Flakes: NixOS and Home Manager migration

Jan 10, 2022 by Gabriel Volpe

nix nixos linux home-manager flakes

Table of contents

Introduction

I have recently migrated my entire NixOS and Home Manager (HM) configuration — including programs, services, dotfiles, etc — over to the new kid on the block: Nix flakes.

It was not as difficult as I thought it would be but there were a lot of things I had to figure out on my own or by asking

So let me tell you the important bits of this migration story in this short blog post ;)

How it started: NixOS

I initially had my NixOS configuration under the `system` directory and my Home Manager configuration under `home`, as you can see in the Github repo. So I decided to do the migration step by step, starting with the former.

I created a `flake.nix` file under `/etc/nixos` with the following content.

```
{
  description = "NixOS configuration";

  inputs.nixpkgs.url = "nixpkgs/nixos-unstable";

  outputs = inputs @ { self, nixpkgs }:
    let system = "x86_64-linux"; in {
      nixosConfigurations = {
        tongfang-amd = nixpkgs.lib.nixosSystem {
          inherit system;
          specialArgs = { inherit inputs; };
          modules = [
            ./machine/tongfang-amd
            ./configuration.nix
          ];
        };
      };
```

I could then build my system using this flake! Easy, right?

```
$ sudo nixos-rebuild switch --flake .#tongfang-amd
```

By default, `nixos-rebuild` expects the configuration under `/etc/nixos`. However, we can specify a different directory, as shown below.

```
$ sudo nixos-rebuild switch --flake '/home/gvolpe/workspace/nix-config#tongfa
```

It also turns out this flake can be built via `nix build`.

```
$ nix build .#nixosConfigurations.tongfang-amd.config.system.build.toplevel
$ sudo result/bin/switch-to-configuration switch
```

This means we can switch the system configuration from any directory by using either command!

That's handy if you keep all your configurations in a single directory and these are tracked by a version control system (VCS) such as git.

Next step: Home Manager

started creating a `flake.nix` under the `home` directory but I quickly realized having two different flakes for a single machine is not ideal.

However, since both the NixOS and HM configurations can be built from anywhere (no need to be under `/etc/nixos` and `$HOME/.config/nixpkgs`, respectively), I went with a single `flake.nix` where both the NixOS and HM configurations live (importing modules to make it more readable, of course).

A nice property of having a single flake, is that we can find out all the pinned versions by looking at the `flake.lock` file. We also get to manage everything from a single `nix flake` command.

One flake to rule them all!

So here's the only `flake.nix` that contains both NixOS and HM configurations.

```
{
  description = "Home Manager (dotfiles) and NixOS configurations";

  inputs = {
    nixpkgs.url = "nixpkgs/nixos-unstable";

    nurpkgs = {
```

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

    tex2nix = {
      url = github:Mic92/tex2nix/4b17bc0;
      inputs.utils.follows = "nixpkgs";
    };
  };

  outputs = inputs @ { self, nixpkgs, nurpkgs, home-manager, tex2nix }:
    let
      system = "x86_64-linux";
    in
    {
      homeConfigurations = (
        import ./outputs/home-conf.nix {
          inherit system nixpkgs nurpkgs home-manager tex2nix;
        }
      );

      nixosConfigurations = (
        import ./outputs/nixos-conf.nix {
          inherit (nixpkgs) lib;
          inherit inputs system;
        }
      );

      devShell.${system} = (
        import ./outputs/installation.nix {
          inherit system nixpkgs;
        }
      );
```

It consists of a set of inputs and a set of outputs.

To make things more readable, I moved the corresponding configurations to the `outputs` directory. So the NixOS configuration now lives under `outputs/nixos-conf.nix`, and so on.

You can skip this: installation shell

There is also a `devShell` with two packages that I use for a fresh installation for custom build script, but that's quite personal so feel free to skip this part.

```
{ system, nixpkgs }:

let
  pkgs = nixpkgs.legacyPackages.${system};
in
pkgs.mkShell {
  name = "installation-shell";
  buildInputs = with pkgs; [ wget s-tar ];
}
```

What's great is that I can enter this shell without even checking out the project.

```
$ nix develop github:gvolpe/nix-config
```

Home Manager flake output

recognized by Nix flakes, so when we try to display it, it shows "unknown" as a description but this will probably be supported in the future.

```
$ nix flake show | rg homeConfigurations
├───homeConfigurations: unknown
```

So what's in the `outputs/home-conf.nix`? A basic HM configuration might look as follows.

```
{ system, nixpkgs, nurpkgs, home-manager, ... }:

let
  username = "gvolpe";
  homeDirectory = "/home/${username}";
  configHome = "${homeDirectory}/.config";

  pkgs = import nixpkgs {
    inherit system;
    config.allowUnfree = true;
    config.xdg.configHome = configHome;
    overlays = [ nurpkgs.overlay ];
  };

  nur = import nurpkgs {
    inherit pkgs;
    nurpkgs = pkgs;
  };
in
{
  main = home-manager.lib.homeManagerConfiguration rec {
    inherit pkgs system username homeDirectory;
```

```
        inherit (pkgs) config lib stdenv;
      };
    };
  }
```

Where `./home.nix` is your usual Home Manager configuration. From there, it will more likely get more complicated, as you will always tweak one piece or another.

Mine looks very similar, except I have a few more overlays and two home configurations for two different displays. You can look at it directly on the Github repo to avoid repetition in here.

The most confusing part for me was the order of evaluation in Nix (which seems to have changed?). So the overlays I had defined in `home.nix` were no longer being picked up and I had to define them at the top level.

Also, I had to set the `config.xdg.configHome` manually when importing `nixpkgs`, which was before set in `home.nix` via the `xdg.enable = true;` attribute. I still haven't figured out the right way to do this so I'm setting it myself, but if you do know, I'd appreciate if you let me know in the comments.

`home-manager switch`. However, now I prefer to directly build the flake and run the activation script from its result.

```
$ nix build .#homeConfigurations.gvolpe-hdmi.activationPackage
$ result/activate
```

It is more verbose, though, so it is a good idea to have a script for this.

Flake outputs

These are all the flake outputs I currently have (you can query the repo directly).

```
$ nix flake show github:gvolpe/nix-config
github:gvolpe/nix-config/962a766ab98217aba249f2614592bd5513a267a9
├───devShell
│   └───x86_64-linux: development environment 'installation-shelbash'
├───homeConfigurations: unknown
└───nixosConfigurations
    ├───dell-xps: NixOS configuration
    └───tongfang-amd: NixOS configuration
```

So far, I'm liking the flakes experience, and I only have words of gratitude for the thousands of contributors who have taken the Nix ecosystem where it is today, and it keeps on getting closer to perfection every day!

ain't over yet. I'm still figuring things out and learning new stuff on a daily basis, so I'm sure there will be plenty of changes in the near future, specially considering that flakes are still marked as experimental.

Anyway, that's all I have to say. Thanks for stopping by and have a look at my Nix configuration files, perhaps something in there helps you get that missing piece in yours :)

Cheers, Gabriel.

Post a comment on Github

Awan posted at *Tue, 09 Aug 2022 09:02:38 GMT*

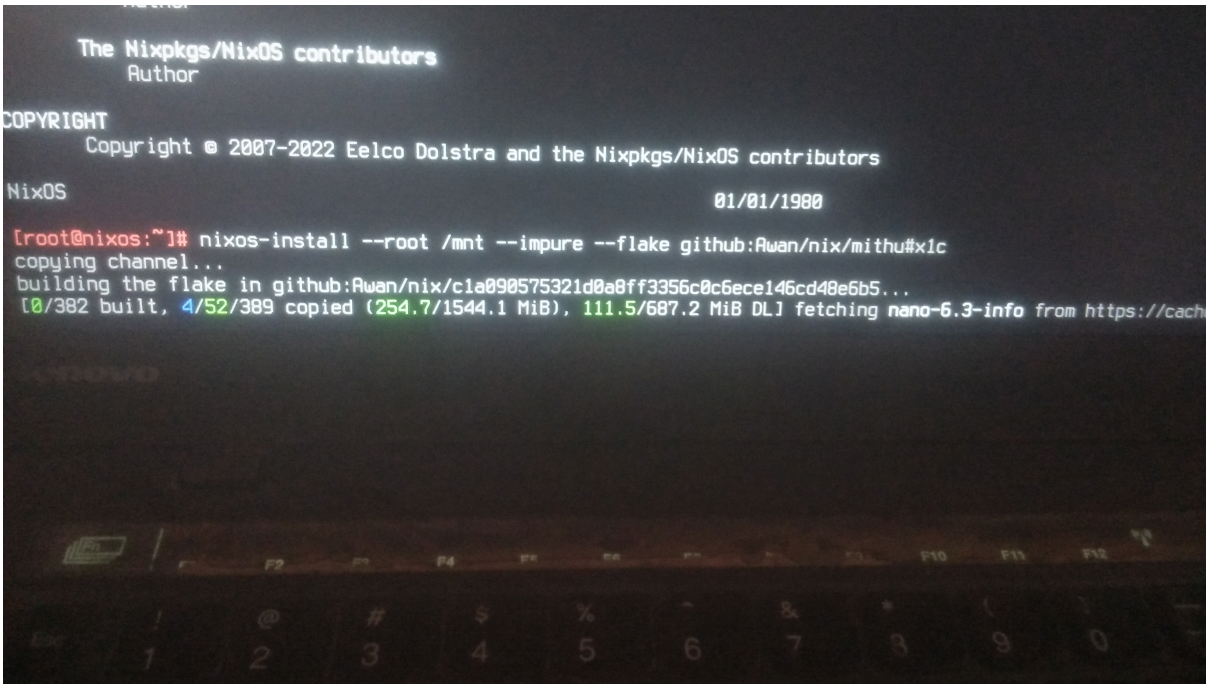
You can run home-manager switch --flake directly from any where and it will switch as well.

Awan posted at *Tue, 09 Aug 2022 09:29:32 GMT*

asciicast

@Awan I use `nix build` directly these days 😉 https://github.com/gvolpe/nix-config/blob/master/switch#L7

Awan posted at *Thu, 11 Aug 2022 07:07:00 GMT*



gvolpe posted at *Thu, 11 Aug 2022 07:41:20 GMT*

`nixos-install --flake` is very cool for fresh installations :)

I haven't had the need to perform one in a long time (only have one working laptop right now), but definitely something I'd recommend doing. This blog post is probably a bit dated, as that was written when I first migrated to flakes...

yeah, its cool. I tried it yesterday on my laptop as I was moving to unstable. It worked like a charm. All I had to do is clone my [repo](https://github.com/Awan/nix), create a new branch, stripped down it for fast installation and then pushed back to gh. I just added my encrypted drive UUID and then `nixos-install --flake github:Awan/nix/mithu#x1c`. mithu is the branch if you don't want to use default master/main branch...
On 11/08, Gabriel Volpe wrote: `nixos-install --flake` is very cool for fresh installations :) I haven't had the need to perform one in a long time (only have one working laptop right now), but definitely something I'd recommend doing. This blog post is probably a bit dated, as that was written when I first migrated to flakes... -- Reply to this email directly or view it on GitHub: #16 (comment) You are receiving this because you were mentioned. Message ID: ***@***.***>
Abdullah Khabir Cyber Security Analyst https://abdullah.today
C20F 2707 3025 2569 BAC5 534B 7820 6670 C19D 1580

Awan posted at *Fri, 12 Aug 2022 04:29:23 GMT*

Here you can use `sudo -E nixos-rebuild --flake .#hostname` and it will save you from activation next step ;-)

@Awan I think you got the wrong line, you're pointing to the Home Manager activation, you don't need `sudo` for that.

Awan posted at *Fri, 12 Aug 2022 08:12:15 GMT*

@gvolpe yeah, I just noticed it. That `rebuild_system` is what I was mentioning.

gvolpe posted at *Fri, 12 Aug 2022 08:30:03 GMT*

There's no activation there, though 🤔 It's just `sudo nixos-rebuild switch --flake .#tongfang-amd`