Introduction to Flakes

The flakes experimental feature is a major development for Nix, it introduces a policy for managing dependencies between Nix expressions, it improves reproducibility, composability and usability in the Nix ecosystem. Although it's still an experimental feature, flakes have been widely used by the Nix community.[1]

Flakes is one of the most significant changes the nix project has ever seen.[2]

In simple terms, if you've worked with some JavaScript/Go/Rust/Python, you should be familiar with files like `package.json` / `go.mod` / `Cargo.toml` / `pyproject.toml` . In these programming languages, these files are used to describe the dependencies between software packages and how to build projects.

Similarly, the package managers in these programming languages also use files like `package-lock.json` / `go.sum` / `Cargo.lock` / `poetry.lock` to lock the versions of dependencies, ensuring the reproducibility of projects.

Flakes borrow ideas from these package managers to enhance the reproducibility, composability, and usability of the Nix ecosystem.

Flakes introduce `flake.nix` , similar to `package.json` , to describe the dependencies between Nix packages and how to build projects. Additionally, it provides `flake.lock` , akin to `package-lock.json` , to lock the versions of dependencies, ensuring project reproducibility.

On the other hand, Flakes experimental features did not break Nix's original design at the user level. The two new files `flake.nix` / `flake.lock` introduced by Flakes are just a wrapper for other Nix configurations. In the following chapters, we will see that Flakes features provide a new and more convenient way to manage the dependencies between Nix expressions based on Nix's original design.

A Word of Caution about Flakes caution

The benefits of Flakes are evident, and the entire NixOS community has embraced it wholeheartedly. Currently, more than half of the users utilize Flakes[3], providing

assurance that Flakes will not be deprecated.

⚠️ However, it's important to note that **Flakes is still an experimental feature**. Some issues persist, and there is a possibility of introducing breaking changes during the stabilization process. The extent of these breaking changes remains uncertain.

Overall, I strongly recommend everyone to use Flakes, especially since this book revolves around NixOS and Flakes. However, it's crucial to be prepared for potential problems that may arise due to forthcoming breaking changes.

When Will Flakes Be Stabilized?

I delved into some details regarding Flakes:

- [RFC 0136] A Plan to Stabilize Flakes and the New CLI Incrementally: A plan to incrementally stabilize Flakes and the new CLI, merged.
- CLI stabilization effort: An issue tracking the progress of the New CLI stabilization effort.
- Why Are Flakes Still Experimental? - NixOS Discourse: A post discussing why Flakes are still considered experimental.
- Flakes Are Such an Obviously Good Thing - Graham Christensen: An article emphasizing the advantages of Flakes while suggesting areas for improvement in its design and development process.
- teaching Nix 3 CLI and Flakes #281 - nix.dev: An issue about "Teaching Nix 3 CLI and Flakes" in nix.dev, and the conclusion is that we should not promote unstable features in nix.dev.
- Draft: 1-year Roadmap - NixOS Foundation: A roadmap provided by the NixOS Foundation, which includes plans regarding the stabilization of Flakes.

After reviewing these resources, it seems that Flakes may be(or may not...) stabilized within two years, possibly accompanied by some breaking changes.

The New CLI and the Classic CLI

Nix introduced two experimental features, `nix-command` and `flakes`, in the year 2020. These features bring forth a new command-line interface (referred to as the New CLI), a standardized Nix package structure definition (known as the Flakes feature), and features

like `flake.lock`, similar to version lock files in cargo/npm. Despite being experimental as of February 1, 2024, these features have gained widespread adoption within the Nix community due to their significant enhancement of Nix capabilities.

The current Nix New CLI (the `nix-command` experimental feature) is tightly coupled with the Flakes experimental feature. While there are ongoing efforts to explicitly separate them, using Flakes essentially requires the use of the New CLI. In this book, serving as a beginner's guide to NixOS and Flakes, it is necessary to introduce the differences between the New CLI, which Flakes relies on, and the old CLI.

Here, we list the old Nix CLI and related concepts that are no longer needed when using the New CLI and Flakes (`nix-command` and `flakes`). When researching, you can replace them with the corresponding New CLI commands (except for `nix-collect-garbage`, as there is currently no alternative for this command):

1. `nix-channel`: `nix-channel` manages software package versions through stable/unstable/test channels, similar to other package management tools such as apt/yum/pacman.
    1. In Flakes, The functionality of `nix-channel` is entirely replaced by the `inputs` section in `flake.nix`.
2. `nix-env`: `nix-env` is a core command-line tool for classic Nix used to manage software packages in the user environment.
    1. It installs packages from the data sources added by `nix-channel`, causing the installed package's version to be influenced by the channel. Packages installed with `nix-env` are not automatically recorded in Nix's declarative configuration and are completely independent of its control, making them challenging to reproduce on other machines. Therefore, it is not recommended to use this command directly.
    2. The corresponding command in the New CLI is `nix profile`. Personally, I don't recommend it for beginners.
3. `nix-shell`: `nix-shell` creates a temporary shell environment, which is useful for development and testing.
    1. New CLI: This tool is divided into three sub-commands: `nix develop`, `nix shell`, and `nix run`. We will discuss these three commands in detail in the "Development" chapter.
4. `nix-build`: `nix-build` builds Nix packages and places the build results in `/nix/store`, but it does not record them in Nix's declarative configuration.
    1. New CLI: `nix-build` is replaced by `nix build`.

5. `nix-collect-garbage`：Garbage collection command used to clean up unused Store Objects in `/nix/store`.
    1. There is a similar command in the New CLI, `nix store gc --debug`, but it does not clean the profile generations, so there is currently no alternative for this command.
6. And other less commonly used commands are not listed here.
    1. You can refer to the detailed command comparison list in Try to explain nix commands.

---

1. Flakes - NixOS Wiki ↩
2. Flakes are such an obviously good thing ↩
3. Draft: 1 year roadmap - NixOS Foundation ↩
