XMonad Config

Table Of Contents toc

About This Config

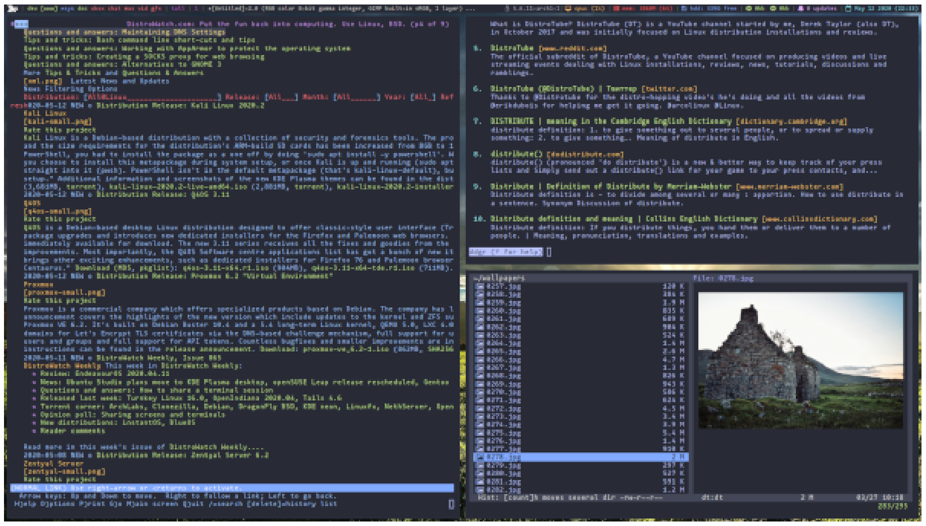

Figure 1: XMonad Scrot

Xmonad is a dynamically tiling X11 window manager that is written and configured in Haskell.

- XMonad Official Website: https://xmonad.org

This is the xmonad configuration of Derek Taylor (DistroTube)

- My YouTube: http://www.youtube.com/c/DistroTube
- My GitLab: http://www.gitlab.com/dwt1/

Keep in mind, that my configs are purposely bloated with examples of what you can do with xmonad. It is written more as a study guide rather than a config that you should download and use.

Imports

These are Haskell modules that we must import so that we can use their functions later in the config.

```
  -- Base
import XMonad
import System.Directory
import System.IO (hPutStrLn)
import System.Exit (exitSuccess)
import qualified XMonad.StackSet as W

    -- Actions
import XMonad.Actions.CopyWindow (kill1)
import XMonad.Actions.CycleWS (Direction1D(..), moveTo, shiftTo, WSType(..), nextScreen, pre
import XMonad.Actions.GridSelect
import XMonad.Actions.MouseResize
import XMonad.Actions.Promote
import XMonad.Actions.RotSlaves (rotSlavesDown, rotAllDown)
import XMonad.Actions.WindowGo (runOrRaise)
import XMonad.Actions.WithAll (sinkAll, killAll)
import qualified XMonad.Actions.Search as S

    -- Data
import Data.Char (isSpace, toUpper)
import Data.Maybe (fromJust)
import Data.Monoid
import Data.Maybe (isJust)
import Data.Tree
import qualified Data.Map as M

    -- Hooks
import XMonad.Hooks.DynamicLog (dynamicLogWithPP, wrap, xmobarPP, xmobarColor, shorten, PP(
```

```haskell
import XMonad.Hooks.EwmhDesktops  -- for some fullscreen events, also for xcomposite in obs
import XMonad.Hooks.ManageDocks (avoidStruts, docksEventHook, manageDocks, ToggleStruts(..))
import XMonad.Hooks.ManageHelpers (isFullscreen, doFullFloat)
import XMonad.Hooks.ServerMode
import XMonad.Hooks.SetWMName
import XMonad.Hooks.WorkspaceHistory

    -- Layouts
import XMonad.Layout.Accordion
import XMonad.Layout.GridVariants (Grid(Grid))
import XMonad.Layout.SimplestFloat
import XMonad.Layout.Spiral
import XMonad.Layout.ResizableTile
import XMonad.Layout.Tabbed
import XMonad.Layout.ThreeColumns

    -- Layouts modifiers
import XMonad.Layout.LayoutModifier
import XMonad.Layout.LimitWindows (limitWindows, increaseLimit, decreaseLimit)
import XMonad.Layout.Magnifier
import XMonad.Layout.MultiToggle (mkToggle, single, EOT(EOT), (??))
import XMonad.Layout.MultiToggle.Instances (StdTransformers(NBFULL, MIRROR, NOBORDERS))
import XMonad.Layout.NoBorders
import XMonad.Layout.Renamed
import XMonad.Layout.ShowWName
import XMonad.Layout.Simplest
import XMonad.Layout.Spacing
import XMonad.Layout.SubLayouts
import XMonad.Layout.WindowArranger (windowArrange, WindowArrangerMsg(..))
import qualified XMonad.Layout.ToggleLayouts as T (toggleLayouts, ToggleLayout(Toggle))
import qualified XMonad.Layout.MultiToggle as MT (Toggle(..))

   -- Utilities
import XMonad.Util.Dmenu
import XMonad.Util.EZConfig (additionalKeysP)
import XMonad.Util.NamedScratchpad
import XMonad.Util.Run (runProcessWithInput, safeSpawn, spawnPipe)
import XMonad.Util.SpawnOnce
```

Variables

It's nice to assign values to stuff that you will use more than once in the config. Setting values for things like font, terminal and editor means you only have to change the value here to make changes globally.

```haskell
myFont :: String
myFont = "xft:SauceCodePro Nerd Font Mono:regular:size=9:antialias=true:hinting=true"

myModMask :: KeyMask
myModMask = mod4Mask        -- Sets modkey to super/windows key

myTerminal :: String
myTerminal = "alacritty"    -- Sets default terminal

myBrowser :: String
myBrowser = "qutebrowser "  -- Sets qutebrowser as browser

myEmacs :: String
myEmacs = "emacsclient -c -a 'emacs' "  -- Makes emacs keybindings easier to type

myEditor :: String
myEditor = "emacsclient -c -a 'emacs' "  -- Sets emacs as editor
-- myEditor = myTerminal ++ " -e vim "    -- Sets vim as editor

myBorderWidth :: Dimension
myBorderWidth = 2           -- Sets border width for windows

myNormColor :: String
myNormColor   = "#282c34"   -- Border color of normal windows

myFocusColor :: String
myFocusColor  = "#46d9ff"   -- Border color of focused windows

windowCount :: X (Maybe String)
windowCount = gets $ Just . show . length . W.integrate' . W.stack . W.workspace . W.current
```

Autostart (The Startup Hook)

These are commands we want XMonad to execute on startup, such as running a compositor, setting our wallpaper, starting the emacs daemon, and starting our system tray and the applications that belong in it.

```haskell
myStartupHook :: X ()
myStartupHook = do
    spawnOnce "lxsession &"
    spawnOnce "picom &"
    spawnOnce "nm-applet &"
    spawnOnce "volumeicon &"
    spawnOnce "conky -c $HOME/.config/conky/xmonad.conkyrc"
    spawnOnce "trayer --edge top --align right --widthtype request --padding 6 --SetDockType
    spawnOnce "/usr/bin/emacs --daemon &" -- emacs daemon for the emacsclient
    -- spawnOnce "kak -d -s mysession &"  -- kakoune daemon for better performance
    -- spawnOnce "urxvtd -q -o -f &"      -- urxvt daemon for better performance
```

Uncomment only ONE of the following options for setting wallpaper.

```
spawnOnce "xargs xwallpaper --stretch < ~/.xwallpaper"  -- set last saved with xwallpaper
-- spawnOnce "/bin/ls ~/wallpapers | shuf -n 1 | xargs xwallpaper --stretch"  -- set random
-- spawnOnce "~/.fehbg &"  -- set last saved feh wallpaper
-- spawnOnce "feh --randomize --bg-fill ~/wallpapers/*"  -- feh set random wallpaper
-- spawnOnce "nitrogen --restore &"   -- if you prefer nitrogen to feh
setWMName "LG3D"
```

Gridselect

GridSelect displays items (programs, open windows, etc.) in a 2D grid and lets the user select from it with the cursor/hjkl keys or the mouse.

```
myColorizer :: Window -> Bool -> X (String, String)
myColorizer = colorRangeFromClassName
                  (0x28,0x2c,0x34) -- lowest inactive bg
                  (0x28,0x2c,0x34) -- highest inactive bg
                  (0xc7,0x92,0xea) -- active bg
                  (0xc0,0xa7,0x9a) -- inactive fg
                  (0x28,0x2c,0x34) -- active fg

-- gridSelect menu layout
mygridConfig :: p -> GSConfig Window
mygridConfig colorizer = (buildDefaultGSConfig myColorizer)
    { gs_cellheight   = 40
    , gs_cellwidth    = 200
    , gs_cellpadding  = 6
    , gs_originFractX = 0.5
    , gs_originFractY = 0.5
    , gs_font         = myFont
    }

spawnSelected' :: [(String, String)] -> X ()
spawnSelected' lst = gridselect conf lst >>= flip whenJust spawn
    where conf = def
                   { gs_cellheight   = 40
                   , gs_cellwidth    = 200
                   , gs_cellpadding  = 6
                   , gs_originFractX = 0.5
                   , gs_originFractY = 0.5
                   , gs_font         = myFont
                  }

myAppGrid = [ ("Audacity", "audacity")
                 , ("Deadbeef", "deadbeef")
                 , ("Emacs", "emacsclient -c -a emacs")
                 , ("Firefox", "firefox")
                 , ("Geany", "geany")
                 , ("Geary", "geary")
                 , ("Gimp", "gimp")
                 , ("Kdenlive", "kdenlive")
                 , ("LibreOffice Impress", "loimpress")
                 , ("LibreOffice Writer", "lowriter")
                 , ("OBS", "obs")
                 , ("PCManFM", "pcmanfm")
                 ]
```

Scratchpads

Allows to have several floating scratchpads running different applications. Import Util.NamedScratchpad and bind a key to namedScratchpadSpawnAction. In the example below, I have created named scratchpads for:

- alacritty – my terminal
- mocp – a terminal music player
- qalculate-gtk – a nice calculator

```
myScratchPads :: [NamedScratchpad]
myScratchPads = [ NS "terminal" spawnTerm findTerm manageTerm
                , NS "mocp" spawnMocp findMocp manageMocp
                , NS "calculator" spawnCalc findCalc manageCalc
                ]
  where
    spawnTerm  = myTerminal ++ " -t scratchpad"
    findTerm   = title =? "scratchpad"
    manageTerm = customFloating $ W.RationalRect l t w h
               where
                 h = 0.9
                 w = 0.9
                 t = 0.95 -h
                 l = 0.95 -w
    spawnMocp  = myTerminal ++ " -t mocp -e mocp"
    findMocp   = title =? "mocp"
    manageMocp = customFloating $ W.RationalRect l t w h
               where
                 h = 0.9
                 w = 0.9
                 t = 0.95 -h
                 l = 0.95 -w
    spawnCalc  = "qalculate-gtk"
    findCalc   = className =? "Qalculate-gtk"
    manageCalc = customFloating $ W.RationalRect l t w h
               where
                 h = 0.5
                 w = 0.4
                 t = 0.75 -h
                 l = 0.70 -w
```

Layouts

Defining the layouts that I want to have available.

```
--Makes setting the spacingRaw simpler to write. The spacingRaw module adds a configurable
mySpacing :: Integer -> l a -> XMonad.Layout.LayoutModifier.ModifiedLayout Spacing l a
mySpacing i = spacingRaw False (Border i i i i) True (Border i i i i) True

-- Below is a variation of the above except no borders are applied
-- if fewer than two windows. So a single window has no gaps.
mySpacing' :: Integer -> l a -> XMonad.Layout.LayoutModifier.ModifiedLayout Spacing l a
mySpacing' i = spacingRaw True (Border i i i i) True (Border i i i i) True

-- Defining a bunch of layouts, many that I don't use.
-- limitWindows n sets maximum number of windows displayed for layout.
-- mySpacing n sets the gap size around the windows.
tall     = renamed [Replace "tall"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ limitWindows 12
           $ mySpacing 8
           $ ResizableTall 1 (3/100) (1/2) []
magnify  = renamed [Replace "magnify"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ magnifier
           $ limitWindows 12
           $ mySpacing 8
           $ ResizableTall 1 (3/100) (1/2) []
monocle  = renamed [Replace "monocle"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ limitWindows 20 Full
floats   = renamed [Replace "floats"]
           $ smartBorders
           $ limitWindows 20 simplestFloat
grid     = renamed [Replace "grid"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ limitWindows 12
           $ mySpacing 8
           $ mkToggle (single MIRROR)
           $ Grid (16/10)
spirals  = renamed [Replace "spirals"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ mySpacing' 8
           $ spiral (6/7)
threeCol = renamed [Replace "threeCol"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ limitWindows 7
           $ ThreeCol 1 (3/100) (1/2)
threeRow = renamed [Replace "threeRow"]
           $ smartBorders
           $ addTabs shrinkText myTabTheme
           $ subLayout [] (smartBorders Simplest)
           $ limitWindows 7
           -- Mirror takes a layout and rotates it by 90 degrees.
           -- So we are applying Mirror to the ThreeCol layout.
           $ Mirror
           $ ThreeCol 1 (3/100) (1/2)
tabs     = renamed [Replace "tabs"]
           -- I cannot add spacing to this layout because it will
           -- add spacing between window and tabs which looks bad.
           $ tabbed shrinkText myTabTheme
tallAccordion  = renamed [Replace "tallAccordion"]
           $ Accordion
wideAccordion  = renamed [Replace "wideAccordion"]
           $ Mirror Accordion

-- setting colors for tabs layout and tabs sublayout.
myTabTheme = def { fontName            = myFont
                 , activeColor         = "#46d9ff"
                 , inactiveColor       = "#313846"
                 , activeBorderColor   = "#46d9ff"
                 , inactiveBorderColor = "#282c34"
                 , activeTextColor     = "#282c34"
                 , inactiveTextColor   = "#d0d0d0"
                 }

-- Theme for showWName which prints current workspace when you change workspaces.
myShowWNameTheme :: SWNConfig
myShowWNameTheme = def
    { swn_font              = "xft:Ubuntu:bold:size=60"
    , swn_fade              = 1.0
    , swn_bgcolor           = "#1c1f24"
    , swn_color             = "#ffffff"
    }

-- The layout hook
myLayoutHook = avoidStruts $ mouseResize $ windowArrange $ T.toggleLayouts floats
               $ mkToggle (NBFULL ?? NOBORDERS ?? EOT) myDefaultLayout
             where
               myDefaultLayout =     withBorder myBorderWidth tall
                                 ||| magnify
                                 ||| noBorders monocle
                                 ||| floats
                                 ||| noBorders tabs
                                 ||| grid
                                 ||| spirals
                                 ||| threeCol
                                 ||| threeRow
                                 ||| tallAccordion
                                 ||| wideAccordion
```

Workspaces

I have made my workspaces in xmobar "clickable." Clickable workspaces means the mouse can be used to switch workspaces. This requires *xdotool* to be installed. You need to use UnsafeStdInReader instead of simply StdInReader in your xmobar config so you can pass actions to it.

```
-- myWorkspaces = [" 1 ", " 2 ", " 3 ", " 4 ", " 5 ", " 6 ", " 7 ", " 8 ", " 9 "]
myWorkspaces = [" dev ", " www ", " sys ", " doc ", " vbox ", " chat ", " mus ", " vid ", "
myWorkspaceIndices = M.fromList $ zipWith (,) myWorkspaces [1..] -- (,) == \x y -> (x,y)

clickable ws = "<action=xdotool key super+"++show i++">"++ws++"</action>"
    where i = fromJust $ M.lookup ws myWorkspaceIndices
```

Managehook

Sets some rules for certain programs. Examples include forcing certain programs to always float, or to always appear on a certain workspace. Forcing programs to a certain workspace with a doShift requires xdotool if you are using clickable workspaces. You need the className or title of the program. Use xprop to get this info.

```
myManageHook :: XMonad.Query (Data.Monoid.Endo WindowSet)
myManageHook = composeAll
     -- 'doFloat' forces a window to float.  Useful for dialog boxes and such.
     -- using 'doShift ( myWorkspaces !! 7)' sends program to workspace 8!
     -- I'm doing it this way because otherwise I would have to write out the full
     -- name of my workspaces and the names would be very long if using clickable workspace
     [ className =? "confirm"         --> doFloat
     , className =? "file_progress"   --> doFloat
     , className =? "dialog"          --> doFloat
     , className =? "download"        --> doFloat
     , className =? "error"           --> doFloat
     , className =? "Gimp"            --> doFloat
     , className =? "notification"    --> doFloat
     , className =? "pinentry-gtk-2"  --> doFloat
     , className =? "splash"          --> doFloat
     , className =? "toolbar"         --> doFloat
     , title =? "Oracle VM VirtualBox Manager"  --> doFloat
     , title =? "Mozilla Firefox"     --> doShift ( myWorkspaces !! 1 )
     , className =? "brave-browser"   --> doShift ( myWorkspaces !! 1 )
     , className =? "qutebrowser"     --> doShift ( myWorkspaces !! 1 )
     , className =? "mpv"             --> doShift ( myWorkspaces !! 7 )
     , className =? "Gimp"            --> doShift ( myWorkspaces !! 8 )
     , className =? "VirtualBox Manager" --> doShift  ( myWorkspaces !! 4 )
     , (className =? "firefox" <&&> resource =? "Dialog") --> doFloat  -- Float Firefox Dia
     , isFullscreen -->  doFullFloat
     ] <+> namedScratchpadManageHook myScratchPads
```

Keybindings

I am using the Xmonad.Util.EZConfig module which allows keybindings to be written in simpler, emacs-like format. The Super/Windows key is 'M' (the modkey). The ALT key is 'M1'. SHIFT is 'S' and CTR is 'C'.

A FEW KEYBINDINGS	ASSOCIATED ACTION
MODKEY + RETURN	opens terminal (alacritty)
MODKEY + SHIFT + RETURN	opens run launcher (dmenu)
MODKEY + TAB	rotates through the available layouts

A FEW KEYBINDINGS	ASSOCIATED ACTION
MODKEY + SPACE	toggles fullscreen on/off (useful for watching videos)
MODKEY + SHIFT + c	closes window with focus
MODKEY + SHIFT + r	restarts xmonad
MODKEY + SHIFT + q	quits xmonad
MODKEY + 1-9	switch focus to workspace (1-9)
MODKEY + SHIFT + 1-9	send focused window to workspace (1-9)
MODKEY + j	windows focus down (switches focus between windows in stack)
MODKEY + k	windows focus up (switches focus between windows in stack)
MODKEY + SHIFT + j	windows swap down (swap windows in the stack)
MODKEY + SHIFT + k	windows swap up (swap the windows in the stack)
MODKEY + h	shrink window (decreases window width)
MODKEY + l	expand window (increases window width)
MODKEY + w	switches focus to monitor 1
MODKEY + e	switches focus to monitor 2
MODKEY + r	switches focus to monitor 3
MODKEY + period	switch focus to next monitor
MODKEY + comma	switch focus to prev monitor
MODKEY + SPACE	toggles fullscreen on/off (useful for watching videos)
MODKEY + t	force floating window back into tiling

```haskell
myKeys :: [(String, X ())]
myKeys =
    -- Xmonad
        [ ("M-C-r", spawn "xmonad --recompile")  -- Recompiles xmonad
        , ("M-S-r", spawn "xmonad --restart")    -- Restarts xmonad
        , ("M-S-q", io exitSuccess)              -- Quits xmonad

    -- Run Prompt
        , ("M-S-<Return>", spawn "dmenu_run -i -p \"Run: \"") -- Dmenu

    -- Other Dmenu Prompts
    -- In Xmonad and many tiling window managers, M-p is the default keybinding to
    -- launch dmenu_run, so I've decided to use M-p plus KEY for these dmenu scripts.
        , ("M-p c", spawn "~/dmscripts/dcolors")  -- pick color from our scheme
        , ("M-p e", spawn "~/dmscripts/dmconf")   -- edit config files
        , ("M-p i", spawn "~/dmscripts/dmscrot")  -- screenshots (images)
        , ("M-p k", spawn "~/dmscripts/dmkill")   -- kill processes
        , ("M-p m", spawn "~/dmscripts/dman")     -- manpages
        , ("M-p o", spawn "~/dmscripts/dmqute")   -- qutebrowser bookmarks/history
        , ("M-p p", spawn "passmenu")                    -- passmenu
        , ("M-p q", spawn "~/dmscripts/dmlogout") -- logout menu
        , ("M-p r", spawn "~/dmscripts/dmred")    -- reddio (a reddit viewer)
        , ("M-p s", spawn "~/dmscripts/dmsearch") -- search various search engines
```

```
-- Useful programs to have a keybinding for launch
    , ("M-<Return>", spawn (myTerminal))
    , ("M-b", spawn (myBrowser ++ " www.youtube.com/c/DistroTube/"))
    , ("M-M1-h", spawn (myTerminal ++ " -e htop"))

-- Kill windows
    , ("M-S-c", kill1)     -- Kill the currently focused client
    , ("M-S-a", killAll)   -- Kill all windows on current workspace

-- Workspaces
    , ("M-.", nextScreen)  -- Switch focus to next monitor
    , ("M-,", prevScreen)  -- Switch focus to prev monitor
    , ("M-S-<KP_Add>", shiftTo Next nonNSP >> moveTo Next nonNSP)       -- Shifts focuse
    , ("M-S-<KP_Subtract>", shiftTo Prev nonNSP >> moveTo Prev nonNSP)  -- Shifts focuse

-- Floating windows
    , ("M-f", sendMessage (T.Toggle "floats")) -- Toggles my 'floats' layout
    , ("M-t", withFocused $ windows . W.sink)  -- Push floating window back to tile
    , ("M-S-t", sinkAll)                       -- Push ALL floating windows to tile

-- Increase/decrease spacing (gaps)
    , ("C-M1-j", decWindowSpacing 4)         -- Decrease window spacing
    , ("C-M1-k", incWindowSpacing 4)         -- Increase window spacing
    , ("C-M1-h", decScreenSpacing 4)         -- Decrease screen spacing
    , ("C-M1-l", incScreenSpacing 4)         -- Increase screen spacing

-- Grid Select (CTR-g followed by a key)
    , ("C-g g", spawnSelected' myAppGrid)                 -- grid select favorite apps
    , ("C-g t", goToSelected $ mygridConfig myColorizer)  -- goto selected window
    , ("C-g b", bringSelected $ mygridConfig myColorizer) -- bring selected window

-- Windows navigation
    , ("M-m", windows W.focusMaster)  -- Move focus to the master window
    , ("M-j", windows W.focusDown)    -- Move focus to the next window
    , ("M-k", windows W.focusUp)      -- Move focus to the prev window
    , ("M-S-m", windows W.swapMaster) -- Swap the focused window and the master window
    , ("M-S-j", windows W.swapDown)   -- Swap focused window with next window
    , ("M-S-k", windows W.swapUp)     -- Swap focused window with prev window
    , ("M-<Backspace>", promote)      -- Moves focused window to master, others maintai
    , ("M-S-<Tab>", rotSlavesDown)    -- Rotate all windows except master and keep focu
    , ("M-C-<Tab>", rotAllDown)       -- Rotate all the windows in the current stack

-- Layouts
    , ("M-<Tab>", sendMessage NextLayout)           -- Switch to next layout
    , ("M-<Space>", sendMessage (MT.Toggle NBFULL) >> sendMessage ToggleStruts) -- Togg
```

```
-- Increase/decrease windows in the master pane or the stack
    , ("M-S-<Up>", sendMessage (IncMasterN 1))      -- Increase # of clients master pan
    , ("M-S-<Down>", sendMessage (IncMasterN (-1))) -- Decrease # of clients master pan
    , ("M-C-<Up>", increaseLimit)                   -- Increase # of windows
    , ("M-C-<Down>", decreaseLimit)                 -- Decrease # of windows

-- Window resizing
    , ("M-h", sendMessage Shrink)                   -- Shrink horiz window width
    , ("M-l", sendMessage Expand)                   -- Expand horiz window width
    , ("M-M1-j", sendMessage MirrorShrink)          -- Shrink vert window width
    , ("M-M1-k", sendMessage MirrorExpand)          -- Expand vert window width

-- Sublayouts
-- This is used to push windows to tabbed sublayouts, or pull them out of it.
    , ("M-C-h", sendMessage $ pullGroup L)
    , ("M-C-l", sendMessage $ pullGroup R)
    , ("M-C-k", sendMessage $ pullGroup U)
    , ("M-C-j", sendMessage $ pullGroup D)
    , ("M-C-m", withFocused (sendMessage . MergeAll))
    -- , ("M-C-u", withFocused (sendMessage . UnMerge))
    , ("M-C-/", withFocused (sendMessage . UnMergeAll))
    , ("M-C-.", onGroup W.focusUp')    -- Switch focus to next tab
    , ("M-C-,", onGroup W.focusDown')  -- Switch focus to prev tab

-- Scratchpads
-- Toggle show/hide these programs.  They run on a hidden workspace.
-- When you toggle them to show, it brings them to your current workspace.
-- Toggle them to hide and it sends them back to hidden workspace (NSP).
    , ("C-s t", namedScratchpadAction myScratchPads "terminal")
    , ("C-s m", namedScratchpadAction myScratchPads "mocp")
    , ("C-s c", namedScratchpadAction myScratchPads "calculator")

-- Set wallpaper with 'feh'. Type 'SUPER+F1' to launch sxiv in the wallpapers directory
-- Then in sxiv, type 'C-x w' to set the wallpaper that you choose.
    , ("M-<F1>", spawn "sxiv -r -q -t -o ~/wallpapers/*")
    , ("M-<F2>", spawn "/bin/ls ~/wallpapers | shuf -n 1 | xargs xwallpaper --stretch")
    --, ("M-<F2>", spawn "feh --randomize --bg-fill ~/wallpapers/*")

-- Controls for mocp music player (SUPER-u followed by a key)
    , ("M-u p", spawn "mocp --play")
    , ("M-u l", spawn "mocp --next")
    , ("M-u h", spawn "mocp --previous")
    , ("M-u <Space>", spawn "mocp --toggle-pause")

-- Emacs (CTRL-e followed by a key)
    -- , ("C-e e", spawn myEmacs)                 -- start emacs
    , ("C-e e", spawn (myEmacs ++ ("--eval '(dashboard-refresh-buffer)'")))   -- emacs 
    , ("C-e b", spawn (myEmacs ++ ("--eval '(ibuffer)'")))   -- list buffers
    , ("C-e d", spawn (myEmacs ++ ("--eval '(dired nil)'"))) -- dired
    , ("C-e i", spawn (myEmacs ++ ("--eval '(erc)'")))       -- erc irc client
    , ("C-e m", spawn (myEmacs ++ ("--eval '(mu4e)'")))      -- mu4e email
    , ("C-e n", spawn (myEmacs ++ ("--eval '(elfeed)'")))    -- elfeed rss
    , ("C-e s", spawn (myEmacs ++ ("--eval '(eshell)'")))    -- eshell
    , ("C-e t", spawn (myEmacs ++ ("--eval '(mastodon)'")))  -- mastodon.el
    -- , ("C-e v", spawn (myEmacs ++ ("--eval '(vterm nil)'"))) -- vterm if on GNU Emac
    , ("C-e v", spawn (myEmacs ++ ("--eval '(+vterm/here nil)'"))) -- vterm if on Doom 
    -- , ("C-e w", spawn (myEmacs ++ ("--eval '(eww \"distrotube.com\")'"))) -- eww bro
    , ("C-e w", spawn (myEmacs ++ ("--eval '(doom/window-maximize-buffer(eww \"distrotu
    -- emms is an emacs audio player. I set it to auto start playing in a specific dire
    , ("C-e a", spawn (myEmacs ++ ("--eval '(emms)' --eval '(emms-play-directory-tree \'

-- Multimedia Keys
    , ("<XF86AudioPlay>", spawn (myTerminal ++ "mocp --play"))
    , ("<XF86AudioPrev>", spawn (myTerminal ++ "mocp --previous"))
    , ("<XF86AudioNext>", spawn (myTerminal ++ "mocp --next"))
    , ("<XF86AudioMute>", spawn "amixer set Master toggle")
    , ("<XF86AudioLowerVolume>", spawn "amixer set Master 5%- unmute")
    , ("<XF86AudioRaiseVolume>", spawn "amixer set Master 5%+ unmute")
    , ("<XF86HomePage>", spawn "qutebrowser https://www.youtube.com/c/DistroTube")
    , ("<XF86Search>", spawn "dmsearch")
    , ("<XF86Mail>", runOrRaise "thunderbird" (resource =? "thunderbird"))
    , ("<XF86Calculator>", runOrRaise "qalculate-gtk" (resource =? "qalculate-gtk"))
    , ("<XF86Eject>", spawn "toggleeject")
    , ("<Print>", spawn "dmscrot")
    ]
-- The following lines are needed for named scratchpads.
      where nonNSP          = WSIs (return (\ws -> W.tag ws /= "NSP"))
            nonEmptyNonNSP  = WSIs (return (\ws -> isJust (W.stack ws) && W.tag ws /= "
```

Main

This is the "main" of XMonad. This where everything in our configs comes together and works.

```
main :: IO ()
main = do
    -- Launching three instances of xmobar on their monitors.
    xmproc0 <- spawnPipe "xmobar -x 0 $HOME/.config/xmobar/xmobarrc0"
    xmproc1 <- spawnPipe "xmobar -x 1 $HOME/.config/xmobar/xmobarrc2"
    xmproc2 <- spawnPipe "xmobar -x 2 $HOME/.config/xmobar/xmobarrc1"
    -- the xmonad, ya know...what the WM is named after!
```

```
xmonad $ ewmh def
    { manageHook          = myManageHook <+> manageDocks
    , handleEventHook    = docksEventHook
                           -- Uncomment this line to enable fullscreen support on things
                           -- This works perfect on SINGLE monitor systems. On multi-mon
                           -- it adds a border around the window if screen does not have
                           -- is to use a keybinding to toggle fullscreen noborders inst
                           -- <+> fullscreenEventHook
    , modMask            = myModMask
    , terminal           = myTerminal
    , startupHook        = myStartupHook
    , layoutHook         = showWName' myShowWNameTheme $ myLayoutHook
    , workspaces         = myWorkspaces
    , borderWidth        = myBorderWidth
    , normalBorderColor  = myNormColor
    , focusedBorderColor = myFocusColor
    , logHook = dynamicLogWithPP $ namedScratchpadFilterOutWorkspacePP $ xmobarPP
          -- the following variables beginning with 'pp' are settings for xmobar.
          { ppOutput = \x -> hPutStrLn xmproc0 x                          -- xmobar on m
                          >> hPutStrLn xmproc1 x                          -- xmobar on m
                          >> hPutStrLn xmproc2 x                          -- xmobar on m
          , ppCurrent = xmobarColor "#98be65" "" . wrap "[" "]"           -- Current wor
          , ppVisible = xmobarColor "#98be65" "" . clickable              -- Visible but
          , ppHidden = xmobarColor "#82AAFF" "" . wrap "*" "" . clickable -- Hidden work
          , ppHiddenNoWindows = xmobarColor "#c792ea" ""  . clickable     -- Hidden work
          , ppTitle = xmobarColor "#b3afc2" "" . shorten 60               -- Title of ac
          , ppSep =  "<fc=#666666> <fn=1>|</fn> </fc>"                    -- Separator c
          , ppUrgent = xmobarColor "#C45500" "" . wrap "!" "!"            -- Urgent work
          , ppExtras  = [windowCount]                                     -- # of window
          , ppOrder  = \(ws:l:t:ex) -> [ws,l]++ex++[t]                    -- order of th
          }
    } `additionalKeysP` myKeys
```