Declarative shell environments with shell.nix

Contents

Overview

Declarative shell environments allow you to:

- Automatically run bash commands during environment activation
- Automatically set environment variables
- Put the environment definition under version control and reproduce it on other machines

What will you learn?

In the Ad hoc shell environments tutorial, you learned how to imperatively create shell environments using `nix-shell -p`. This is great when you want to quickly access tools without installing them permanently. You also learned how to execute that command with a specific Nixpkgs revision using a Git commit as an argument, to recreate the same environment used previously.

Skip to main content

In this tutorial we'll take a look at how to create reproducible shell environments with a declarative configuration in a Nix file. This file can be shared with anyone to recreate the same environment on a different machine.

How long will it take?

30 minutes

What do you need?

- Familiarity with the Unix shell
- A rudimentary understanding of the Nix language

Entering a temporary shell

Suppose we want an environment where `cowsay` and `lolcat` are available. The simplest possible way to accomplish this is via the `nix-shell -p` command:

```
$ nix-shell -p cowsay lolcat
```

This command works, but there's a number of drawbacks:

- You have to type out `-p cowsay lolcat` every time you enter the shell.
- It doesn't (ergonomically) allow you any further customization of your shell environment.

A better solution is to create our shell environment from a `shell.nix` file.

A basic `shell.nix` file

Create a file called `shell.nix` with these contents:

```
1 let
2   nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11";
3   pkgs = import nixpkgs { config = {}; overlays = []; };
```

Skip to main content

```
pkgs.mkShellNoCC {
  packages = with pkgs; [
    cowsay
    lolcat
  ];
}
```

Detailed explanation

Enter the environment by running `nix-shell` in the same directory as `shell.nix`:

```
$ nix-shell
[nix-shell]$ cowsay hello | lolcat
```

`nix-shell` by default looks for a file called `shell.nix` in the current directory and builds a shell environment from the Nix expression in this file. Packages defined in the `packages` attribute will be available in `$PATH`.

Environment variables

You may want to automatically export certain environment variables when you enter a shell environment.

Set `GREETING` so it can be used in the shell environment:

```
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11";
  pkgs = import nixpkgs { config = {}; overlays = []; };
in

pkgs.mkShellNoCC {
  packages = with pkgs; [
    cowsay
    lolcat
  ];

+  GREETING = "Hello, Nix!";
 }
```

Any attribute name passed to `mkShellNoCC` that is not reserved otherwise and has a value which can be coerced to a string will end up as an environment variable

Skip to main content

Try it out! Exit the shell by typing `exit` or pressing `Ctrl`+`D`, then start it again with `nix-shell`.

```
[nix-shell]$ echo $GREETING
```

> **Warning**
>
> Some variables are protected from being set as described above.
>
> For example, the shell prompt format for most shells is set by the `PS1` environment variable, but `nix-shell` already sets this by default, and will ignore a `PS1` attribute set in the argument.
>
> If you need to override these protected environment variables, use the `shellHook` attribute as described in the next section.

Startup commands

You may want to run some commands before entering the shell environment. These commands can be placed in the `shellHook` attribute provided to `mkShellNoCC`.

Set `shellHook` to output a colorful greeting:

```
 let
   nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11";
   pkgs = import nixpkgs { config = {}; overlays = []; };
 in

 pkgs.mkShellNoCC {
   packages = with pkgs; [
     cowsay
     lolcat
   ];

   GREETING = "Hello, Nix!";
+
+  shellHook = ''
+    echo $GREETING | cowsay | lolcat
+  '';
 }
```

Skip to main content

Try it again! Exit the shell by typing `exit` or pressing `Ctrl` + `D`, then start it again with `nix-shell` to observe the effect.

References

- `mkShell` documentation
- Nixpkgs shell functions and utilities documentation
- `nix-shell` documentation

Next steps

- Nix language basics
- Automatic environment activation with direnv
- Dependencies in the development shell
- Automatically managing remote sources with npins