Dependencies in the development shell

Contents

When packaging software in `default.nix`, you'll want a development environment in `shell.nix` to enter it conveniently with `nix-shell` or automatically with `direnv`.

How to share the package's dependencies in `default.nix` with the development environment in `shell.nix`?

Summary

Use the `inputsFrom` attribute to `pkgs.mkShellNoCC`:

```
# default.nix
let
  pkgs = import <nixpkgs> {};
  build = pkgs.callPackage ./build.nix {};
in
{
  inherit build;
  shell = pkgs.mkShellNoCC {
    inputsFrom = [ build ];
  };
}
```

Import the `shell` attribute in `shell.nix`:

```
# shell.nix
(import ./.).shell
```

Skip to main content

Complete example

Assume your build is defined in `build.nix`:

```
# build.nix
{ cowsay, runCommand }:
runCommand "cowsay-output" { buildInputs = [ cowsay ]; } ''
  cowsay Hello, Nix! > $out
''
```

In this example, `cowsay` is declared as a build-time dependency using `buildInputs`.

Further assume your project is defined in `default.nix`:

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

Add an attribute to `default.nix` specifying an environment:

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

Move the `build` attribute into the `let` binding to be able to re-use it. Then take the package's dependencies into the environment with `inputsFrom`:

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

Skip to main content

```
    };
  }
```

Finally, import the `shell` attribute in `shell.nix`:

```nix
# shell.nix
(import ./.).shell
```

Check the development environment, it contains the build-time dependency `cowsay`:

```
$ nix-shell --pure
[nix-shell]$ cowsay shell.nix
```

Next steps

- Towards reproducibility: pinning Nixpkgs
- Automatic environment activation with direnv
- Setting up a Python development environment
- Packaging existing software with Nix