README.md

Nix Starter Config

This repo contains a few a simple nix flake templates for getting started with NixOS + home-manager.

What this provides

- Minimal version:
  - NixOS configuration on `nixos/configuration.nix`, accessible via `nixos-rebuild --flake .`
  - Home-manager configuration on `home-manager/home.nix`, accessible via `home-manager --flake .`
- Standard version:
  - Basic boilerplate for adding custom packages (under `pkgs`) and overlays (under `overlay`). Accessible on your system, home config, as well as `nix build .#package-name`.
  - Boilerplate for custom NixOS (`modules/nixos`) and home-manager (`modules/home-manager`) modules
  - NixOS and home-manager configurations from minimal, and they should also use your overlays and custom packages right out of the box.

Getting started

Assuming you have a basic NixOS booted up (either live or installed, anything works). Here's a link to the latest NixOS downloads, just for you.

Alternatively, you can totally use `nix` and `home-manager` on your existing distro (or even on Darwin). Install nix and follow along (just ignore the `nixos-*` commands).

What template to chose?

If this is your first trying flakes, or you're attempting to migrate your (simple) config to it; you should use the minimal version.

If you're here looking for inspiration/tips/good practices (and you already use flakes), or you're migrating a config that already has overlays and custom packages; try the standard version.

I like your funny words, magic man

Not sure what this all means?

Take a look at the learn hub on the NixOS website (scroll down to guides, the manuals, and the other awesome learning resources).

Learning the basics of what Nix (the package manager) is, how the Nix language works, and a bit of NixOS basics should get you up and running. Don't worry if it seems a little confusing at first. Get confortable with the basic concepts and come back here to get your feet wet, it's the best way to learn!

The repo

- Install git, if you haven't already.
- Create a repository for your config, for example:

```
cd ~/Documents
git init nix-config
cd nix-config
```

- Make sure you're running Nix 2.4+, and opt into the experimental `flakes` and `nix-command` features:

```
# Should be 2.4+
nix --version
export NIX_CONFIG="experimental-features = nix-command flakes"
```

- Get the template:

```
# For minimal version
nix flake init -t github:misterio77/nix-starter-config#minimal

# For standard version
nix flake init -t github:misterio77/nix-starter-config#standard
```

- If you want to use NixOS: add stuff you currently have on `/etc/nixos/` to `nixos` (usually `configuration.nix` and `hardware-configuration.nix`, when you're starting out).
  - The included file has some options you might want, specially if you don't have a configuration ready. Make sure you have generated your own `hardware-configuration.nix`; if not, just mount your partitions to `/mnt` and run: `nixos-generate-config --root /mnt`.
- If you want to use home-manager: add your stuff from `~/.config/nixpkgs` to `home-manager` (probably `home.nix`).
  - The included file is also a good starting point if you don't have a config yet.
- Take a look at `flake.nix`, making sure to fill out anything marked with FIXME (required) or TODO (usually tips or optional stuff you might want)
- `git add` and `git push` your changes! Or at least copy them somewhere if you're on a live medium.

Usage

- Run `sudo nixos-rebuild switch --flake .#hostname` to apply your system configuration.
  - If you're still on a live installation medium, run `nixos-install --flake .#hostname` instead, and reboot.
- Run `home-manager switch --flake .#username@hostname` to apply your home configuration.
  - If you don't have home-manager installed, try `nix shell nixpkgs#home-manager`.

And that's it, really! You're ready to have fun with your configurations using the latest and greatest nix3 flake-enabled command UX.

What next?

Adding more hosts or users

You can organize them by hostname and username on `nixos` and `home-manager` directories, be sure to also add them to `flake.nix`.

You can take a look at my (beware, here be reproductible dragons) configuration repo for ideas.

NixOS makes it easy to share common configuration between hosts (you might want to create a common directory for these), while keeping everything in sync. home-manager can help you sync your environment (from editor to WM and everything in between) anywhere you use it. Have fun!

User password and secrets

You have basically two ways of setting up default passwords:

- By default, you'll be prompted for a root password when installing with `nixos-install`. After you reboot, be sure to add a password to your own account and lock root using `sudo passwd -l root`.
- Alternatively, you can specify `initialPassword` for your user. This will give your account a default password, be sure to change it after rebooting! If you do, you should pass `--no-root-passwd` to `nixos-install`, to skip setting a password on the root account.

If you don't want to set your password imperatively, you can also use `passwordFile` for safely and declaratively setting a password from a file outside the nix store.

There's also more advanced options for secret management, including some that can include them (encrypted) into your config repo and/or nix store, be sure to check them out if you're interested.

Dotfile management with home-manager

Besides just adding packages to your environment, home-manager can also manage your dotfiles. I strongly recommend you do, it's awesome!

For full nix goodness, check out the home-manager options with `man home-configuration.nix`. Using them, you'll be able to fully configure any program with nix syntax and its powerful abstractions.

Alternatively, if you're still not ready to rewrite all your configs to nix syntax, there's home-manager options (such as `xdg.configFile`) for including files from your config repository into your usual dot directories. Add your existing dotfiles to this repo and try it out!

Try opt-in persistance

You might have noticed that there's impurity in your NixOS system, in the form of configuration files and other cruft your system generates when running. What if you change them in a whim to get something working and forget about it? Boom, your system is not fully reproductible anymore.

You can instead fully delete your `/` and `/home` on every boot! Nix is okay with a empty root on boot (all you need is `/boot` and `/nix`), and will happily reapply your configurations.

There's two main approaches to this: mount a `tmpfs` (RAM disk) to `/`, or (using a filesystem such as btrfs or zfs) mount a blank snapshot and reset it on boot.

For stuff that can't be managed through nix (such as games downloaded from steam, or logs), use impermanence for mounting stuff you to keep to a separate partition/volume (such as `/nix/persist` or `/persist`). This makes everything vanish by default, and you can keep track of what you specifically asked to be kept.

Here's some awesome blog posts about it:

- Erase your darlings
- Encrypted BTRFS with Opt-In State on NixOS
- NixOS: tmpfs as root and tmpfs as home

Adding custom packages

Something you want to use that's not in nixpkgs yet? You can easily build and iterate on a derivation (package) from this very repository.

Create a folder with the desired name inside `pkgs`, and add a `default.nix` file containing a derivation. Be sure to also `callPackage` them on `pkgs/default.nix`.

You'll be able to refer to that package from anywhere on your home-manager/nixos configurations, build them with `nix build .#package-name`, or bring them into your shell with `nix shell .#package-name`.

See the manual for some tips on how to package stuff.

Adding overlays

Found some outdated package on nixpkgs you need the latest version of? Perhaps you want to apply a patch to fix a behaviour you don't like? Nix makes it easy and manageble with overlays!

Use the `overlay/default.nix` file for this.

If you're creating patches, you can keep them on the `overlay` folder as well.

See the wiki article to see how it all works.

Adding your own modules

Got some configurations you want to create an abstraction of? Modules are the answer. These awesome files can expose *options* and implement *configurations* based on how the options are set.

Create a file for them on either `modules/nixos` or `modules/home-manager`. Be sure to also add them to the listing at `modules/nixos/default.nix` or `modules/home-manager/default.nix`.

See the wiki article to learn more about them.

Troubleshooting / FAQ

Please let me know any questions or issues you face with these templates, so I can add more info here!

I'm trying to set nixpkgs.config options (e.g. allowUnfree), but they won't work!

We instantiate nixpkgs and pass it to NixOS, to make the flake a bit simpler and ensure both configs have the same nixpkgs instance, this has the drawback of breaking nixpkgs configuration through in your NixOS config files.

This is a design choice I made based on the `homeManagerConfiguration` interface, that requires a `pkgs` instance anyway. If you prefer to set them modularly, you'll have to remove the `pkgs` argument. If you use overlays, you'll have to pass them into your configuration (through specialArgs) or set `nixpkgs.overlays` right on the flake; in this case, you might prefer to use home-manager as a NixOS module and set `home-manager.useGlobalPkgs = true`.

Nix says my repo files don't exist, even though they do!

Nix flakes only see files that git is currently tracked, so just `git add .` and you should be good to go. Files on `.gitignore`, of course, are invisible to nix - this is to guarantee your build won't depend on anything that is not on your repo.

Contributors 2

Languages

- **Nix** 100.0%