Nix Reproducible packages

START HERE

`nix run nixpkgs#hello`
Run a package once

`nix shell nixpkgs#jq nixpkgs#curl`
Shell with tools on PATH

`nix search nixpkgs <term>`
Find a package

`nix develop`
Enter the project dev shell

`nix build`
Build, result symlink

`nix flake init`
Create a flake.nix

`nix flake update`
Bump the lock file

`nix store gc`
Free unused store paths

flake.nix inputs and outputs → lock → **flake.lock** pinned revisions → evaluate → **Derivation** .drv build recipe → build → **/nix/store** hash-addressed path

CONCEPTS

The nix command and flakes are experimental. Enable them once: experimental-features = nix-command flakes in ~/.config/nix/nix.conf.

An installable is flakeref#attr, e.g. nixpkgs#hello or .#myapp. A bare . means the flake in the current directory. Flakes only see files tracked by git: git add new files.

Nothing in /nix/store is ever modified. Profiles and result links are just symlinks into it, which makes rollback cheap.

RUN AND TRY

`nix run <installable> [-- <args>]`
Run an app from a flake
Arguments after -- go to the program.

`nix shell <installable>...`
Shell with packages on PATH

`nix shell <installable> -c <cmd>`
Run one command with the package

`nix run nixpkgs#<pkg>`
Run a nixpkgs package

`nix run .`
Run this flake's default app

`nix run github:<owner>/<repo>`
Run a flake from GitHub

`nix run nixpkgs/<branch>#<pkg>`
Use a specific nixpkgs branch

`nix shell --ignore-environment <installable>`
Shell with only these tools

`nix bundle <installable>`
Pack an app to run without Nix

BUILD

`nix build`
Build default package of ./flake
Creates a ./result symlink to the output.

`nix build <installable>`
Build a given installable

`nix build .#<name>`
Build a named package of this flake

`nix build --no-link --print-out-paths`
Build and print store path only

`nix build -L`
Show full build logs

BUILD (cont.)

`nix build --rebuild <installable>`
Rebuild to check determinism

`nix log <installable>`
Show a build log

`nix path-info -rsSh <installable>`
Closure with sizes

`nix why-depends <a> <b>`
Why a depends on b

DEVELOP

`nix develop`
Shell with the dev environment
Uses devShells.<system>.default, then packages.<system>.default.

`nix develop .#<name>`
Use a named devShell

`nix develop -c <cmd>`
Run a command in that environment

`nix develop nixpkgs#<pkg>`
Build environment of a package

`nix develop --impure`
Allow env vars and mutable paths

`nix develop --build`
Run a single build phase

`nix print-dev-env`
Print the environment as shell code

`nix fmt`
Format with the flake's formatter

`direnv: use flake`
Load the dev shell via nix-direnv
Put it in .envrc; needs direnv and nix-direnv.

FLAKES

`nix flake init`
Flake from the default template

`nix flake init -t <template>`
Flake from a given template

`nix flake new <dir>`
New flake in a directory

`nix flake show [<flakeref>]`
List a flake's outputs

`nix flake check`
Evaluate and run flake checks

`nix flake update`
Update all inputs in the lock

`nix flake update <input>`
Update a single input

FLAKES (cont.)

`nix flake lock`
Create missing lock entries

`nix flake metadata`
Show inputs and revisions

`nix flake lock --override-input <in> <ref>`
Override an input temporarily

`nix flake prefetch <flakeref>`
Fetch a flake into the store

`nix flake clone <flakeref> --dest <dir>`
Clone a flake's repo

Flake references

Form	Example
`Registry`	nixpkgs, nixpkgs/nixos-unstable
`GitHub`	github:NixOS/nixpkgs/<ref>
`Git`	git+https://<host>/<repo>?ref=
`Path`	./dir, path:/abs/dir
`Tarball`	https://<host>/<f>.tar.gz

PROFILE

`nix profile add <installable>`
Install into your profile
Called nix profile install in older releases.

`nix profile list`
List installed packages

`nix profile remove <name>`
Uninstall a package

`nix profile upgrade --all`
Upgrade installed packages

`nix profile rollback`
Back to the previous generation

`nix profile history`
List profile generations

`nix profile diff-closures`
Changes between generations

`nix profile wipe-history`
Delete old generations

SEARCH AND INSPECT

`nix search nixpkgs <regex>`
Search packages by name or description

`nix eval <installable>`
Evaluate and print an attribute

`nix eval --expr '<expr>'`
Evaluate a Nix expression

`nix eval --raw <installable>.<attr>`
Print a string without quotes

`nix repl`
Interactive evaluator

SEARCH AND INSPECT (cont.)

`nix repl nixpkgs`
Repl with nixpkgs loaded

`:l <flakeref>`
Load a flake or file in the repl
Also :r reloads, :q quits, :t shows a type.

`nix registry list`
Show flake registry aliases

`nix registry pin <name>`
Pin an alias to its revision

STORE AND CLEANUP

`nix store gc`
Delete unreachable store paths

`nix store delete <path>`
Delete a given path

`nix store optimise`
Hard-link identical files

`nix store diff-closures <a> <b>`
Compare two closures

`nix copy --to <store> <installable>`
Copy a closure to another store

`nix config show`
Show effective settings

`--offline`
Do not use substituters

`--refresh`
Re-fetch cached tarballs and refs

RECIPES

Minimal dev shell (flake.nix)

```
{
  inputs.nixpkgs.url =
    "github:NixOS/nixpkgs";
  outputs = { nixpkgs, ... }: {
    devShells.x86_64-
linux.default =
      nixpkgs.legacyPackages
        .x86_64-linux.mkShell {
          packages = [ ];
        };
  };
}
```

Try a tool without installing

```
nix search nixpkgs ripgrep
nix shell nixpkgs#ripgrep -c rg
--version
nix run nixpkgs#cowsay -- hi
```

Pin and update one input

```
nix flake metadata
nix flake update nixpkgs
git diff flake.lock
nix flake check
```

Reclaim disk space

```
nix profile wipe-history
nix store gc
nix store optimise
```