Ad hoc shell environments

Contents

In a Nix shell environment, you can immediately use any program packaged with Nix, without installing it permanently.

You can also share the command invoking such a shell with others, and it will work on all Linux distributions, WSL, and macOS[1].

Create a shell environment

Once you install Nix, you can use it to create new *shell environments* with programs that you want to use.

In this section you will run two exotic programs called `cowsay` and `lolcat` that you will probably not have installed on your machine:

```
$ cowsay no can do
The program ‘cowsay’ is currently not installed.

$ echo no chance | lolcat
The program ‘lolcat’ is currently not installed.
```

Use `nix-shell` with the `-p` (`--packages`) option to specify that we need the `cowsay` and

Skip to main content

> **Note**
>
> The first invocation of `nix-shell` for these packages may take a while to download all dependencies.

```
$ nix-shell -p cowsay lolcat
these 3 derivations will be built:
  /nix/store/zx1j8gchgwzfjn7sr4r8yxb7a0afkjdg-builder.pl.drv
  /nix/store/h9sbaa2k8ivnihw2czhl5b58k0f7fsfh-lolcat-100.0.1.drv
  ...

[nix-shell:~]$
```

Within the Nix shell, you can use the programs provided by these packages:

```
[nix-shell:~]$ cowsay Hello, Nix! | lolcat
```

Type `exit` or press `CTRL-D` to exit the shell, and the programs won't be available anymore.

```
[nix-shell:~]$ exit
exit

$ cowsay no more
The program 'cowsay' is currently not installed.

$ echo all gone | lolcat
The program 'lolcat' is currently not installed.
```

Running programs once

You can go even faster, by running any program directly:

```
$ nix-shell -p cowsay --run "cowsay Nix"
```

If the command consists only of the program name, no quotes are needed:

```
$ nix-shell -p hello --run hello
```

Skip to main content

Search for packages

What can you put in a shell environment? If you can think of it, there's probably a Nix package of it.

> **Tip**
>
> Enter the program name you want to run in search.nixos.org to find packages that provide it.

For the following example, find the package names for these programs:

- `git`
- `nvim`
- `npm`

In the search results, each item shows the package name, and the details list the available programs.[2]

Run any combination of programs

Once you have the package name, you can start a shell with that package. The `-p` (`--packages`) argument can take multiple package names.

Start a Nix shell with the packages providing `git`, `nvim`, and `npm`. Again, the first invocation may take a while to download all dependencies.

```
$ nix-shell -p git neovim nodejs
these 9 derivations will be built:
  /nix/store/7gz8jyn99kw4k74bgm4qp6z487l5ap06-packdir-start.drv
  /nix/store/d6fkgxc3b04m85wrhg6j0l5y0ray82l7-packdir-opt.drv
  /nix/store/da6njv7r0zzc2n54n2j54g2a5sbi4a5i-manifest.vim.drv
  /nix/store/zs4jb2ybr4rcyzwq0dagg9rlhlc368h6-builder.pl.drv
  /nix/store/g8sl2xnsshfrz9f39ki94k8p15vp3xd7-vim-pack-dir.drv
  /nix/store/jmxkg8b1psk52awsvfziy9nq6dwmxmjp-luajit-2.1.0-2022-10-04-env.drv
  /nix/store/kn83q8yk6ds74zgyklrjhvv5wkv5wmch-python3-3.10.9-env.drv
  /nix/store/m445wn3vizcgg7syna2cdkkws3kk1gq8-neovim-ruby-env.drv
  /nix/store/r2wa882mw99c311a4my7hcis9lq3kp3v-neovim-0.8.1.drv
these 151 paths will be fetched (186.43 MiB download, 1018.20 MiB unpacked):
  /nix/store/046zxlxhq4srm3ggafkymx794bn1jksc-bzip2-1.0.8
  /nix/store/0p1jxcb7b4p8jhhlf8qnjc4cqwy89460-unibilium-2.1.1
  /nix/store/0q4fpnqmg8liqraj7zidylcyd062f6z0-perl5.36.0-URI-5.05
  ...
```

Skip to main content

```
[nix-shell:~]$
```

Check package versions

Check that you have indeed the specific version of these programs provided by Nix, even if you had any of them already installed on your machine.

```
[nix-shell:~]$ which git
/nix/store/3cdi52xh6lk3h1fb51jkxs3p561p37wg-git-2.38.3/bin/git

[nix-shell:~]$ git --version
git version 2.38.3

[nix-shell:~]$ which nvim
/nix/store/ynskzgkf07lmrrs3cl2kzr9ah487lwab-neovim-0.8.1/bin/nvim

[nix-shell:~]$ nvim --version | head -1
NVIM v0.8.1

[nix-shell:~]$ which npm
/nix/store/q12w83z0i5pi1y0m6am7qmw1r73228sh-nodejs-18.12.1/bin/npm

[nix-shell:~]$ npm --version
8.19.2
```

Nested shell sessions

If you need an additional program temporarily, you can run a nested Nix shell. The programs provided by the specified packages will be added to the current environment.

```
[nix-shell:~]$ nix-shell -p python3
this path will be fetched (11.42 MiB download, 62.64 MiB unpacked):
  /nix/store/pwy30a7siqrkki9r7xd1lksyv9fg4l1r-python3-3.10.11
copying path '/nix/store/pwy30a7siqrkki9r7xd1lksyv9fg4l1r-python3-3.10.11' from 'ht

[nix-shell:~]$ python --version
Python 3.10.11
```

Exit the shell as usual to return to the previous environment.

Skip to main content

Towards reproducibility

These shell environments are very convenient, but the examples so far are not reproducible yet. Running these commands on another machine may fetch different versions of packages, depending on when Nix was installed there.

What do we mean by reproducible? A fully reproducible example would give exactly the same results no matter when or where you run the command. The environment provided would be identical each time.

The following example creates a fully reproducible environment. You can run it anywhere, anytime to obtain the exact same version of the `git`.

```
$ nix-shell -p git --run "git --version" --pure -I nixpkgs=https://github.com/NixOS
...
git version 2.33.1
```

There are three things going on here:

1. `--run` executes the given Bash command within the environment created by Nix, and exits when it's done.

   You can use this with `nix-shell` whenever you want to quickly run a program you don't have installed on your machine.

2. `--pure` discards most environment variables set on your system when running the shell.

   It means that only the `git` provided by Nix is available inside that shell. This is useful for simple one-liners such as in the example. While developing, however, you will usually want to have your editor and other tools around. Therefore we recommend to omit `--pure` for development environments, and to add it only when the extra isolation is needed.

3. `-I` determines what to use as a source of package declarations.

   Here we provided a specific Git revision of `nixpkgs`, leaving no doubt about which version of the packages in that collection will be used.

References

- Nix manual: `nix-shell` (or run `man nix-shell`)

Skip to main content

Next steps

- Reproducible interpreted scripts – use Nix for reproducible scripts
- Nix language basics – learn reading the Nix language, which is used to declare packages and configurations
- Declarative shell environments with shell.nix – create reproducible shell environments with a declarative configuration file
- Towards reproducibility: pinning Nixpkgs – learn different ways of specifying exact versions of package sources

If you're done trying out Nix for now, you may want to free up some disk space occupied by the different versions of programs you downloaded by running the examples:

```
$ nix-collect-garbage
```

---

[1] Not all packages are supported for both Linux and macOS. Especially support for graphical programs may vary.

[2] A package name is not the same as a program name. Many packages provide multiple programs, or no programs at all if they are libraries. Even for packages that provide exactly one program, the package and program name are not necessarily the same.