Hyprland Dynamic tiling Wayland compositor

START HERE

Mod + Q
Open a terminal

Mod + 1-9
Switch workspace

Mod + R
Open the launcher

Mod + Shift + 1-9
Send window to workspace

Mod + C
Close the window

Mod + V
Toggle floating

Mod + Arrows
Focus a neighbour

`hyprctl reload`
Reload config

Monitor an output
Workspace numbered or named
Window tiled, floating or grouped

CONCEPTS

Mod is Super in the stock config (local mainMod = "SUPER"). Every key below is a default bind from the example hyprland.lua and is yours to change.

Since 0.55 the config is Lua (hyprland.lua); the old hyprlang syntax is deprecated. A bind is hl.bind("SUPER + Q", hl.dsp.exec_cmd("kitty")), keys joined with ' + '.

Dispatchers are actions (hl.dsp.*) that you pass to hl.bind(), or to hl.dispatch() inside a Lua function. Lua binds run on the compositor loop, so never block in them.

LAUNCH AND CLOSE

Mod + Q Open terminal
The command is the terminal variable at the top of hyprland.lua (kitty in the example).

Mod + R Open app launcher
Runs the menu variable (hyprlauncher in the example).

Mod + E Open file manager

Mod + C
Close focused window
Asks the app to close politely; a hung app needs window.kill.

Mod + M Quit Hyprland
Runs hyprshutdown if installed, else the exit dispatcher. uwsm users should swap this for uwsm stop.

`hl.dsp.window.kill()`
Kill the window's process (SIGKILL)

FOCUS AND MOVE

Mod + Arrows
Focus left / right / up / down

`hl.dsp.window.move({ direction = "left" })`
Move window in a direction
The example config has no key for this; bind it yourself.

`hl.dsp.window.swap({ direction = "left" })`
Swap with the neighbour

`hl.dsp.window.cycle_next()`
Focus next window

`hl.dsp.focus({ last = true })`
Focus the previous window

FOCUS AND MOVE (cont.)

`hl.dsp.focus({ monitor = "right" })`
Focus another monitor

`hl.dsp.window.move({ monitor = "right" })`
Send window to another monitor

LAYOUT AND STATE

Mod + V Toggle floating

Mod + J
Toggle split direction
Dwindle layout only.

Mod + P Toggle pseudotiling

`hl.dsp.window.fullscreen()`
Toggle fullscreen
Bind it yourself; mode = "maximized" keeps gaps and bars.

`hl.dsp.window.resize({ x = 10, y = 0, relative = true })`
Resize by pixels

`hl.dsp.group.toggle()`
Make / dissolve a tabbed group

WORKSPACES

Mod + 1-9
Switch to workspace N
Mod+0 goes to workspace 10.

Mod + Shift + 1-9
Move window to workspace N
Mod+Shift+0 moves to workspace 10.

Mod + Scroll
Next / previous open workspace
mouse_down is e+1, mouse_up is e-1.

Mod + S
Toggle scratchpad workspace
Special workspace named magic.

Mod + Shift + S
Send window to scratchpad

`focus({ workspace = "name:<name>" })`
Go to a named workspace

`focus({ workspace = "previous" })`
Back to previous workspace

`focus({ workspace = "e+1" })`
Next open workspace
+1 is by ID, m+1 on this monitor, r+1 includes empty ones.

`hl.dsp.window.move({ workspace = "<ws>", follow = false })`
Send window without following

MOUSE

Mod + LMB Drag a window
Bound with { mouse = true } to window.drag().

Mod + RMB
Resize a window
Bound to window.resize(); LMB is mouse:272, RMB mouse:273.

BIND SYNTAX

`hl.bind(<keys>, <dispatcher>, { <flags> })`
Define a keybind
Keys are a string like "SUPER + SHIFT + Q"; a Lua function works in place of a dispatcher.

`hl.bind("SUPER + code:28", ...)`
Bind by keycode
Find codes and key names with wev; names are the XKB_KEY_ suffix.

`hl.unbind("SUPER + O")`
Remove a bind
Case-sensitive: must match the original string exactly.

`hl.dsp.exec_cmd(<cmd>)`
Run a shell command

`{ repeating = true }`
Repeat while held

`{ locked = true }`
Work on the lock screen

`{ release = true }`
Fire on key release

SUBMAPS

`hl.dsp.submap("<name>")`
Enter a submap

`hl.dsp.submap("reset")`
Leave to the global binds
Always give a submap an exit bind; otherwise hyprctl dispatch 'hl.dsp.submap("reset")' rescues you.

`hl.define_submap("<name>", function() ... end)`
Define the binds in a submap

MEDIA AND GLOBAL

`XF86AudioRaiseVolume`
Volume key bind with wpctl
Use { locked = true, repeating = true } so it works on the lock screen and when held.

HYPRCTL

`hyprctl reload`
Reload the config
hyprctl reload full-reset also switches between Lua and hyprlang; use sparingly.

HYPRCTL (cont.)

`hyprctl dispatch '<lua dispatcher>'`
Run a dispatcher now
Example: hyprctl dispatch 'hl.dsp.focus({ workspace = "3" })'.

`hyprctl clients`
List windows with properties

`hyprctl monitors`
List monitors
monitors all includes disabled ones.

`hyprctl activewindow`
Show the focused window
Handy for class and title when writing window rules.

`hyprctl workspaces`
List workspaces

`hyprctl binds`
List all registered binds

`hyprctl devices`
List keyboards, mice and switches

`hyprctl getoption <section.option>`
Read a config value

`hyprctl configerrors`
Show config parse errors

`hyprctl eval '<lua>'`
Run Lua in the compositor

`hyprctl -j <command>`
JSON output
Pipe to jq in scripts.

RECIPES

Core binds

```
local mainMod = "SUPER"
hl.bind(mainMod .. " + Q",
hl.dsp.exec_cmd("kitty"))
hl.bind(mainMod .. " + C",
hl.dsp.window.close())
hl.bind(mainMod .. " + V",
hl.dsp.window.float({ action =
"toggle" }))
```

Resize submap

```
hl.bind("ALT + R",
hl.dsp.submap("resize"))
hl.define_submap("resize",
function()
  hl.bind("right",
hl.dsp.window.resize({ x = 10,
y = 0, relative = true }),
{ repeating = true })
  hl.bind("escape",
hl.dsp.submap("reset"))
end)
```

Workspace keys in a loop

```
for i = 1, 9 do
  hl.bind(mainMod .. " + " ..
i, hl.dsp.focus({ workspace =
i }))
  hl.bind(mainMod .. " + SHIFT
+ " .. i,
hl.dsp.window.move({ workspace
= i }))
end
```