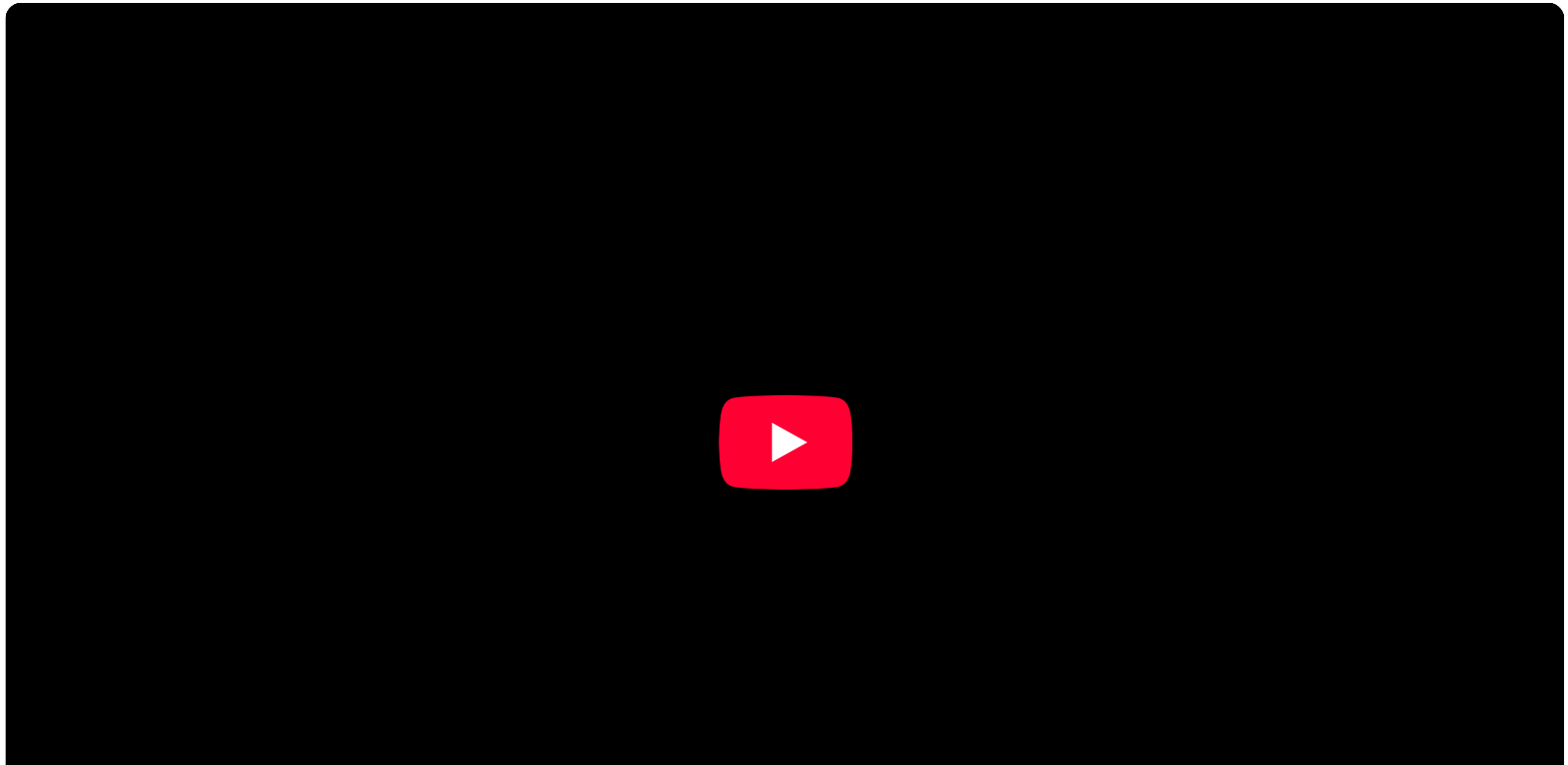
Ricing linux has never been easier | NixOS + stylix

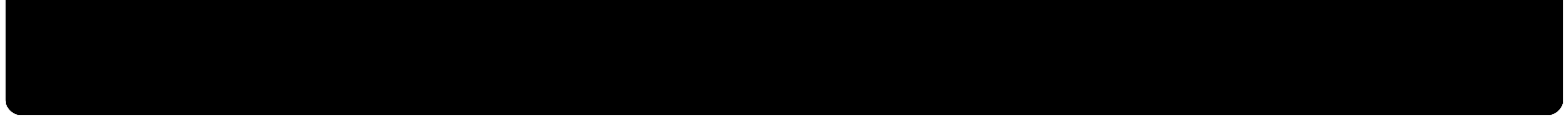
A post about auto-generating your colorscheme from a base16 palette

Vimjoyer

May 19, 2024 / 5 min read (1193 words)

This post is available in video form





Getting into linux ricing is undoubtedly difficult. And the sheer amount of information you have to learn about your programs and their special configuration types makes it a quite time consuming hobby to say the least.

But what if I told you, that if you are using NixOS, you could rice your entire system with just a cuouple of lines?

Sounds too good to be true, right? Well, introducing stylix, a pair of modules for NixOS and home-manager that will do all of the heavy lifting for you.

Because even though customizing your desktop is a highly personal and creative process, there are some aspects that can make it tedious or even unpleasant, such as applying matching color schemes, fonts, and opacities for every single program.

Let’s be realistic, most people simply want their programs to have the same backgound color and several matching accents.

Historically, this has led to the creation of dozens of templating programs and wonky bash scripts, which often turn editing and managing your desktops into a complete mess.

With NixOS, and in turn, stylix, this problem can be completely avoided, because every program in your configuration is broken into options, and it does not matter where these options are defined.

Meaning that by installing stylix we are essentially outsorcing the color option management to it, leaving us to worry about more importaint stuff like your preferred keybindings, layouts and so on.

And of course, none of these options will take effect, unless you choose to enable the programs explicitly. Meaning, your .config directory won't get cluttered with loads of dotfiles right away.

Installation

To install stylix, open your flake.nix file and include this github url in the inputs. Then make sure that you are passing your inputs to the outputs, and finally, include the module in your nixos setup.

```
{

  inputs = {
    # …
    stylix.url = "github:danth/stylix";
  };

  outputs = { nixpkgs, … }@inputs: {
    nixosConfigurations.default = nixpkgs.lib.nixosSystem {
      specialArgs = {inherit inputs;};
      modules = [
        ./configuration.nix
        inputs.stylix.nixosModules.stylix
      ];
    };
  };

}
```

At this point, you may be wondering why are we using a NixOS module instead of a home-manager one. And that is because the home-manager stylix module is automatically bundled within the NixOS one.

You can, of course, go ahead, and only import the home-manager module separately, but you will miss

out on some of the features like styling bootloaders or display managers.

```
{

  inputs = {
    # …
    stylix.url = “github:danth/stylix”;
  };

  outputs = { nixpkgs, home-manager, stylix, … }@inputs: {
    homeConfigurations.”«username»” = home-manager.lib.homeManagerConfiguration {
      pkgs = nixpkgs.legacyPackages.x86_64-linux;
      modules = [
        ./home.nix
        stylix.homeManagerModules.stylix
      ];
    };
  }

}
```

Just be careful not to include both of them at the same time, because you will run into option redeclaration issues.

Usage

Now that the module is imported, open your NixOS configuration, and let’s select a colorscheme.

Stylix uses a base16 colorscheme framework, which means that it only needs 16 colors to style your desktop.

There are plenty of base16 colorschemes available online, but the easiest way to get one is through the

`base16-schemes` package.

It contains a bunch of thmes under the /share/themes directory, and the easiest way to choose one is to simply build the package and check the result directory.

Afterwards, we can simply pass one of these schemes directly, or define them manually using a nix set.

Alternatively the theme can be autogenerated from a wallpaer, which can be defined with a stylix.image option.

Said wallpaper will also be applied automatically on supported desktop environments, but note that for some reason defining this option is mandatory, so even if you are going to use a custom theme, don't forget to include it in your configuration.

```
{ pkgs, … }:

{

  stylix.base16Scheme = "${pkgs.base16-schemes}/share/themes/gruvbox-dark-medium.yaml";

  # OR

  stylix.base16Scheme = {
    base00 = "282828";
    base01 = "3c3836";
    base02 = "504945";
    base03 = "665c54";
    base04 = "bdae93";
    base05 = "d5c4a1";
    base06 = "ebdbb2";
    base07 = "fbf1c7";
    base08 = "fb4934";
    base09 = "fe8019";
    base0A = "fabd2f";
```

```
    base0B = "b8bb26";
    base0C = "8ec07c";
    base0D = "83a598";
    base0E = "d3869b";
    base0F = "d65d0e";
  };

  # Don't forget to apply wallpaper

  stylix.image = ./my-cool-wallpaper.png;

}
```

That's pretty much all we need to do. And if you now rebuild your system, you'll notice that all the programs you enabled in your NixOS and home-manager configurations will be themed.

I personally didn't expect the generated btop++ theme to look this good, and was really surprised to see that even mangohud got a nice gruvbox theme after a rebuild.

There can, of course, be some exceptions, but given that many programs simply rely on GTK, QT or your terminal colors, most of your programs should be covered.

Advanced usage

Now to talk about some other options, starting with applying a custom cursor. Stylix allows us to define a cursor package and the name of the specific cursor inside it. Meaning, you can simply build any of the various cursor packages from nixpkgs, and check the available cursors by their directory names.

```
$ nix build nixpkgs#bibata-cursors
$ nix build nixpkgs#base16-schemes

$ cd result
```

```
$ nix run nixpkgs#eza — —tree —level 3
```

In this example, I am going to choose the `Bibata-Modern-Ice` cursor from the `bibata-cursors` package.

As I mentioned earlier, we can also define fonts here, with several options available, allowing you to select a serif, sans serif, and monospaced font.

```
{ pkgs, … }:

{
  # …

  stylix.cursor.package = pkgs.bibata-cursors;
  stylix.cursor.name = “Bibata-Modern-Ice”;

  stylix.fonts = {
    monospace = {
      package = pkgs.nerdfonts.override {fonts = [“JetBrainsMono”];};
      name = “JetBrainsMono Nerd Font Mono”;
    };
    sansSerif = {
      package = pkgs.dejavu_fonts;
      name = “DejaVu Sans”;
    };
    serif = {
      package = pkgs.dejavu_fonts;
      name = “DejaVu Serif”;
    };
  };

}
```

You obviously don’t want a monospaced font in your browser or a proportional font in your terminal, so stylix will automatically choose the appropriate variant from the selected options.

Other notable options include the ability to change font sizes and opacities for different types of programs, as well as the ability to select theme polarity to force-generate a light or dark theme.

```
{ pkgs, … }:

{
  # …

  stylix.fonts.sizes = {
    applications = 12;
    terminal = 15;
    desktop = 10;
    popups = 10;
  };

  stylix.opacity = {
    applications = 1.0;
    terminal = 1.0;
    desktop = 1.0;
    popups = 1.0;
  };

  stylix.polarity = “dark” # “light” or “either”

}
```

And to explore these and many many more options that Stylix provides, you can visit the Stylix Book website via the link in the description

After applying a few of these options, I launched several terminal emulators including Kitty, Foot, and

WezTerm to make nice screenshot. But now I cannot tell which one is which, so we can can clearly see that Stylix did an amazing job unifying their styles.

Even then, if you don't like some options set by stylix, it is absolutely trivial to disable specific program themes with this `` `stylix.targets.<program>.enable` `` option, or even cheery pick a specific option like I did in this example for hyprland active border color.

To do it you will want to use a `lib.mkForce function, and you can still access your selected colors, allowing you to easily swap yellow for red or magenta to green or any other color you prefer.

```
1  wayland.windowManager.hyprland.settings.general.”col.active_border” =
2    lib.mkForce “rgb(${config.stylix.base16Scheme.base0E})“;
```

Just don't forget to include the `` `lib` `` in paramer set at the top.