

# Introduction to the Nix Package Manager

Millian Poquet

2021-05-27 — Workshop of the LIG Axes

# Introduction

## Our use case

- Develop, test, distribute software (OAR, Melissa, Batsim...)
- Define and run controlled experiments

## How Nix can help?

- Define/manage/customize packages and their dependencies
- Reproducible/deterministic packages
- Run code in controlled environments

# Summary

- 1 Nix's main concepts
- 2 What we do with it
- 3 Conclusion and additional resources

# Concept 1 — How to store the packages?

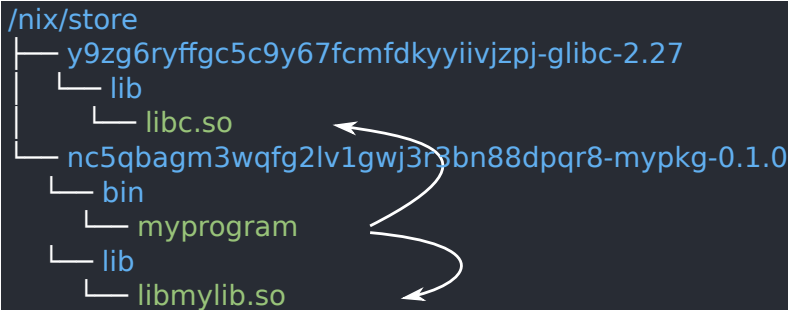
```
/usr
├── bin
│   └── myprogram
├── lib
│   ├── libc.so
│   └── libmylib.so
```

Usual (non Nix) approach: Merge them all

- Default environment all the time (or hacked manually by user)
  - PATH set to /usr/bin (or similar)
  - LD\_LIBRARY\_PATH set to /usr/lib (or similar)
  - LIBRARY\_PATH set to /usr/lib (or similar)
- Huge limitation on package variation
  - Conflicts, as present implies accessible here
  - Files do not have well-defined needs

# Concept 1 — Nix Store

```
/nix/store
├── y9zg6ryffgc5c9y67fcmfdkyyiivzpj-glibc-2.27
│   └── lib
│       └── libc.so
├── nc5qbagm3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0
│   ├── bin
│   │   └── myprogram
│   ├── lib
│   │   └── libmylib.so
```

A diagram showing the Nix Store directory structure. The root is /nix/store. It branches into two main packages. The first package is y9zg6ryffgc5c9y67fcmfdkyyiivzpj-glibc-2.27, which contains a subdirectory 'lib' containing the file 'libc.so'. The second package is nc5qbagm3wqfg2lv1gwj3r3bn88dpqr8-mypkg-0.1.0, which contains subdirectories 'bin' (with file 'myprogram') and 'lib' (with file 'libmylib.so'). Three white arrows originate from the right side of the diagram and point to the files 'libc.so', 'myprogram', and 'libmylib.so'.

Nix approach: Keep them separated

- Convenient package variation
  - Different package source → Different store path
  - Files have precise dependencies  
(DT\_RUNPATH in ELFs, wrappers for PYTHONPATH...)
- Can *install* in default environment (tune PATH...)
- Isolated environments (enter shell with well-defined PATH...)

## Concept 2 — Expressions, derivations, functions

### Descriptive approach

- Define packages/environments in a DSL (also called Nix)
- Derivation = build action that populates the store
- Most of the time, package = function that returns a derivation
  - Inputs: dependencies, build systems, source, build script
  - Outputs: trees of files to write in the store

### Convenient customization

- Can override a package to change its inputs
  - Sources for a specific version of your soft
  - Add or change a dependency
- Can also override *phases* of the build script

# Concept 3 — How to achieve reproducible builds?

## Main ideas

- Package description
  - Default *phases* rarely need customization
  - Only customize what you need, do not write full build scripts
- Pure build environment (depends on arch, not system)
- Compilers told to use generic instruction sets by default
- Build in a sandbox/jail
  - No network access (*inputs* such as sources are fetched before)
  - No filesystem access (unless it is an *input*)
  - No ipc accesses (unless part of your build)
  - No time accesses: back to *epoch* we go

## Key advantages

- Makes it really hard to change behavior depending on weather
- Do not miss dependencies anymore

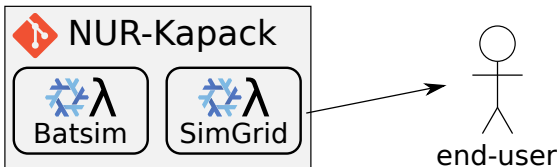
# What we do with Nix in a nutshell



Write Nix expressions.

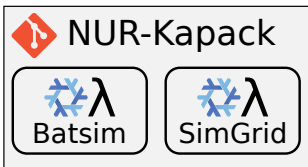


# What we do with Nix in a nutshell



Put them in a Git repository.

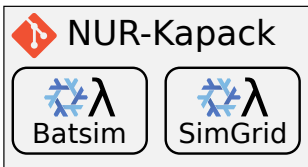
# What we do with Nix in a nutshell



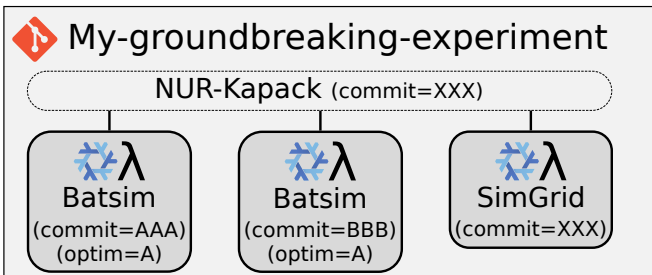
Use them for your experiment.



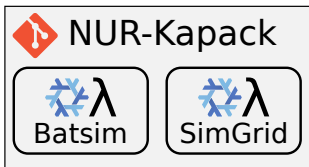
# What we do with Nix in a nutshell



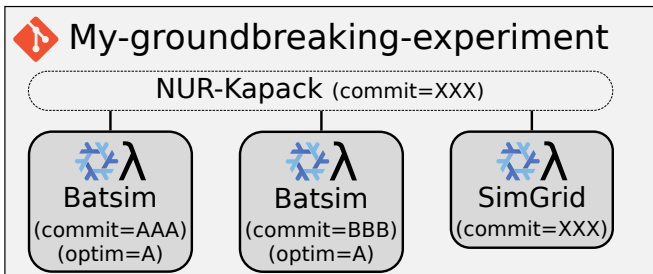
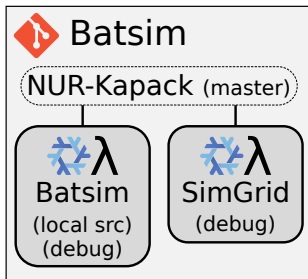
Customize them.



# What we do with Nix in a nutshell



Use them for dev,  
including CI.



# Take home message

## Nix: Reproducible packages/environments

- Descriptive approach for packages/environments
- Multilanguage `virtualenv` → lightweight containers
- Steep learning curve, worth it for complex software or repro

## Additional resources

- 1-hour introduction presentation (french)<sup>1</sup>
- Tutorial on Nix for reproducible experiments<sup>2</sup>
- Nix official website<sup>3</sup> — install, getting started...
- Nix pills<sup>4</sup> — how Nix works

---

<sup>1</sup><https://mpoquet.github.io/research.html#presentations-tutorials>

<sup>2</sup><https://nix-tutorial.gitlabpages.inria.fr/nix-tutorial>

<sup>3</sup><https://nixos.org>

<sup>4</sup><https://nixos.org/guides/nix-pills>