminal window, it would show up to the right due to the splith layout. Instead, press `$mod+v` to split the container with the splitv layout (to open a `Horizontal Split Container`, use `$mod+h`). Now you can open a new terminal and it will open below the current one:

Figure 3. Vertical Split Container
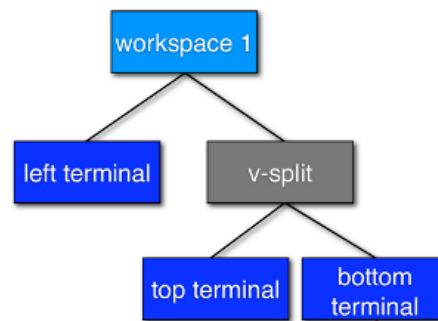
You probably guessed it already: There is no limit on how deep your hierarchy of splits can be.

3.3. Focus parent

Let's stay with our example from above. We have a terminal on the left and two vertically split terminals on the right, focus is on the bottom right one. When you open a new terminal, it will open below the current one.

So, how can you open a new terminal window to the **right** of the current one? The solution is to use `focus parent`, which will focus the `Parent Container` of the current `Container`. In default configuration, use `$mod+a` to navigate one `Container` up the tree (you can repeat this multiple times until you get to the `Workspace Container`). In this case, you would focus the `Vertical Split Container` which is **inside** the horizontally oriented workspace. Thus, now new windows will be opened to the right of the `Vertical Split Container`:
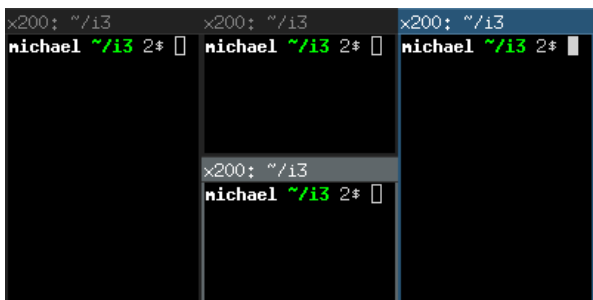
Figure 4. Focus parent, then open new terminal

3.4. Implicit containers

In some cases, i3 needs to implicitly create a container to fulfill your command.

One example is the following scenario: You start i3 with a single monitor and a single workspace on which you open three terminal windows. All these terminal windows are directly attached to one node inside i3's layout tree, the workspace node. By default, the workspace node's orientation is `horizontal`.

Now you move one of these terminals down (`$mod+Shift+j` by default). The workspace node's orientation will be changed to `vertical`. The terminal window you moved down is directly attached to the workspace and appears on the bottom of the screen. A new (horizontal) container was created to accommodate the other two terminal windows. You will notice this when switching to tabbed mode (for example). You would end up having one tab with a representation of the split container (e.g., "H[urxvt firefox]") and the other one being the terminal window you moved down.

4. Configuring i3

This is where the real fun begins ;-). Most things are very dependent on your ideal working environment so we can't make reasonable defaults for them.

While not using a programming language for the configuration, i3 stays quite flexible in regards to the things you usually want your window manager to do.

For example, you can configure bindings to jump to specific windows, you can set specific applications to start on specific workspaces, you can automatically start applications, you can change the colors of i3, and you can bind your keys to do useful things.

To change the configuration of i3, copy `/etc/i3/config` to `~/.i3/config` (or `~/.config/i3/config` if you like the XDG directory scheme) and edit it with a text editor.

On first start (and on all following starts, unless you have a configuration file), i3 will offer you to create a configuration file. You can tell the wizard to use either Alt (`Mod1`) or Windows (`Mod4`) as modifier in the config file. Also, the created config file will use the key symbols of your current keyboard layout. To start the wizard, use the command `i3-config-wizard`. Please note that you must not have `~/.i3/config`, otherwise the wizard will exit.

Since i3 4.0, a new configuration format is used. i3 will try to automatically detect the format version of a config file based on a few different keywords, but if you want to make sure that your config is read with the new format, include the following line in your config file:

```
# i3 config file (v4)
```

4.1. Comments

It is possible and recommended to use comments in your configuration file to properly document your setup for later reference. Comments are started with a # and can only be used at the beginning of a line:

Examples:

```
# This is a comment
```

4.2. Fonts

i3 has support for both X core fonts and FreeType fonts (through Pango) to render window titles.

To generate an X core font description, you can use `xfontsel(1)`. To see special characters (Unicode), you need to use a font which supports the ISO-10646 encoding.

A FreeType font description is composed by a font family, a style, a weight, a variant, a stretch and a size. FreeType fonts support right-to-left rendering and contain often more Unicode glyphs than X core fonts.

If i3 cannot open the configured font, it will output an error in the logfile and fall back to a working font.

Syntax:

```
font <X core font description>
font pango:<family list> [<style options>] <size>
```

Examples:

```
font -misc-fixed-medium-r-normal--13-120-75-75-C-70-iso10646-1
font pango:DejaVu Sans Mono 10
font pango:DejaVu Sans Mono, Terminus Bold Semi-Condensed 11
font pango:Terminus 11px
```

4.3. Keyboard bindings

A keyboard binding makes i3 execute a command (see below) upon pressing a specific key. i3 allows you to bind either on keycodes or on keysyms (you can also mix your bindings, though i3 will not protect you from overlapping ones).

- A keysym (key symbol) is a description for a specific symbol, like "a" or "b", but also more strange ones like "underscore" instead of "_". These are the ones you use in Xmodmap to remap your keys. To get the current mapping of your keys, use xmodmap -pke. To interactively enter a key and see what keysym it is configured to, use xev.

- Keycodes do not need to have a symbol assigned (handy for custom vendor hotkeys on some notebooks) and they will not change their meaning as you switch to a different keyboard layout (when using xmodmap).

My recommendation is: If you often switch keyboard layouts but you want to keep your bindings in the same physical location on the keyboard, use keycodes. If you don't switch layouts, and want a clean and simple config file, use keysyms.

Some tools (such as import or xdotool) might be unable to run upon a KeyPress event, because the keyboard/pointer is still grabbed. For these situations, the --release flag can be used, which will execute the command after the keys have been released.

Syntax:

```
bindsym [--release] [<Group>+][<Modifiers>+]<keysym> command
bindcode [--release] [<Group>+][<Modifiers>+]<keycode> command
```

Examples:

```
# Fullscreen
bindsym $mod+f fullscreen toggle

# Restart
bindsym $mod+Shift+r restart

# Notebook-specific hotkeys
bindcode 214 exec --no-startup-id /home/michael/toggle_beamer.sh

# Simulate ctrl+v upon pressing $mod+x
bindsym --release $mod+x exec --no-startup-id xdotool key --clearmodifiers ctrl+v

# Take a screenshot upon pressing $mod+x (select an area)
bindsym --release $mod+x exec --no-startup-id import /tmp/latest-screenshot.png
```

Available Modifiers:

Mod1-Mod5, Shift, Control
Standard modifiers, see xmodmap(1)

Group1, Group2, Group3, Group4
When using multiple keyboard layouts (e.g. with setxkbmap -layout us,ru), you can specify in which XKB group (also called "layout") a keybinding should be active. By default, keybindings are translated in Group1 and are active in all groups. If you want to override keybindings in one of your layouts, specify the corresponding group. For backwards compatibility, the group "Mode_switch" is an alias for Group2.

4.4. Mouse bindings

A mouse binding makes i3 execute a command upon pressing a specific mouse button in the scope of the clicked container (see [command_criteria]). You can configure mouse bindings in a similar way to key bindings.

Syntax:

```
bindsym [--release] [--border] [--whole-window] [--exclude-titlebar] [<Modifiers>+]button<n> co
```

By default, the binding will only run when you click on the titlebar of the window. If the `--release` flag is given, it will run when the mouse button is released.

If the `--whole-window` flag is given, the binding will also run when any part of the window is clicked, with the exception of the border. To have a bind run when the border is clicked, specify the `--border` flag.

If the `--exclude-titlebar` flag is given, the titlebar will not be considered for the keybinding.

Examples:

```
# The middle button over a titlebar kills the window
bindsym --release button2 kill

# The middle button and a modifer over any part of the window kills the window
bindsym --whole-window $mod+button2 kill

# The right button toggles floating
bindsym button3 floating toggle
bindsym $mod+button3 floating toggle

# The side buttons move the window around
bindsym button9 move left
bindsym button8 move right
```

4.5. Binding modes

You can have multiple sets of bindings by using different binding modes. When you switch to another binding mode, all bindings from the current mode are released and only the bindings defined in the new mode are valid for as long as you stay in that binding mode. The only predefined binding mode is `default`, which is the mode i3 starts out with and to which all bindings not defined in a specific binding mode belong.

Working with binding modes consists of two parts: defining a binding mode and switching to it. For these purposes, there are one config directive and one command, both of which are called mode. The directive is used to define the bindings belonging to a certain binding mode, while the command will switch to the specified mode.

It is recommended to use binding modes in combination with [variables] in order to make maintenance easier. Below is an example of how to use a binding mode.

Note that it is advisable to define bindings for switching back to the default mode.

Note that it is possible to use [pango_markup] for binding modes, but you need to enable it explicitly by passing the `--pango_markup` flag to the mode definition.

Syntax:

```
# config directive
mode [--pango_markup] <name>

# command
mode <name>
```

Example:

```
# Press $mod+o followed by either f, t, Escape or Return to launch firefox,
# thunderbird or return to the default mode, respectively.
set $mode_launcher Launch: [f]irefox [t]hunderbird
bindsym $mod+o mode "$mode_launcher"

mode "$mode_launcher" {
    bindsym f exec firefox
    bindsym t exec thunderbird

    bindsym Escape mode "default"
    bindsym Return mode "default"
}
```

4.6. The floating modifier

To move floating windows with your mouse, you can either grab their titlebar or configure the so-called floating modifier which you can then press and click anywhere in the window itself to move it. The most common setup is to use the same key you use for managing windows (Mod1 for example). Then you can press Mod1, click into a window using your left mouse button, and drag it to the position you want.

When holding the floating modifier, you can resize a floating window by pressing the right mouse button on it and moving around while holding it. If you hold the shift button as well, the resize will be proportional (the aspect ratio will be preserved).

Syntax:

```
floating_modifier <Modifier>
```

Example:

```
floating_modifier Mod1
```

4.7. Constraining floating window size

The maximum and minimum dimensions of floating windows can be specified. If either dimension of `floating_maximum_size` is specified as -1, that dimension will be unconstrained with respect to its maximum value. If either dimension of `floating_maximum_size` is undefined, or specified as 0, i3 will use a default value to constrain the maximum size. `floating_minimum_size` is treated in a manner analogous to `floating_maximum_size`.

Syntax:

```
floating_minimum_size <width> x <height>
floating_maximum_size <width> x <height>
```

Example:

```
floating_minimum_size 75 x 50
floating_maximum_size -1 x -1
```

4.8. Orientation for new workspaces

New workspaces get a reasonable default orientation: Wide-screen monitors (anything wider than high) get horizontal orientation, rotated monitors (anything higher than wide) get vertical orientation.

With the `default_orientation` configuration directive, you can override that behavior.

Syntax:

```
default_orientation horizontal|vertical|auto
```

Example:

```
default_orientation vertical
```

4.9. Layout mode for new containers

This option determines in which mode new containers on workspace level will start.

Syntax:

```
workspace_layout default|stacking|tabbed
```

Example:

```
workspace_layout tabbed
```

4.10. Window title alignment

This option determines the window title's text alignment. Default is `left`

Syntax:

```
title_align left|center|right
```

4.11. Default border style for new windows

This option determines which border style new windows will have. The default is `normal`. Note that default_floating_border applies only to windows which are starting out as floating windows, e.g., dialog windows, but not windows that are floated later on.

Setting border style to `pixel` eliminates title bars. The border style `normal` allows you to adjust edge border width while keeping your title bar.

Syntax:

```
default_border normal|none|pixel
default_border normal|pixel <px>
default_floating_border normal|none|pixel
default_floating_border normal|pixel <px>
```

Please note that new_window and new_float have been deprecated in favor of the above options and will be removed in a future release. We strongly recommend using the new options instead.

Example:

```
default_border pixel
```

The "normal" and "pixel" border styles support an optional border width in pixels:

Example:

```
# The same as default_border none
default_border pixel 0

# A 3 px border
default_border pixel 3
```

4.12. Hiding borders adjacent to the screen edges

You can hide container borders adjacent to the screen edges using `hide_edge_borders`. This is useful if you are using scrollbars, or do not want to waste even two pixels in displayspace. The "smart" setting hides borders on workspaces with only one window visible, but keeps them on workspaces with multiple windows visible. Default is none.

Syntax:

```
hide_edge_borders none|vertical|horizontal|both|smart
```

Example:

```
hide_edge_borders vertical
```

4.13. Arbitrary commands for specific windows (for_window)

With the `for_window` directive, you can let i3 execute any command when it encounters a specific window. This can be used to set windows to floating or to change their border style, for example.

Syntax:

```
for_window <criteria> <command>
```

Examples:

```
# enable floating mode for all XTerm windows
for_window [class="XTerm"] floating enable

# Make all urxvts use a 1-pixel border:
for_window [class="urxvt"] border pixel 1

# A less useful, but rather funny example:
# makes the window floating as soon as I change
# directory to ~/work
for_window [title="x200: ~/work"] floating enable
```

The valid criteria are the same as those for commands, see [command_criteria]. Only commands can be executed at runtime, not config directives, see [list_of_commands].

4.14. Don't focus window upon opening

When a new window appears, it will be focused. The `no_focus` directive allows preventing this from happening and must be used in combination with [command_criteria].

Note that this does not apply to all cases, e.g., when feeding data into a running application causing it to request being focused. To configure the behavior in such cases, refer to [focus_on_window_activation].

`no_focus` will also be ignored for the first window on a workspace as there shouldn't be a reason to not focus the window in this case. This allows for better usability in combination with `workspace_layout`.

Syntax:

```
no_focus <criteria>
```

Example:

```
no_focus [window_role="pop-up"]
```

4.15. Variables

As you learned in the section about keyboard bindings, you will have to configure lots of bindings containing modifier keys. If you want to save yourself some typing and be able to change the modifier you use later, variables can be handy.

Syntax:

```
set $<name> <value>
```

Example:

```
set $m Mod1
bindsym $m+Shift+r restart
```

Variables are directly replaced in the file when parsing. Variables expansion is not recursive so it is not possible to define a variable with a value containing another variable. There is no fancy handling and there are absolutely no plans to change this. If you need a more dynamic configuration you should create a little script which generates a configuration file and run it before starting i3 (for example in your `~/.xsession` file).

Also see [xresources] to learn how to create variables based on resources loaded from the X resource database.

4.16. X resources

[variables] can also be created using a value configured in the X resource database. This is useful, for example, to avoid configuring color values within the i3 configuration. Instead, the values can be configured, once, in the X resource database to achieve an easily maintainable, consistent color theme across many X applications.

Defining a resource will load this resource from the resource database and assign its value to the specified variable. This is done verbatim and the value must therefore be in the format that i3 uses. A fallback must be specified in case the resource cannot be loaded from the database.

Syntax:

```
set_from_resource $<name> <resource_name> <fallback>
```

Example:

```
# The ~/.Xresources should contain a line such as
#     *color0: #121212
# and must be loaded properly, e.g., by using
#     xrdb ~/.Xresources
# This value is picked up on by other applications (e.g., the URxvt terminal
# emulator) and can be used in i3 like this:
set_from_resource $black i3wm.color0 #000000
```

4.17. Automatically putting clients on specific workspaces

To automatically make a specific window show up on a specific workspace, you can use an **assignment**. You can match windows by using any criteria, see [command criteria]. The difference between `assign` and `for_window <criteria> move to workspace` is that the former will only be executed when the application maps the window (mapping means actually displaying it on the screen) but the latter will be executed whenever a window changes its properties to something that matches the specified criteria.

Thus, it is recommended that you match on window classes (and instances, when appropriate) instead of window titles whenever possible because some applications first create their window, and then worry about setting the correct title. Firefox with Vimperator comes to mind. The window starts up being named Firefox, and only when Vimperator is loaded does the title change. As i3 will get the title as soon as the application maps the window, you'd need to have to match on *Firefox* in this case. Another known issue is with Spotify, which doesn't set the class hints when mapping the window, meaning you'll have to use a `for_window` rule to assign Spotify to a specific workspace. Finally, using `assign [tiling]` and `assign [floating]` is not supported.

You can also assign a window to show up on a specific output. You can use RandR names such as `VGA1` or names relative to the output with the currently focused workspace such as `left` and `down`.

Assignments are processed by i3 in the order in which they appear in the config file. The first one which matches the window wins and later assignments are not considered.

Syntax:

```
assign <criteria> [→] [workspace] [number] <workspace>
assign <criteria> [→] output left|right|up|down|primary|<output>
```

Examples:

```
# Assign URxvt terminals to workspace 2
assign [class="URxvt"] 2

# Same thing, but more precise (exact match instead of substring)
assign [class="^URxvt$"] 2

# Same thing, but with a beautiful arrow :)
assign [class="^URxvt$"] → 2

# Assignment to a named workspace
assign [class="^URxvt$"] → work

# Assign to the workspace with number 2, regardless of name
assign [class="^URxvt$"] → number 2

# You can also specify a number + name. If the workspace with number 2 exists,
# assign will skip the text part.
assign [class="^URxvt$"] → number "2: work"

# Start urxvt -name irssi
assign [class="^URxvt$" instance="^irssi$"] → 3

# Assign urxvt to the output right of the current one
assign [class="^URxvt$"] → output right

# Assign urxvt to the primary output
assign [class="^URxvt$"] → output primary
```

Note that you might not have a primary output configured yet. To do so, run:

```
xrandr --output <output> --primary
```

Also, the arrow is not required, it just looks good :-). If you decide to use it, it has to be a UTF-8 encoded arrow, not `->` or something like that.

To get the class and instance, you can use `xprop`. After clicking on the window, you will see the following output:

xprop:

```
WM_CLASS(STRING) = "irssi", "URxvt"
```

The first part of the WM_CLASS is the instance ("irssi" in this example), the second part is the class ("URxvt" in this example).

Should you have any problems with assignments, make sure to check the i3 logfile first (see https://i3wm.org/docs/debugging.html). It includes more details about the matching process and the window's actual class, instance and title when starting up.

Note that if you want to start an application just once on a specific workspace, but you don't want to assign all instances of it permanently, you can make use of i3's startup-notification support (see [exec]) in your config file in the following way:

Start iceweasel on workspace 3 (once):

```
# Start iceweasel on workspace 3, then switch back to workspace 1
# (Being a command-line utility, i3-msg does not support startup notifications,
#  hence the exec --no-startup-id.)
# (Starting iceweasel with i3's exec command is important in order to make i3
#  create a startup notification context, without which the iceweasel window(s)
#  cannot be matched onto the workspace on which the command was started.)
exec --no-startup-id i3-msg 'workspace 3; exec iceweasel; workspace 1'
```

4.18. Automatically starting applications on i3 startup

By using the `exec` keyword outside a keybinding, you can configure which commands will be performed by i3 on initial startup. `exec` commands will not run when restarting i3, if you need a command to run also when restarting i3 you should use the `exec_always` keyword. These commands will be run in order.

See [command_chaining] for details on the special meaning of `;` (semicolon) and `,` (comma): they chain commands together in i3, so you need to use quoted strings (as shown in [exec_quoting]) if they appear in your command.

Syntax:

```
exec [--no-startup-id] <command>
exec_always [--no-startup-id] <command>
```

Examples:

```
exec chromium
exec_always ~/my_script.sh

# Execute the terminal emulator urxvt, which is not yet startup-notification aware.
exec --no-startup-id urxvt
```

The flag --no-startup-id is explained in [exec].

4.19. Automatically putting workspaces on specific screens

If you assign clients to workspaces, it might be handy to put the workspaces on specific screens. Also, the assignment of workspaces to screens will determine which workspace i3 uses for a new screen when adding screens or when starting (e.g., by default it will use 1 for the first screen, 2 for the second screen and so on).

Syntax:

```
workspace <workspace> output <output1> [output2]…
```

The *output* is the name of the RandR output you attach your screen to. On a laptop, you might have VGA1 and LVDS1 as output names. You can see the available outputs by running `xrandr --current`.

If your X server supports RandR 1.5 or newer, i3 will use RandR monitor objects instead of output objects. Run `xrandr --listmonitors` to see a list. Usually, a monitor object contains exactly one output, and has the same name as the output; but should that not be the case, you can specify the name of either the monitor or the output in i3's configuration. For example, the Dell UP2414Q uses two scalers internally, so its output names might be "DP1" and "DP2", but the monitor name is "Dell UP2414Q".

(Note that even if you specify the name of an output which doesn't span the entire monitor, i3 will still use the entire area of the containing monitor rather than that of just the output's.)

You can specify multiple outputs. The first available will be used.

If you use named workspaces, they must be quoted:

Examples:

```
workspace 1 output LVDS1
workspace 2 output primary
workspace 5 output VGA1 LVDS1
workspace "2: vim" output VGA1
```

4.20. Changing colors

You can change all colors which i3 uses to draw the window decorations.

Syntax:

```
<colorclass> <border> <background> <text> <indicator> <child_border>
```

Where colorclass can be one of:

client.focused

A client which currently has the focus.

client.focused_inactive

A client which is the focused one of its container, but it does not have the focus at the moment.

client.unfocused

A client which is not the focused one of its container.

client.urgent

A client which has its urgency hint activated.

client.placeholder

Background and text color are used to draw placeholder window contents (when restoring layouts). Border and indicator are ignored.

client.background

Background color which will be used to paint the background of the client window on top of which the client will be rendered. Only clients which do not cover the whole area of this window expose the color. Note that this colorclass only takes a single color.

Colors are in HTML hex format (#rrggbb), see the following example:

Examples (default colors):

```
# class                 border  backgr. text    indicator child_border
client.focused          #4c7899 #285577 #ffffff #2e9ef4   #285577
client.focused_inactive #333333 #5f676a #ffffff #484e50   #5f676a
client.unfocused        #333333 #222222 #888888 #292d2e   #222222
client.urgent           #2f343a #900000 #ffffff #900000   #900000
client.placeholder      #000000 #0c0c0c #ffffff #000000   #0c0c0c

client.background       #ffffff
```

Note that for the window decorations, the color around the child window is the "child_border", and "border" color is only the two thin lines around the titlebar.

The indicator color is used for indicating where a new window will be opened. For horizontal split containers, the right border will be painted in indicator color, for vertical split containers, the bottom border. This only applies to single windows within a split container, which are otherwise indistinguishable from single windows outside of a split container.

4.21. Interprocess communication

i3 uses Unix sockets to provide an IPC interface. This allows third-party programs to get information from i3, such as the current workspaces (to display a workspace bar), and to control i3.

The IPC socket is enabled by default and will be created in `$XDG_RUNTIME_DIR/i3/ipc-socket.%p` if the directory is available, falling back to `/tmp/i3-%u.XXXXXX/ipc-socket.%p`, where `%u` is your UNIX username, `%p` is the PID of i3 and XXXXXX is a string of random characters from the portable filename character set (see mkdtemp(3)).

You can override the default path through the environment-variable `I3SOCK` or by specifying the `ipc-socket` directive. This is discouraged, though, since i3 does the right thing by default. If you decide to change it, it is strongly recommended to set this to a location in your home directory so that no other user can create that directory.

Examples:

```
ipc-socket ~/.i3/i3-ipc.sock
```

You can then use the `i3-msg` application to perform any command listed in [list_of_commands].

4.22. Focus follows mouse

By default, window focus follows your mouse movements as the mouse crosses window borders. However, if you have a setup where your mouse usually is in your way (like a touchpad on your laptop which you do not want to disable completely), you might want to disable *focus follows mouse* and control focus only by using your keyboard. The mouse will still be useful inside the currently active window (for example to click on links in your browser window).

Syntax:

```
focus_follows_mouse yes|no
```

Example:

```
focus_follows_mouse no
```

4.23. Mouse warping

By default, when switching focus to a window on a different output (e.g. focusing a window on workspace 3 on output VGA-1, coming from workspace 2 on LVDS-1), the mouse cursor is warped to the center of that window.

With the `mouse_warping` option, you can control when the mouse cursor should be warped. `none` disables warping entirely, whereas `output` is the default behavior described above.

Syntax:

```
mouse_warping output|none
```

Example:

```
mouse_warping none
```

4.24. Popups during fullscreen mode

When you are in fullscreen mode, some applications still open popup windows (take Xpdf for example). This is because these applications might not be aware that they are in fullscreen mode (they do not check the corresponding hint). There are three things which are possible to do in this situation:

1. Display the popup if it belongs to the fullscreen application only. This is the default and should be reasonable behavior for most users.

2. Just ignore the popup (don't map it). This won't interrupt you while you are in fullscreen. However, some apps might react badly to this (deadlock until you go out of fullscreen).

3. Leave fullscreen mode.

Syntax:

```
popup_during_fullscreen smart|ignore|leave_fullscreen
```

Example:

```
popup_during_fullscreen smart
```

4.25. Focus wrapping

By default, when in a container with several windows or child containers, the opposite window will be focused when trying to move the focus over the edge of a container (and there are no other containers in that direction) — the focus wraps.

If desired, you can disable this behavior by setting the `focus_wrapping` configuration directive to the value `no`.

When enabled, focus wrapping does not occur by default if there is another window or container in the specified direction, and focus will instead be set on that window or container. This is the default behavior so you can navigate to all your windows without having to use `focus parent`.

If you want the focus to **always** wrap and you are aware of using `focus parent` to switch to different containers, you can instead set `focus_wrapping` to the value `force`.

To restrict focus inside the current workspace set `focus_wrapping` to the value `workspace`. You will need to use `focus parent` until a workspace is selected to switch to a different workspace using the focus commands (the `workspace` command will still work as expected).

Syntax:

```
focus_wrapping yes|no|force|workspace

# Legacy syntax, equivalent to "focus_wrapping force"
force_focus_wrapping yes
```

Examples:

```
# Disable focus wrapping
focus_wrapping no

# Force focus wrapping
focus_wrapping force
```

4.26. Forcing Xinerama

As explained in-depth in https://i3wm.org/docs/multi-monitor.html, some X11 video drivers (especially the nVidia binary driver) only provide support for Xinerama instead of RandR. In such a situation, i3 must be told to use the inferior Xinerama API explicitly and therefore don't provide support for reconfiguring your screens on the fly (they are read only once on startup and that's it).

For people who cannot modify their `~/.xsession` to add the `--force-xinerama` commandline parameter, a configuration option is provided:

Syntax:

```
force_xinerama yes|no
```

Example:

```
force_xinerama yes
```

Also note that your output names are not descriptive (like `HDMI1`) when using Xinerama, instead they are counted up, starting at 0: `xinerama-0`, `xinerama-1`, ...

4.27. Automatic back-and-forth when switching to the current workspace

This configuration directive enables automatic `workspace back_and_forth` (see [back_and_forth]) when switching to the workspace that is currently focused.

For instance: Assume you are on workspace "1: www" and switch to "2: IM" using mod+2 because somebody sent you a message. You don't need to remember where you came from now, you can just press $mod+2 again to switch back to "1: www".

Syntax:

```
workspace_auto_back_and_forth yes|no
```

Example:

```
workspace_auto_back_and_forth yes
```

4.28. Delaying urgency hint reset on workspace change

If an application on another workspace sets an urgency hint, switching to this workspace might lead to immediate focus of the application, which also means the window decoration color would be immediately reset to `client.focused`. This might make it unnecessarily hard to tell which window originally raised the event.

In order to prevent this, you can tell i3 to delay resetting the urgency state by a certain time using the `force_display_urgency_hint` directive. Setting the value to 0 disables this feature.

The default is 500ms.

Syntax:

```
force_display_urgency_hint <timeout> ms
```

Example:

```
force_display_urgency_hint 500 ms
```

4.29. Focus on window activation

If a window is activated, e.g., via `google-chrome www.google.com`, it may request to take focus. Since this might not be preferable, different reactions can be configured.

Note that this might not affect windows that are being opened. To prevent new windows from being focused, see [no_focus].

Syntax:

```
focus_on_window_activation smart|urgent|focus|none
```

The different modes will act as follows:

smart
This is the default behavior. If the window requesting focus is on an active workspace, it will receive the focus. Otherwise, the urgency hint will be set.

urgent
The window will always be marked urgent, but the focus will not be stolen.

focus
The window will always be focused and not be marked urgent.

none
The window will neither be focused, nor be marked urgent.

4.30. Drawing marks on window decoration

If activated, marks (see [vim_like_marks]) on windows are drawn in their window decoration. However, any mark starting with an underscore in its name (_) will not be drawn even if this option is activated.

The default for this option is `yes`.

Syntax:

```
show_marks yes|no
```

Example:

```
show_marks yes
```

4.31. Line continuation

Config files support line continuation, meaning when you end a line in a backslash character (`\`), the line-break will be ignored by the parser. This feature can be used to create more readable configuration files. Commented lines are not continued.

Examples:

```
bindsym Mod1+f \
fullscreen toggle

# this line is not continued \
bindsym Mod1+F fullscreen toggle
```

5. Configuring i3bar

The bar at the bottom of your monitor is drawn by a separate process called i3bar. Having this part of "the i3 user interface" in a separate process has several advantages:

1. It is a modular approach. If you don't need a workspace bar at all, or if you prefer a different one (dzen2, xmobar, maybe even gnome-panel?), you can just remove the i3bar configuration and start your favorite bar instead.

2. It follows the UNIX philosophy of "Make each program do one thing well". While i3 manages your windows well, i3bar is good at displaying a bar on each monitor (unless you configure it otherwise).

3. It leads to two separate, clean codebases. If you want to understand i3, you don't need to bother with the details of i3bar and vice versa.

That said, i3bar is configured in the same configuration file as i3. This is because it is tightly coupled with i3 (in contrary to i3lock or i3status which are useful for people using other window managers). Therefore, it makes no sense to use a different configuration place when we already have a good configuration infrastructure in place.

Configuring your workspace bar starts with opening a `bar` block. You can have multiple bar blocks to use different settings for different outputs (monitors):

Example:

```
bar {
    status_command i3status
}
```

5.1. i3bar command

By default i3 will just pass `i3bar` and let your shell handle the execution, searching your `$PATH` for a correct version. If you have a different `i3bar` somewhere or the binary is not in your `$PATH` you can tell i3 what to execute.

The specified command will be passed to `sh -c`, so you can use globbing and have to have correct quoting etc.

Syntax:

```
i3bar_command <command>
```

Example:

```
bar {
    i3bar_command /home/user/bin/i3bar
}
```

5.2. Statusline command

i3bar can run a program and display every line of its `stdout` output on the right hand side of the bar. This is useful to display system information like your current IP address, battery status or date/time.

The specified command will be passed to `sh -c`, so you can use globbing and have to have correct quoting etc. Note that for signal handling, depending on your shell (users of dash(1) are known to be affected), you have to use the shell's exec command so that signals are passed to your program, not to the shell.

Syntax:

```
status_command <command>
```

Example:

```
bar {
    status_command i3status --config ~/.i3status.conf

    # For dash(1) users who want signal handling to work:
    status_command exec ~/.bin/my_status_command
}
```

5.3. Display mode

You can either have i3bar be visible permanently at one edge of the screen (`dock` mode) or make it show up when you press your modifier key (`hide` mode). It is also possible to force i3bar to always stay hidden (`invisible` mode). The modifier key can be configured using the `modifier` option.

The mode option can be changed during runtime through the `bar mode` command. On reload the mode will be reverted to its configured value.

The hide mode maximizes screen space that can be used for actual windows. When the bar is hidden, i3bar sends the `SIGSTOP` and `SIGCONT` signals to the `status_command` process in order to conserve battery power. This feature can be disabled by the `status_command` process by setting the appropriate values in its JSON header message.

Invisible mode allows to permanently maximize screen space, as the bar is never shown. Thus, you can configure i3bar to not disturb you by popping up because of an urgency hint or because the modifier key is pressed.

In order to control whether i3bar is hidden or shown in hide mode, there exists the hidden_state option, which has no effect in dock mode or invisible mode. It indicates the current hidden_state of the bar: (1) The bar acts like in normal hide mode, it is hidden and is only unhidden in case of urgency hints or by pressing the modifier key (`hide` state), or (2) it is drawn on top of the currently visible workspace (`show` state).

Like the mode, the hidden_state can also be controlled through i3, this can be done by using the `bar hidden_state` command.

The default mode is dock mode; in hide mode, the default modifier is Mod4 (usually the windows key). The default value for the hidden_state is hide.

Syntax:

```
mode dock|hide|invisible
hidden_state hide|show
modifier <Modifier>|none
```

Example:

```
bar {
    mode hide
    hidden_state hide
    modifier Mod1
}
```

Available modifiers are Mod1-Mod5, Shift, Control (see `xmodmap(1)`). You can also use "none" if you don't want any modifier to trigger this behavior.

5.4. Mouse button commands

Specifies a command to run when a button was pressed on i3bar to override the default behavior. This is useful, e.g., for disabling the scroll wheel action or running scripts that implement custom behavior for these buttons.

A button is always named `button<n>`, where 1 to 5 are default buttons as follows and higher numbers can be special buttons on devices offering more buttons:

button1
Left mouse button.

button2
Middle mouse button.

button3
Right mouse button.

button4
Scroll wheel up.

button5
Scroll wheel down.

button6
Scroll wheel right.

button7
Scroll wheel left.

Please note that the old `wheel_up_cmd` and `wheel_down_cmd` commands are deprecated and will be removed in a future release. We strongly recommend using the more general `bindsym` with `button4` and `button5` instead.

Syntax:

```
bindsym [--release] button<n> <command>
```

Example:

```
bar {
    # disable clicking on workspace buttons
    bindsym button1 nop
    # Take a screenshot by right clicking on the bar
    bindsym --release button3 exec --no-startup-id import /tmp/latest-screenshot.png
    # execute custom script when scrolling downwards
    bindsym button5 exec ~/.i3/scripts/custom_wheel_down
}
```

5.5. Bar ID

Specifies the bar ID for the configured bar instance. If this option is missing, the ID is set to *bar-x*, where x corresponds to the position of the embedding bar block in the config file (*bar-0*, *bar-1*, ...).

Syntax:

```
id <bar_id>
```

Example:

```
bar {
    id bar-1
}
```

5.6. Position

This option determines in which edge of the screen i3bar should show up.

The default is bottom.

Syntax:

```
position top|bottom
```

Example:

```
bar {
    position top
}
```

5.7. Output(s)

You can restrict i3bar to one or more outputs (monitors). The default is to handle all outputs. Restricting the outputs is useful for using different options for different outputs by using multiple *bar* blocks.

To make a particular i3bar instance handle multiple outputs, specify the output directive multiple times.

These output names have a special meaning:

primary
Selects the output that is configured as primary in the X server.

nonprimary
Selects every output that is not configured as primary in the X server.

Syntax:

```
output primary|nonprimary|<output>
```

Example:

```
# big monitor: everything
bar {
    # The display is connected either via HDMI or via DisplayPort
    output HDMI2
    output DP2
    status_command i3status
}

# laptop monitor: bright colors and i3status with less modules.
bar {
    output LVDS1
    status_command i3status --config ~/.i3status-small.conf
    colors {
        background #000000
        statusline #ffffff
    }
}
```

```
# show bar on the primary monitor and on HDMI2
bar {
    output primary
    output HDMI2
    status_command i3status
}
```

Note that you might not have a primary output configured yet. To do so, run:

```
xrandr --output <output> --primary
```

5.8. Tray output

i3bar by default provides a system tray area where programs such as NetworkManager, VLC, Pidgin, etc. can place little icons.

You can configure on which output (monitor) the icons should be displayed or you can turn off the functionality entirely.

You can use multiple `tray_output` directives in your config to specify a list of outputs on which you want the tray to appear. The first available output in that list as defined by the order of the directives will be used for the tray output.

Syntax:

```
tray_output none|primary|<output>
```

Example:

```
# disable system tray
bar {
    tray_output none
}

# show tray icons on the primary monitor
bar {
    tray_output primary
}

# show tray icons on the big monitor
bar {
    tray_output HDMI2
}
```

Note that you might not have a primary output configured yet. To do so, run:

```
xrandr --output <output> --primary
```

Note that when you use multiple bar configuration blocks, either specify `tray_output primary` in all of them or explicitly specify `tray_output none` in bars which should not display the tray, otherwise the different instances might race each other in trying to display tray icons.

5.9. Tray padding

The tray is shown on the right-hand side of the bar. By default, a padding of 2 pixels is used for the upper, lower and right-hand side of the tray area and between the individual icons.

Syntax:

```
tray_padding <px> [px]
```

```
# Obey Fitts's law
tray_padding 0
```

5.10. Font

Specifies the font to be used in the bar. See [fonts].

```
font <font>
```

```
bar {
    font -misc-fixed-medium-r-normal--13-120-75-75-C-70-iso10646-1
    font pango:DejaVu Sans Mono 10
}
```

5.11. Custom separator symbol

Specifies a custom symbol to be used for the separator as opposed to the vertical, one pixel thick separator.

```
separator_symbol <symbol>
```

```
bar {
    separator_symbol ":|:"
}
```

5.12. Workspace buttons

Specifies whether workspace buttons should be shown or not. This is useful if you want to display a statusline-only bar containing additional information.

The default is to show workspace buttons.

```
workspace_buttons yes|no
```

```
bar {
    workspace_buttons no
}
```

5.13. Minimal width for workspace buttons

By default, the width a workspace button is determined by the width of the text showing the workspace name. If the name is too short (say, one letter), then the workspace button might look too small.

This option specifies the minimum width for workspace buttons. If the name of a workspace is too short to cover the button, an additional padding is added on both sides of the button so that the text is centered.

The default value of zero means that no additional padding is added.

The setting also applies to the current binding mode indicator.

Note that the specified pixels refer to logical pixels, which might translate into more pixels on HiDPI displays.

Syntax:

```
workspace_min_width <px> [px]
```

Example:

```
bar {
    workspace_min_width 40
}
```

5.14. Strip workspace numbers/name

Specifies whether workspace numbers should be displayed within the workspace buttons. This is useful if you want to have a named workspace that stays in order on the bar according to its number without displaying the number prefix.

When `strip_workspace_numbers` is set to `yes`, any workspace that has a name of the form "[n][:][NAME]" will display only the name. You could use this, for instance, to display Roman numerals rather than digits by naming your workspaces to "1:I", "2:II", "3:III", "4:IV", ...

When `strip_workspace_name` is set to `yes`, any workspace that has a name of the form "[n][:][NAME]" will display only the number.

The default is to display the full name within the workspace button. Be aware that the colon in the workspace name is optional, so `[n][NAME]` will also have the workspace name and number stripped correctly.

Syntax:

```
strip_workspace_numbers yes|no
strip_workspace_name yes|no
```

Example:

```
bar {
    strip_workspace_numbers yes
}
```

5.15. Binding Mode indicator

Specifies whether the current binding mode indicator should be shown or not. This is useful if you want to hide the workspace buttons but still be able to see the current binding mode indicator. See [binding_modes] to learn what modes are and how to use them.

The default is to show the mode indicator.

Syntax:

```
binding_mode_indicator yes|no
```

Example:

```
bar {
    binding_mode_indicator no
}
```

5.16. Colors

As with i3, colors are in HTML hex format (#rrggbb). The following colors can be configured at the moment:

background
Background color of the bar.

statusline
Text color to be used for the statusline.

separator
Text color to be used for the separator.

focused_background
Background color of the bar on the currently focused monitor output. If not used, the color will be taken from background.

focused_statusline
Text color to be used for the statusline on the currently focused monitor output. If not used, the color will be taken from statusline.

focused_separator
Text color to be used for the separator on the currently focused monitor output. If not used, the color will be taken from separator.

focused_workspace
Border, background and text color for a workspace button when the workspace has focus.

active_workspace
Border, background and text color for a workspace button when the workspace is active (visible) on some output, but the focus is on another one. You can only tell this apart from the focused workspace when you are using multiple monitors.

inactive_workspace
Border, background and text color for a workspace button when the workspace does not have focus and is not active (visible) on any output. This will be the case for most workspaces.

urgent_workspace
Border, background and text color for a workspace button when the workspace contains a window with the urgency hint set.

binding_mode
Border, background and text color for the binding mode indicator. If not used, the colors will be taken from urgent_workspace.

Syntax:

```
colors {
    background <color>
    statusline <color>
    separator <color>

    <colorclass> <border> <background> <text>
}
```

Example (default colors):

```
bar {
    colors {
        background #000000
        statusline #ffffff
        separator #666666

        focused_workspace  #4c7899 #285577 #ffffff
        active_workspace   #333333 #5f676a #ffffff
        inactive_workspace #333333 #222222 #888888
        urgent_workspace   #2f343a #900000 #ffffff
        binding_mode       #2f343a #900000 #ffffff
```

```
        }
}
```

5.17. Transparency

i3bar can support transparency by passing the `--transparency` flag in the configuration:

Syntax:

```
bar {
    i3bar_command i3bar --transparency
}
```

In the i3bar color configuration and i3bar status block color attribute you can then use colors in the RGBA format, i.e. the last two (hexadecimal) digits specify the opacity. For example, #00000000 will be completely transparent, while #000000FF will be a fully opaque black (the same as #000000).

Please note that due to the way the tray specification works, enabling this flag will cause all tray icons to have a transparent background.

6. List of commands

Commands are what you bind to specific keypresses. You can also issue commands at runtime without pressing a key by using the IPC interface. An easy way to do this is to use the `i3-msg` utility:

Example:

```
# execute this on your shell to make the current container borderless
i3-msg border none
```

Commands can be chained by using `;` (a semicolon). So, to move a window to a specific workspace and immediately switch to that workspace, you can configure the following keybinding:

Example:

```
bindsym $mod+x move container to workspace 3; workspace 3
```

Furthermore, you can change the scope of a command - that is, which containers should be affected by that command, by using various criteria. The criteria are specified before any command in a pair of square brackets and are separated by space.

When using multiple commands, separate them by using a `,` (a comma) instead of a semicolon. Criteria apply only until the next semicolon, so if you use a semicolon to separate commands, only the first one will be executed for the matched window(s).

Example:

```
# if you want to kill all windows which have the class Firefox, use:
bindsym $mod+x [class="Firefox"] kill

# same thing, but case-insensitive
bindsym $mod+x [class="(?i)firefox"] kill

# kill only the About dialog from Firefox
bindsym $mod+x [class="Firefox" window_role="About"] kill

# enable floating mode and move container to workspace 4
for_window [class="^evil-app$"] floating enable, move container to workspace 4
```

```
# move all floating windows to the scratchpad
bindsym $mod+x [floating] move scratchpad
```

The criteria which are currently implemented are:

class
Compares the window class (the second part of WM_CLASS). Use the special value __focused__ to match all windows having the same window class as the currently focused window.

instance
Compares the window instance (the first part of WM_CLASS). Use the special value __focused__ to match all windows having the same window instance as the currently focused window.

window_role
Compares the window role (WM_WINDOW_ROLE). Use the special value __focused__ to match all windows having the same window role as the currently focused window.

window_type
Compare the window type (_NET_WM_WINDOW_TYPE). Possible values are normal, dialog, utility, toolbar, splash, menu, dropdown_menu, popup_menu, tooltip and notification.

id
Compares the X11 window ID, which you can get via xwininfo for example.

title
Compares the X11 window title (_NET_WM_NAME or WM_NAME as fallback). Use the special value __focused__ to match all windows having the same window title as the currently focused window.

urgent
Compares the urgent state of the window. Can be "latest" or "oldest". Matches the latest or oldest urgent window, respectively. (The following aliases are also available: newest, last, recent, first)

workspace
Compares the workspace name of the workspace the window belongs to. Use the special value __focused__ to match all windows in the currently focused workspace.

con_mark
Compares the marks set for this container, see [vim_like_marks]. A match is made if any of the container's marks matches the specified mark.

con_id
Compares the i3-internal container ID, which you can get via the IPC interface. Handy for scripting. Use the special value __focused__ to match only the currently focused window.

floating
Only matches floating windows. This criterion requires no value.

floating_from
Like floating but this criterion takes two possible values: "auto" and "user". With "auto", only windows that were automatically opened as floating are matched. With "user", only windows that the user made floating are matched.

tiling
Only matches tiling windows. This criterion requires no value.

tiling_from
Like tiling but this criterion takes two possible values: "auto" and "user". With "auto", only windows that were automatically opened as tiling are matched. With "user", only windows that the user made tiling are matched.

The criteria class, instance, role, title, workspace and mark are actually regular expressions (PCRE). See pcresyntax(3) or perldoc perlre for information on how to use them.

6.1. Executing applications (exec)

What good is a window manager if you can't actually start any applications? The exec command starts an application by passing the command you specify to a shell. This implies that you can use globbing (wildcards) and programs will be searched in your $PATH.

See [command_chaining] for details on the special meaning of `;` (semicolon) and `,` (comma): they chain commands together in i3, so you need to use quoted strings (as shown in [exec_quoting]) if they appear in your command.

Syntax:

```
exec [--no-startup-id] <command>
```

Example:

```
# Start the GIMP
bindsym $mod+g exec gimp

# Start the terminal emulator urxvt which is not yet startup-notification-aware
bindsym $mod+Return exec --no-startup-id urxvt
```

The `--no-startup-id` parameter disables startup-notification support for this particular exec command. With startup-notification, i3 can make sure that a window appears on the workspace on which you used the exec command. Also, it will change the X11 cursor to `watch` (a clock) while the application is launching. So, if an application is not startup-notification aware (most GTK and Qt using applications seem to be, though), you will end up with a watch cursor for 60 seconds.

If the command to be executed contains a `;` (semicolon) and/or a `,` (comma), the entire command must be quoted. For example, to have a keybinding for the shell command `notify-send Hello, i3`, you would add an entry to your configuration file like this:

Example:

```
# Execute a command with a comma in it
bindsym $mod+p exec "notify-send Hello, i3"
```

If however a command with a comma and/or semicolon itself requires quotes, you must escape the internal quotation marks with double backslashes, like this:

Example:

```
# Execute a command with a comma, semicolon and internal quotes
bindsym $mod+p exec "notify-send \\"Hello, i3; from $USER\\""
```

6.2. Splitting containers

The split command makes the current window a split container. Split containers can contain multiple windows. Depending on the layout of the split container, new windows get placed to the right of the current one (splith) or new windows get placed below the current one (splitv).

If you apply this command to a split container with the same orientation, nothing will happen. If you use a different orientation, the split container's orientation will be changed (if it does not have more than one window). The `toggle` option will toggle the orientation of the split container if it contains a single window. Otherwise it makes the current window a split container with opposite orientation compared to the parent container. Use `layout toggle split` to change the layout of any split container from splitv to splith or vice-versa. You can also define a custom sequence of layouts to cycle through with `layout toggle`, see [manipulating_layout].

Syntax:

```
split vertical|horizontal|toggle
```

Example:

```
bindsym $mod+v split vertical
bindsym $mod+h split horizontal
bindsym $mod+t split toggle
```

6.3. Manipulating layout

Use `layout toggle split`, `layout stacking`, `layout tabbed`, `layout splitv` or `layout splith` to change the current container layout to splith/splitv, stacking, tabbed layout, splitv or splith, respectively.

Specify up to four layouts after `layout toggle` to cycle through them. Every time the command is executed, the layout specified after the currently active one will be applied. If the currently active layout is not in the list, the first layout in the list will be activated.

To make the current window (!) fullscreen, use `fullscreen enable` (or `fullscreen enable global` for the global mode), to leave either fullscreen mode use `fullscreen disable`, and to toggle between these two states use `fullscreen toggle` (or `fullscreen toggle global`).

Likewise, to make the current window floating (or tiling again) use `floating enable` respectively `floating disable` (or `floating toggle`):

Syntax:

```
layout default|tabbed|stacking|splitv|splith
layout toggle [split|all]
layout toggle [split|tabbed|stacking|splitv|splith] [split|tabbed|stacking|splitv|splith]…
```

Examples:

```
bindsym $mod+s layout stacking
bindsym $mod+l layout toggle split
bindsym $mod+w layout tabbed

# Toggle between stacking/tabbed/split:
bindsym $mod+x layout toggle

# Toggle between stacking/tabbed/splith/splitv:
bindsym $mod+x layout toggle all

# Toggle between stacking/tabbed/splith:
bindsym $mod+x layout toggle stacking tabbed splith

# Toggle between splitv/tabbed
bindsym $mod+x layout toggle splitv tabbed

# Toggle between last split layout/tabbed/stacking
bindsym $mod+x layout toggle split tabbed stacking

# Toggle fullscreen
bindsym $mod+f fullscreen toggle

# Toggle floating/tiling
bindsym $mod+t floating toggle
```

6.4. Focusing containers

To change focus, you can use the `focus` command. The following options are available:

<criteria>
Sets focus to the container that matches the specified criteria. See [command_criteria].

left|right|up|down
Sets focus to the nearest container in the given direction.

parent
Sets focus to the parent container of the current container.

child

The opposite of `focus parent`, sets the focus to the last focused child container.

next|prev

Automatically sets focus to the adjacent container. If `sibling` is specified, the command will focus the exact sibling container, including non-leaf containers like split containers. Otherwise, it is an automatic version of `focus left|right|up|down` in the orientation of the parent container.

floating

Sets focus to the last focused floating container.

tiling

Sets focus to the last focused tiling container.

mode_toggle

Toggles between floating/tiling containers.

output

Followed by a direction or an output name, this will focus the corresponding output.

Syntax:

```
<criteria> focus
focus left|right|down|up
focus parent|child|floating|tiling|mode_toggle
focus next|prev [sibling]
focus output left|right|up|down|primary|<output>
```

Examples:

```
# Focus firefox
bindsym $mod+F1 [class="Firefox"] focus

# Focus container on the left, bottom, top, right
bindsym $mod+j focus left
bindsym $mod+k focus down
bindsym $mod+l focus up
bindsym $mod+semicolon focus right

# Focus parent container
bindsym $mod+u focus parent

# Focus last floating/tiling container
bindsym $mod+g focus mode_toggle

# Focus the output right to the current one
bindsym $mod+x focus output right

# Focus the big output
bindsym $mod+x focus output HDMI-2

# Focus the primary output
bindsym $mod+x focus output primary
```

Note that you might not have a primary output configured yet. To do so, run:

```
xrandr --output <output> --primary
```

6.5. Moving containers

Use the move command to move a container.

Syntax:

```
# Moves the container into the given direction.
# The optional pixel argument specifies how far the
# container should be moved if it is floating and
# defaults to 10 pixels. The optional ppt argument
# means "percentage points", and if specified it indicates
# how many points the container should be moved if it is
# floating rather than by a pixel value.
move <left|right|down|up> [<amount> [px|ppt]]

# Moves the container to the specified pos_x and pos_y
# coordinates on the screen.
move position <pos_x> [px|ppt] <pos_y> [px|ppt]

# Moves the container to the center of the screen.
# If 'absolute' is used, it is moved to the center of
# all outputs.
move [absolute] position center

# Moves the container to the current position of the
# mouse cursor. Only affects floating containers.
move position mouse
```

Examples:

```
# Move container to the left, bottom, top, right
bindsym $mod+j move left
bindsym $mod+k move down
bindsym $mod+l move up
bindsym $mod+semicolon move right

# Move container, but make floating containers
# move more than the default
bindsym $mod+j move left 20 px

# Move floating container to the center of all outputs
bindsym $mod+c move absolute position center

# Move container to the current position of the cursor
bindsym $mod+m move position mouse
```

6.6. Swapping containers

Two containers can be swapped (i.e., move to each other's position) by using the swap command. They will assume the position and geometry of the container they are swapped with.

The first container to participate in the swapping can be selected through the normal command criteria process with the focused window being the usual fallback if no criteria are specified. The second container can be selected using one of the following methods:

id
The X11 window ID of a client window.

con_id
The i3 container ID of a container.

mark
A container with the specified mark, see [vim_like_marks].

Note that swapping does not work with all containers. Most notably, swapping containers that have a parent-child relationship to one another does not work.

Syntax:

```
swap container with id|con_id|mark <arg>
```

Examples:

```
# Swaps the focused container with the container marked »swapee«.
swap container with mark swapee

# Swaps container marked »A« and »B«
[con_mark="^A$"] swap container with mark B
```

6.7. Sticky floating windows

If you want a window to stick to the glass, i.e., have it stay on screen even if you switch to another workspace, you can use the `sticky` command. For example, this can be useful for notepads, a media player or a video chat window.

Note that while any window can be made sticky through this command, it will only take effect if the window is floating.

Syntax:

```
sticky enable|disable|toggle
```

Examples:

```
# make a terminal sticky that was started as a notepad
for_window [instance=notepad] sticky enable
```

6.8. Changing (named) workspaces/moving to workspaces

To change to a specific workspace, use the `workspace` command, followed by the number or name of the workspace. Pass the optional flag `--no-auto-back-and-forth` to disable [workspace_auto_back_and_forth] for this specific call only.

To move containers to specific workspaces, use `move container to workspace`.

You can also switch to the next and previous workspace with the commands `workspace next` and `workspace prev`, which is handy, for example, if you have workspace 1, 3, 4 and 9 and you want to cycle through them with a single key combination. To restrict those to the current output, use `workspace next_on_output` and `workspace prev_on_output`. Similarly, you can use `move container to workspace next`, `move container to workspace prev` to move a container to the next/previous workspace and `move container to workspace current` (the last one makes sense only when used with criteria).

`workspace next` cycles through either numbered or named workspaces. But when it reaches the last numbered/named workspace, it looks for named workspaces after exhausting numbered ones and looks for numbered ones after exhausting named ones.

See [move_to_outputs] for how to move a container/workspace to a different RandR output.

Workspace names are parsed as Pango markup by i3bar.

To switch back to the previously focused workspace, use `workspace back_and_forth`; likewise, you can move containers to the previously focused workspace using `move container to workspace back_and_forth`.

Syntax:

```
workspace next|prev|next_on_output|prev_on_output
workspace back_and_forth
workspace [--no-auto-back-and-forth] <name>
workspace [--no-auto-back-and-forth] number <name>
```

```
move [--no-auto-back-and-forth] [window|container] [to] workspace <name>
move [--no-auto-back-and-forth] [window|container] [to] workspace number <name>
move [window|container] [to] workspace prev|next|current
```

Examples:

```
bindsym $mod+1 workspace 1
bindsym $mod+2 workspace 2
bindsym $mod+3 workspace 3:<span foreground="red">vim</span>
...

bindsym $mod+Shift+1 move container to workspace 1
bindsym $mod+Shift+2 move container to workspace 2
...

# switch between the current and the previously focused one
bindsym $mod+b workspace back_and_forth
bindsym $mod+Shift+b move container to workspace back_and_forth

# move the whole workspace to the next output
bindsym $mod+x move workspace to output right

# move firefox to current workspace
bindsym $mod+F1 [class="Firefox"] move workspace current
```

6.8.1. Named workspaces

Workspaces are identified by their name. So, instead of using numbers in the workspace command, you can use an arbitrary name:

Example:

```
bindsym $mod+1 workspace mail
...
```

If you want the workspace to have a number **and** a name, just prefix the number, like this:

Example:

```
bindsym $mod+1 workspace 1: mail
bindsym $mod+2 workspace 2: www
...
```

Note that the workspace will really be named "1: mail". i3 treats workspace names beginning with a number in a slightly special way. Normally, named workspaces are ordered the way they appeared. When they start with a number, i3 will order them numerically. Also, you will be able to use `workspace number 1` to switch to the workspace which begins with number 1, regardless of which name it has. This is useful in case you are changing the workspace's name dynamically. To combine both commands you can use `workspace number 1: mail` to specify a default name if there's currently no workspace starting with a "1".

6.8.2. Renaming workspaces

You can rename workspaces. This might be useful to start with the default numbered workspaces, do your work, and rename the workspaces afterwards to reflect what's actually on them. You can also omit the old name to rename the currently focused workspace. This is handy if you want to use the rename command with `i3-input`.

Syntax:

```
rename workspace <old_name> to <new_name>
rename workspace to <new_name>
```

Examples:

```
i3-msg 'rename workspace 5 to 6'
i3-msg 'rename workspace 1 to "1: www"'
i3-msg 'rename workspace "1: www" to "10: www"'
i3-msg 'rename workspace to "2: mail"'
bindsym $mod+r exec i3-input -F 'rename workspace to "%s"' -P 'New name: '
```

If you want to rename workspaces on demand while keeping the navigation stable, you can use a setup like this:

Example:

```
bindsym $mod+1 workspace number "1: www"
bindsym $mod+2 workspace number "2: mail"
...
```

If a workspace does not exist, the command `workspace number "1: mail"` will create workspace "1: mail".

If a workspace with number 1 does already exist, the command will switch to this workspace and ignore the text part. So even when the workspace has been renamed to "1: web", the above command will still switch to it.

6.9. Moving workspaces to a different screen

See [move_to_outputs] for how to move a container/workspace to a different RandR output.

6.10. Moving containers/workspaces to RandR outputs

To move a container to another RandR output (addressed by names like LVDS1 or VGA1) or to a RandR output identified by a specific direction (like `left`, `right`, `up` or `down`), there are two commands:

Syntax:

```
move container to output left|right|down|up|current|primary|<output>
move workspace to output left|right|down|up|current|primary|<output>
```

Examples:

```
# Move the current workspace to the next output
# (effectively toggles when you only have two outputs)
bindsym $mod+x move workspace to output right

# Put this window on the presentation output.
bindsym $mod+x move container to output VGA1

# Put this window on the primary output.
bindsym $mod+x move container to output primary
```

Note that you might not have a primary output configured yet. To do so, run:

```
xrandr --output <output> --primary
```

6.11. Moving containers/windows to marks

To move a container to another container with a specific mark (see [vim_like_marks]), you can use the following command.

The window will be moved right after the marked container in the tree, i.e., it ends up in the same position as if you had opened a new window when the marked container was focused. If the mark is on a split container, the window will appear as a new child after the currently focused child within that container.

Syntax:

```
move window|container to mark <mark>
```

Example:

```
for_window [instance="tabme"] move window to mark target
```

6.12. Resizing containers/windows

If you want to resize containers/windows using your keyboard, you can use the `resize` command:

Syntax:

```
resize grow|shrink <direction> [<px> px [or <ppt> ppt]]
resize set [width] <width> [px | ppt]
resize set height <height> [px | ppt]
resize set [width] <width> [px | ppt] [height] <height> [px | ppt]
```

Direction can either be one of `up`, `down`, `left` or `right`. Or you can be less specific and use `width` or `height`, in which case i3 will take/give space from all the other containers. The optional pixel argument specifies by how many pixels a container should be grown or shrunk (the default is 10 pixels). The optional ppt argument means "percentage points", and if specified it indicates that a **tiling container** should be grown or shrunk by that many points, instead of by the `px` value.

Note about `resize set`: a value of 0 for <width> or <height> means "do not resize in this direction".

It is recommended to define bindings for resizing in a dedicated binding mode. See [binding_modes] and the example in the i3 default config for more context.

Example:

```
for_window [class="urxvt"] resize set 640 480
```

6.13. Jumping to specific windows

Often when in a multi-monitor environment, you want to quickly jump to a specific window. For example, while working on workspace 3 you might want to jump to your mail client to email your boss that you've achieved some important goal. Instead of figuring out how to navigate to your mail client, it would be more convenient to have a shortcut. You can use the `focus` command with criteria for that.

Syntax:

```
[class="class"] focus
[title="title"] focus
```

Examples:

```
# Get me to the next open VIM instance
bindsym $mod+a [class="urxvt" title="VIM"] focus
```

6.14. VIM-like marks (mark/goto)

This feature is like the jump feature: It allows you to directly jump to a specific window (this means switching to the appropriate workspace and setting focus to the windows). However, you can directly mark a specific window with an arbitrary label and use it afterwards. You can unmark the label in the same way, using the unmark command. If you don't specify a label, unmark removes all marks. You do not need to ensure that your windows have unique classes or titles, and you do not need to change your configuration file.

As the command needs to include the label with which you want to mark the window, you cannot simply bind it to a key. `i3-input` is a tool created for this purpose: It lets you input a command and sends the command to i3. It can also prefix this command and display a custom prompt for the input dialog.

The additional `--toggle` option will remove the mark if the window already has this mark or add it otherwise. Note that you might need to use this in combination with `--add` (see below) as any other marks will otherwise be removed.

The `--replace` flag causes i3 to remove any existing marks, which is also the default behavior. You can use the `--add` flag to put more than one mark on a window.

Refer to [show_marks] if you don't want marks to be shown in the window decoration.

Syntax:

```
mark [--add|--replace] [--toggle] <identifier>
[con_mark="identifier"] focus
unmark <identifier>
```

You can use `i3-input` to prompt for a mark name, then use the `mark` and `focus` commands to create and jump to custom marks:

Examples:

```
# read 1 character and mark the current window with this character
bindsym $mod+m exec i3-input -F 'mark %s' -l 1 -P 'Mark: '

# read 1 character and go to the window with the character
bindsym $mod+g exec i3-input -F '[con_mark="%s"] focus' -l 1 -P 'Goto: '
```

Alternatively, if you do not want to mess with `i3-input`, you could create separate bindings for a specific set of labels and then only use those labels:

Example (in a terminal):

```
# marks the focused container
mark irssi

# focus the container with the mark "irssi"
'[con_mark="irssi"] focus'

# remove the mark "irssi" from whichever container has it
unmark irssi

# remove all marks on all firefox windows
[class="(?i)firefox"] unmark
```

6.15. Window title format

By default, i3 will simply print the X11 window title. Using `title_format`, this can be customized by setting the format to the desired output. This directive supports Pango markup and the following placeholders which will be replaced:

`%title`
For normal windows, this is the X11 window title (_NET_WM_NAME or WM_NAME as fallback). When used on containers without a window (e.g., a split container inside a tabbed/stacked layout), this will be the tree representation of the container (e.g., "H[xterm xterm]").

`%class`
The X11 window class (second part of WM_CLASS). This corresponds to the `class` criterion, see [command_criteria].

`%instance`
The X11 window instance (first part of WM_CLASS). This corresponds to the `instance` criterion, see [command_criteria].

Using the [for_window] directive, you can set the title format for any window based on [command_criteria].

Syntax:

```
title_format <format>
```

Examples:

```
# give the focused window a prefix
bindsym $mod+p title_format "Important | %title"

# print all window titles bold
for_window [class=".*"] title_format "<b>%title</b>"

# print window titles of firefox windows red
for_window [class="(?i)firefox"] title_format "<span foreground='red'>%title</span>"
```

6.16. Changing border style

To change the border of the current client, you can use `border normal` to use the normal border (including window title), `border pixel 1` to use a 1-pixel border (no window title) and `border none` to make the client borderless.

There is also `border toggle` which will toggle the different border styles. The optional pixel argument can be used to specify the border width when switching to the normal and pixel styles.

Note that "pixel" refers to logical pixel. On HiDPI displays, a logical pixel is represented by multiple physical pixels, so `pixel 1` might not necessarily translate into a single pixel row wide border.

Syntax:

```
border normal|pixel|toggle [<n>]
border none

# legacy syntax, equivalent to "border pixel 1"
border 1pixel
```

Examples:

```
# use window title, but no border
bindsym $mod+t border normal 0
# use no window title and a thick border
bindsym $mod+y border pixel 3
# use neither window title nor border
bindsym $mod+u border none
```

6.17. Enabling shared memory logging

As described in https://i3wm.org/docs/debugging.html, i3 can log to a shared memory buffer, which you can dump using `i3-dump-log`. The `shmlog` command allows you to enable or disable the shared memory logging at runtime.

Note that when using `shmlog <size_in_bytes>`, the current log will be discarded and a new one will be started.

Syntax:

```
shmlog <size_in_bytes>
shmlog on|off|toggle
```

Examples:

```
# Enable/disable logging
bindsym $mod+x shmlog toggle

# or, from a terminal:
# increase the shared memory log buffer to 50 MiB
i3-msg shmlog $((50*1024*1024))
```

6.18. Enabling debug logging

The debuglog command allows you to enable or disable debug logging at runtime. Debug logging is much more verbose than non-debug logging. This command does not activate shared memory logging (shmlog), and as such is most likely useful in combination with the above-described [shmlog] command.

Syntax:

```
debuglog on|off|toggle
```

Examples:

```
# Enable/disable logging
bindsym $mod+x debuglog toggle
```

6.19. Reloading/Restarting/Exiting

You can make i3 reload its configuration file with reload. You can also restart i3 inplace with the restart command to get it out of some weird state (if that should ever happen) or to perform an upgrade without having to restart your X session. To exit i3 properly, you can use the exit command, however you don't need to (simply killing your X session is fine as well).

Examples:

```
bindsym $mod+Shift+r restart
bindsym $mod+Shift+w reload
bindsym $mod+Shift+e exit
```

6.20. Scratchpad

There are two commands to use any existing window as scratchpad window. move scratchpad will move a window to the scratchpad workspace. This will make it invisible until you show it again. There is no way to open that workspace. Instead, when using scratchpad show, the window will be shown again, as a floating window, centered on your current workspace (using scratchpad show on a visible scratchpad window will make it hidden again, so you can have a keybinding to toggle). Note that this is just a normal floating window, so if you want to "remove it from scratchpad", you can simple make it tiling again (floating toggle).

As the name indicates, this is useful for having a window with your favorite editor always at hand. However, you can also use this for other permanently running applications which you don't want to see all the time: Your music player, alsamixer, maybe even your mail client...?

Syntax:

```
move scratchpad

scratchpad show
```

Examples:

```
# Make the currently focused window a scratchpad
bindsym $mod+Shift+minus move scratchpad

# Show the first scratchpad window
bindsym $mod+minus scratchpad show

# Show the sup-mail scratchpad window, if any.
bindsym mod4+s [title="^Sup ::"] scratchpad show
```

6.21. Nop

There is a no operation command nop which allows you to override default behavior. This can be useful for, e.g., disabling a focus change on clicks with the middle mouse button.

The optional comment argument is ignored, but will be printed to the log file for debugging purposes.

Syntax:

```
nop [<comment>]
```

Example:

```
# Disable focus change for clicks on titlebars
# with the middle mouse button
bindsym button2 nop
```

6.22. i3bar control

There are two options in the configuration of each i3bar instance that can be changed during runtime by invoking a command through i3. The commands bar hidden_state and bar mode allow setting the current hidden_state respectively mode option of each bar. It is also possible to toggle between hide state and show state as well as between dock mode and hide mode. Each i3bar instance can be controlled individually by specifying a bar_id, if none is given, the command is executed for all bar instances.

Syntax:

```
bar hidden_state hide|show|toggle [<bar_id>]

bar mode dock|hide|invisible|toggle [<bar_id>]
```

Examples:

```
# Toggle between hide state and show state
bindsym $mod+m bar hidden_state toggle

# Toggle between dock mode and hide mode
bindsym $mod+n bar mode toggle

# Set the bar instance with id 'bar-1' to switch to hide mode
bindsym $mod+b bar mode hide bar-1

# Set the bar instance with id 'bar-1' to always stay hidden
bindsym $mod+Shift+b bar mode invisible bar-1
```

7. Multiple monitors

As you can see in the goal list on the website, i3 was specifically developed with support for multiple monitors in mind. This section will explain how to handle multiple monitors.

When you have only one monitor, things are simple. You usually start with workspace 1 on your monitor and open new ones as you need them.

When you have more than one monitor, each monitor will get an initial workspace. The first monitor gets 1, the second gets 2 and a possible third would get 3. When you switch to a workspace on a different monitor, i3 will switch to that monitor and then switch to the workspace. This way, you don't need shortcuts to switch to a specific monitor, and you don't need to remember where you put which workspace. New workspaces will be opened on the currently active monitor. It is not possible to have a monitor without a workspace.

The idea of making workspaces global is based on the observation that most users have a very limited set of workspaces on their additional monitors. They are often used for a specific task (browser, shell) or for monitoring several things (mail, IRC, syslog, ...). Thus, using one workspace on one monitor and "the rest" on the other monitors often makes sense. However, as you can create an unlimited number of workspaces in i3 and tie them to specific screens, you can have the "traditional" approach of having X workspaces per screen by changing your configuration (using modes, for example).

7.1. Configuring your monitors

To help you get going if you have never used multiple monitors before, here is a short overview of the xrandr options which will probably be of interest to you. It is always useful to get an overview of the current screen configuration. Just run "xrandr" and you will get an output like the following:

```
$ xrandr
Screen 0: minimum 320 x 200, current 1280 x 800, maximum 8192 x 8192
VGA1 disconnected (normal left inverted right x axis y axis)
LVDS1 connected 1280x800+0+0 (normal left inverted right x axis y axis) 261mm x 163mm
   1280x800       60.0*+   50.0
   1024x768       85.0     75.0     70.1     60.0
   832x624        74.6
   800x600        85.1     72.2     75.0     60.3     56.2
   640x480        85.0     72.8     75.0     59.9
   720x400        85.0
   640x400        85.1
   640x350        85.1
```

Several things are important here: You can see that LVDS1 is connected (of course, it is the internal flat panel) but VGA1 is not. If you have a monitor connected to one of the ports but xrandr still says "disconnected", you should check your cable, monitor or graphics driver.

The maximum resolution you can see at the end of the first line is the maximum combined resolution of your monitors. By default, it is usually too low and has to be increased by editing /etc/X11/xorg.conf.

So, say you connected VGA1 and want to use it as an additional screen:

```
xrandr --output VGA1 --auto --left-of LVDS1
```

This command makes xrandr try to find the native resolution of the device connected to VGA1 and configures it to the left of your internal flat panel. When running "xrandr" again, the output looks like this:

```
$ xrandr
Screen 0: minimum 320 x 200, current 2560 x 1024, maximum 8192 x 8192
VGA1 connected 1280x1024+0+0 (normal left inverted right x axis y axis) 338mm x 270mm
   1280x1024      60.0*+   75.0
   1280x960       60.0
   1152x864       75.0
   1024x768       75.1     70.1     60.0
   832x624        74.6
   800x600        72.2     75.0     60.3     56.2
   640x480        72.8     75.0     66.7     60.0
   720x400        70.1
LVDS1 connected 1280x800+1280+0 (normal left inverted right x axis y axis) 261mm x 163mm
   1280x800       60.0*+   50.0
   1024x768       85.0     75.0     70.1     60.0
   832x624        74.6
   800x600        85.1     72.2     75.0     60.3     56.2
   640x480        85.0     72.8     75.0     59.9
   720x400        85.0
   640x400        85.1
   640x350        85.1
```

Please note that i3 uses exactly the same API as xrandr does, so it will see only what you can see in xrandr.

See also [presentations] for more examples of multi-monitor setups.

7.2. Interesting configuration for multi-monitor environments

There are several things to configure in i3 which might be interesting if you have more than one monitor:

1. You can specify which workspace should be put on which screen. This allows you to have a different set of workspaces when starting than just 1 for the first monitor, 2 for the second and so on. See [workspace_screen].

2. If you want some applications to generally open on the bigger screen (MPlayer, Firefox, …), you can assign them to a specific workspace, see [assign_workspace].

3. If you have many workspaces on many monitors, it might get hard to keep track of which window you put where. Thus, you can use vim-like marks to quickly switch between windows. See [vim_like_marks].

4. For information on how to move existing workspaces between monitors, see [move_to_outputs].

8. i3 and the rest of your software world

8.1. Displaying a status line

A very common thing amongst users of exotic window managers is a status line at some corner of the screen. It is an often superior replacement to the widget approach you have in the task bar of a traditional desktop environment.

If you don't already have your favorite way of generating such a status line (self-written scripts, conky, …), then i3status is the recommended tool for this task. It was written in C with the goal of using as few syscalls as possible to reduce the time your CPU is woken up from sleep states. Because i3status only spits out text, you need to combine it with some other tool, like i3bar. See [status_command] for how to display i3status in i3bar.

Regardless of which application you use to display the status line, you want to make sure that it registers as a dock window using EWMH hints. i3 will position the window either at the top or at the bottom of the screen, depending on which hint the application sets. With i3bar, you can configure its position, see [i3bar_position].

8.2. Giving presentations (multi-monitor)

When giving a presentation, you typically want the audience to see what you see on your screen and then go through a series of slides (if the presentation is simple). For more complex presentations, you might want to have some notes which only you can see on your screen, while the audience can only see the slides.

8.2.1. Case 1: everybody gets the same output

This is the simple case. You connect your computer to the video projector, turn on both (computer and video projector) and configure your X server to clone the internal flat panel of your computer to the video output:

```
xrandr --output VGA1 --mode 1024x768 --same-as LVDS1
```

i3 will then use the lowest common subset of screen resolutions, the rest of your screen will be left untouched (it will show the X background). So, in our example, this would be 1024x768 (my notebook has 1280x800).

8.2.2. Case 2: you can see more than your audience

This case is a bit harder. First of all, you should configure the VGA output somewhere near your internal flat panel, say right of it:

```
xrandr --output VGA1 --mode 1024x768 --right-of LVDS1
```

Now, i3 will put a new workspace (depending on your settings) on the new screen and you are in multi-monitor mode (see [multi_monitor]).

Because i3 is not a compositing window manager, there is no ability to display a window on two screens at the same time. Instead, your presentation software needs to do this job (that is, open a window on each screen).

8.3. High-resolution displays (aka HIDPI displays)

See https://wiki.archlinux.org/index.php/HiDPI for details on how to enable scaling in various parts of the Linux desktop. i3 will read the desired DPI from the `Xft.dpi` property. The property defaults to 96 DPI, so to achieve 200% scaling, you'd set `Xft.dpi: 192` in `~/.Xresources`.

If you are a long-time i3 user who just got a new monitor, double-check that:

- You are using a scalable font (starting with "pango:") in your i3 config.

- You are using a terminal emulator which supports scaling. You could temporarily switch to gnome-terminal, which is known to support scaling out of the box, until you figure out how to adjust the font size in your favorite terminal emulator.