NixOS & Flakes

Roy Långsjö

2026-04-14

OtaNix ry

Structure

1. What are flakes?
2. The problem with channels
3. Other flake quirks
4. NixOS + HM config with flakes

Flakes

Inputs & outputs

```nix
{
  description = "My flake";
  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs";
  };
  outputs = { nixpkgs, ... }@inputs: {
    packages = { ... };
    nixosConfigurations = { ... };
    devShells = { ... };
  };
}
```

- Need the experimental features `nix-commmand` and `flakes` enabled
- A flake.nix defines `inputs` and `outputs`
  - Think of it like a function
- `inputs` are other flakes/Nix code[1] that get fetched and passed to your function
  - `man nix3-flake` for input types
- The `outputs` block uses those `inputs` to define *what* it outputs[2]
- `flake.nix` is an *entrypoint* to your other Nix code, keep it simple

[1] Usually flakes or Nix code, but can be anything

[2] Output schema: https://wiki.nixos.org/wiki/Flakes#Output_schema

Simple flake.nix

```nix
{
  inputs.nixpkgs.url = "github:NixOS/nixpkgs/nixos-unstable";
  outputs = { nixpkgs, self }@inputs:
    let
      system = "x86_64-linux";
      pkgs = nixpkgs.legacyPackages.${system};
    in
    {
      packages.${system}.default = pkgs.hello; # nix run
      nixosConfigurations.my-hostname = nixpkgs.lib.nixosSystem {
        modules = [ ./configuration.nix ]; # nixos-rebuild --flake .
      };
      devShells.${system}.default = pkgs.mkShell {
        packages = [ pkgs.python3 ]; # nix develop
      };
    };
}
```

The nix3 CLI

```
# nix2 -> nix3 'equivalent' commands
nix-shell -p hello  -> nix shell nixpkgs#hello
nix-shell           -> nix develop
nix-store           -> nix store
nix-build           -> nix build
nix-instantiate     -> nix eval
nixos-rebuild       -> nixos-rebuild --flake <flakeref>

# other commands
nix run nixpkgs#hello # runs the 'hello' package's main program
nix flake <init|update|lock|show|metadata> # working with flake.nix/flake.lock
nix repl # read eval print loop
nix registry # flake registry
```

The problem with channels

Reproducibility... mostly

```
{ pkgs ? import <nixpkgs> { }, }:
pkgs.mkShell {
  packages = [
    pkgs.python3 # 3.4? 3.9? 3.15?
  ];
}
```

- You'll get the same packages, but no guarantee of version
- Works fine today, might fail tomorrow
- Can rollback, *but only* if you don't garbage collect your previous channels

flake.lock

```
{
"nixpkgs": {
  "locked": {
    "lastModified": 1775710090,
    "narHash": "sha256-ar3rofg+...",
    "owner": "NixOS",
    "repo": "nixpkgs",
    "rev": "4c1018dae018162ec87...",
    "type": "github"
  },
  "original": {
    "owner": "NixOS",
    "ref": "nixos-unstable",
    "repo": "nixpkgs",
    "type": "github"
  }
}
```

- Each input (and its inputs) gets an entry in the lockfile
- Every type of input will at least store the hash of the content
- Git type inputs will also get the exact commit
- Importantly the `flake.lock` file is part of your repo/project, and should be tracked in a VCS
- Everyone will have the same version of everything
- Need to roll back? Just revert the update with your VCS
- Other pinning solutions such as `npins`, `niv` can also do this

Other flake stuff

Purity

```nix
# flake.nix
{
  outputs = _: {
    system = builtins.currentSystem;
    time = builtins.currentTime;
    path = import /absolute/path.nix;
  };
}
```

```
$ nix eval .#system
error: attribute 'currentSystem' missing
$ nix eval .#time
error: attribute 'currentTime' missing
$ nix eval .#path
error: access to absolute path '/absolute'
is forbidden in pure evaluation mode (use
'--impure' to override)
```

- Flakes enforce purity
  - No impure builtins
  - No paths outside the flake[1]
  - Bypass with `--impure`
- Copies the whole flake to the store when it has changed
  - Big repo → slow copies
  - Secrets in repo → world readable in `/nix/store`
  - Unavoidable even with `--impure`

[1] 'Flake' refers to a filesystem tree with a `flake.nix` at its root

Git awareness

```
# flake.nix
{
  outputs = _: {
    myFile = import ./myfile.nix;
  };
}
```

```
$ ls -a
.  ..  flake.nix  .git  myfile.nix
$ git ls-files
flake.nix
$ nix eval .#myFile
error: path '/nix/store/
fvqazvkdwxdnr2hdhdyqp481lhk7ii8x-source/
myfile.nix' does not exist
$ nix eval path:.#myFile
"hello from file"
```

- When part of a Git repo, only tracked files are visible to flakes
- `.gitignore` build artifacts and secrets → not copied to store
- Forget to track a new file → invisible to flakes
- Bypass Git awareness by using explicit `path:` type flakeref

NixOS configuration with flakes

nixpkgs.lib.nixosSystem

- The function that creates a NixOS system
- Only available via the Nixpkgs flake
- The Nixpkgs you call this from becomes your system Nixpkgs

```
{
  outputs = { nixpkgs, ... }@inputs: {
    nixosConfigurations.<hostname> = nixpkgs.lib.nixosSystem {
      # passes `inputs` to all modules as arg
      specialArgs = { inherit inputs; };

      # Modules that make up your configuration
      # Can also be a single module that imports the rest
      modules = [ ./configuration.nix ];
    };
  };
};
```

Demo:
Migrating a NixOS + HM config to use flakes