My NixOS Dotfiles Explained

2023-09-12 · #nix, #nixos, #home-manager, #dotfiles · 1629 words · 8 min ·
1 880 Views ·

▶ **My Development Workflow - This post is part of a series.**

In this post, we will just go over the basics of how we can configure our setup using a git repo, nix flakes and home-manager. I will go over how I structured my nix config.

Not an in-depth tutorial

Note this will not be an in-depth guide into NixOS/Home Manager itself. That could well be a series on its own. We will just go over the main ways I configure NixOS/Nix and why I do it the way I do. I recommend doing some reading and playing around and figuring out what works for you 😄

Introduction

We will be using two different ways to configure our system, NixOS via a `configuration.nix` to configure the machine itself. Which includes partitions, backups, docker anything that needs to be run system-wide. Anything that needs "sudo" permissions is configured via this expression. Note we will of course split our nix expressions into smaller modules. That will be re-used between multiple hosts.

The second way we will configure our system (most of our config in fact) will be `home-manager` . This is a Nix tool to configure things in userland. Such as dotfiles, and most of my apps. So for example here is where I define my neovim config, and various cli tools I use (like zoxide, fzf). [^1]

Git Repo

Typically NixOS can be configured using a file at `/etc/nixos/configuration.nix`, however, to edit this file you need sudo permissions. We also cannot easily track this file in Git. One of the main benefits of using NixOS/Nix is that we can use declaratively define our machine state, store that in a git repo and then re-use that across multiple devices.

Here is my git repo, when setting up my host mesmer it probably took about 20 minutes to setup my entire machine. Most of which was spent downloading, building, and evaluating nix derivations.

Flakes

We will also use Nix flakes alongside our git repo. I won't go into lots of details of what flakes are [^2]. But nix flakes seem to be the preferred way to define our nix configs. Flakes improve reproducibility by creating a lock file for our dependencies and also creating a project so it's easier for new people to navigate our Nix config.

A `flake.nix` file becomes a bit like `package.json` file in javascript land. Alongside this flake file, a lock file will be generated. Which will lock our dependencies to specific values. So if anyone else uses our flake they will get the same versions of the nix derivations that we have. This includes nixpkgs, so even when these are updated we tie them to a specific version.

Typically when we start using flakes initially they also go in `/etc/nixos/flake.nix`. Which causes similar problems as the configuration file. However, this is an easyish way to create a git repo in say `~/dotfiles` which is a flake and can be used to configure both our system and home-manager.

⚠ Experimental

Whilst lots of people in the Nix community have already adopted flakes. It's important to remember they are still an experimental feature and are likely to be the cause of breaking changes in the future. So use it with caution if you are worried about this sort of thing.

Getting Started

So we want to use a git repo and flakes to configure our system. Even before using Nix/NixOS, I was using a dotfiles repo. Which would be located in my home directory i.e `~/dotfiles` . So I kept the same location for my nix dotfiles as well.

Then to enable flakes let's add the following to `nix.settings.experimental-features = [ "nix-command" "flakes" ];` to our `configuration.nix` file. Then run `sudo nixos-rebuild switch` to rebuild our NixOS config.

Repo Structure

There are 100 different ways to structure our dotfiles repo. I ended up using this starter repo (standard version) and used Misterio77's main repo for inspiration.

Where my repo looks something like this (simplified):

```
.
├── flake.lock
├── flake.nix
├── home-manager
├── hosts
│   ├── curve
│   │   └── home.nix
│   ├── framework
│   │   ├── configuration.nix
│   │   ├── hardware-configuration.nix
│   │   ├── home.nix
│   │   ├── secrets.yaml
│   │   └── users
│   └── mesmer
│       ├── configuration.nix
│       ├── hardware-configuration.nix
│       ├── home.nix
│       ├── secrets.yaml
│       └── users
├── LICENSE.md
├── modules
├── nixos
├── overlays
├── pkgs
├── README.md
└── shell.nix
```

Some key bits are:

- `flake.nix` : The entry point into the dotfiles, defines all of our NixOS hosts and home-manager hosts, inputs etc
- `nixos` : Has most of the re-usable NixOS expressions
- `home-manager` : Has most of the re-usable home manager nix expressions
- `hosts` : After `flake.nix` these are the secondary entry points into the configuration for each host

flake.nix

Here is a simplified version of my `flake.nix` file:

```
{
  description = "My Nix Config";

  nixConfig = {
    experimental-features = [ "nix-command" "flakes" ];
  };

  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs/nixos-unstable";
    home-manager.url = "github:nix-community/home-manager";
  };
```

```
    outputs =
      { self
      , nixpkgs
      , home-manager
      , ...
      } @ inputs:
      {
        nixosConfigurations = {
          # Personal laptop
          framework = lib.nixosSystem {
            modules = [ ./hosts/framework/configuration.nix ];
            specialArgs = { inherit inputs outputs; };
          };
        };

        homeConfigurations = {
          # Laptops
          framework = lib.homeManagerConfiguration {
            modules = [ ./hosts/framework/home.nix ];
            pkgs = nixpkgs.legacyPackages.x86_64-linux;
            extraSpecialArgs = { inherit inputs outputs; };
          };

          curve = lib.homeManagerConfiguration {
            modules = [ ./hosts/curve/home.nix ];
            pkgs = nixpkgs.legacyPackages.x86_64-linux;
            extraSpecialArgs = { inherit inputs outputs; };
          };
        };
      };
  }
```

We have:

- `description` : For what the flake is, not important

- `inputs` : Other nix flakes I’m using as imports, such as home-manager to install the home-manager binary and nixpkgs to use unstable
- `output` : Which contains two main bits
    - `nixosConfiguration` : Uses `configuration.nix` as an entry point (in a specific host file)
    - `homeManagerConfiguration` : Uses `home.nix` as an entry point (in specific host file)
        - `curve` : You will notice this host only has home manager config because it is running Ubuntu so only uses home manager, which to be honest is good enough in my opinion

flake.lock

In our `flake.lock` file, we have something like this:

```
"nixpkgs_11": {
  "locked": {
    "lastModified": 1693844670,
    "narHash": "sha256-t69F2nBB8DNQUWHD809oJZJVE+23XBrth4QZuVd6IE0=",
    "owner": "nixos",
    "repo": "nixpkgs",
    "rev": "3c15feef7770eb5500a4b8792623e2d6f598c9c1",
    "type": "github"
  },
  "original": {
    "owner": "nixos",
    "ref": "nixos-unstable",
    "repo": "nixpkgs",
    "type": "github"
  }
},
```

Where we can see `rev` is a git sha. In this case, we are looking at a specific branch `ref : nixos-unstable` so we use the unstable channel, https://search.nixos.org/packages?channel=unstable&from=0&size=50&sort=relevance&type=packages&query=ag.

So if we don’t ever update our `flake.lock` we will forever be tied to this version of the unstable channel at that moment. Of course, that branch is getting updated multiple times a day. So to update our tools/apps etc. we need to update this lock file. We can do this by running `nix flake update`, in our dotfiles repo.

NixOS

Let’s take a look at how we configure a device using NixOS, so looking at my `hosts/framework/configuration.nix`:

```
{ inputs
, pkgs
, ...
}: {
  imports = [
    inputs.hardware.nixosModules.framework-12th-gen-intel
    inputs.nix-gaming.nixosModules.default
    inputs.hyprland.nixosModules.default

    ./hardware-configuration.nix
    ./users/haseeb

    ../../nixos/global
    ../../nixos/optional/backup.nix
    ../../nixos/optional/greetd.nix
    ../../nixos/optional/mullvad.nix
  ];

  # Enable networking
  networking = {
    networkmanager = {
      enable = true;
    };
    hostName = "framework";
  };

  boot = {
    loader = {
      systemd-boot.enable = true;
      efi.canTouchEfiVariables = true;
    };
    initrd.luks.devices = {
      root = {
        device = "/dev/disk/by-uuid/fc112246-8ce0-47c7-95e5-106be34e9501
        preLVM = true;
      };
    };
    kernelPackages = pkgs.linuxPackages_latest;
  };

  system.stateVersion = "23.05";
}
```

The main thing this acts as an entry point for our NixOS config (system-wide). This allows us to specify which packages we want to on different devices. Essentially all the imports are re-usable nix expressions, that can be shared across devices.

Taking a look at an optional expression say `greetd.nix` :

```
{
  services.greetd = {
    enable = true;
    settings = {
      initial_session = {
        command = "Hyprland";
        user = "haseeb";
      };
      default_session = {
        command = "initial_session";
      };
    };
  };
  environment.etc."greetd/environments".text = ''
    Hyprland
  '';
}
```

Here you can see we define a greetd service to run, which will run Hyprland when we log in (and which user to log us into).

Where we can find a full list of options here.

Home Manager

Similar to the above section, the `hosts/framework/home.nix` acts as the entry point to our home-manager config. Here is where most of my nix expression/config lies. This configures the home directory of a user. But can also be used to install and configure most of our tools and apps. Let’s take a look at an example file:

```
  { inputs
  , lib
  , pkgs
  , config
  , outputs
  , ...
  }: {
    imports =
      [
        inputs.nix-colors.homeManagerModule
        inputs.nixvim.homeManagerModules.nixvim

        ../../home-manager/desktops/hyprland

        ../../home-manager/shells/fish.nix
        ../../home-manager/terminals/foot.nix

        ../../home-manager/browsers/firefox.nix
        ../../home-manager/programs/cli
        ../../home-manager/editors/nvim
        ../../home-manager/programs/multiplexers/tmux.nix

        ../../home-manager/games
      ];

    colorscheme = inputs.nix-colors.colorSchemes.catppuccin-frappe;
    wallpaper = "~/dotfiles/home-manager/wallpapers/rainbow-nix.jpg";
    host = "framework";

    nix = {
      package = lib.mkDefault pkgs.nix;
      settings = {
        experimental-features = [ "nix-command" "flakes" "repl-flake" ];
        warn-dirty = false;
      };
    };

    systemd.user.startServices = "sd-switch";

    programs = {
      home-manager.enable = true;
      git.enable = true;
    };

    home = {
      username = lib.mkDefault "haseeb";
      homeDirectory = lib.mkDefault "/home/${config.home.username}";
      stateVersion = lib.mkDefault "23.05";
      sessionPath = [ "$HOME/.local/bin" ];
      sessionVariables = {
        TERMINAL = "alacritty";
        EDITOR = "nvim";
        BROWSER = "firefox";
      };
    };
}
```

You can see the example looks very similar to our `configuration.nix` file. I try to configure my tooling in home-manager first, if I can. We don't need sudo permissions to do this. The more that lives in my home directory the easier is it to backup. For example, we can take a look my `programs/fzf.nix` expression:

```
{
  programs.fzf = {
    enable = true;
    enableFishIntegration = true;
  };
}
```

We can find a set of home-manager options here. Not everything is here (especially nixvim).

That's It! I didn't go into lots of specifics in this article but more a general view of how I go about configuring my system. Hopefully, it made it more clear how I use NixOS. And even a bit of Nix on non-NixOS-based machines.

Appendix

- My dotfiles
- Misterio77's Repo
- home-manager options super useful

← PREV
TIL: How to Check if Another Option Is Set in Home Manager (Nix)

NEXT →
TIL: How to Use fzf.fish History Search