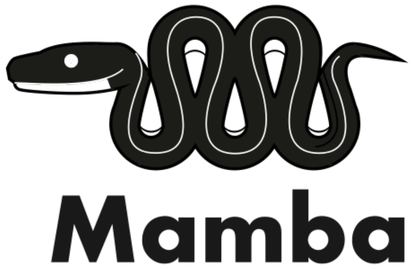
Managing software Environment

with micromamba

F. Flavigny, V. Pestel material

EURO-LABS school. GANIL, Feb 2026



UNIVERSITÉ
CAEN
NORMANDIE

Normandie Université

RÉGION
NORMANDIE

Why an environment manager ?

- We are scientists : **Reproducibility** of our analysis
  - ➡ Required software environment should be reproducible as well

- Handling **Dependencies** (without destroying your mental health)
  - ➡ I need software B and C, bot relying themself on a different versions of software A…
    - It implies to be able to install and swap easily between versions of A

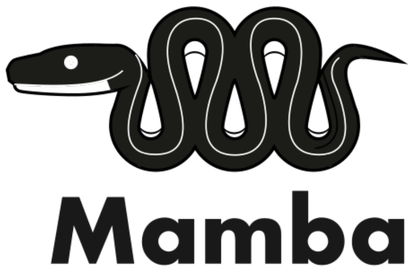
- Easier **Deployment**:
  - ➡ Changes from your own PC to: lab/facility servers, data centers (to get more computing power)
    - I need something to deploy the required softwares

Many solutions

- **Package managers:**
  - ➡ Example: apt on Debian-based systems
  - ➡ Manage package installation on your machine
    - Generally requires root privileges. Only one environment: your machine
- **Python, virtualenv and pip:**
  - ➡ Create a “virtual environment”, where you install python packages with pip
    - No root privileges needed, can handle multiple contexts and environments in //
    - Easy export of environment parameters via text files
    - Handle only python packages
- **Conda environment:**
  - ➡ Create and manage different environments
    - No root privileges needed, can handle multiple contexts and environments in //
    - Easy export of environment parameters via text files
    - Solving dependencies can be terribly slow…..

mamba and micromamba

- **mamba is a C++ reimplementation of conda:**
  - ➡ All the advantages from conda, with ultra fast dependencies resolution
  - ➡ Command line interface identical to conda
  - ➡ Doc: https://mamba.readthedocs.io/en/latest/



- **micromamba is the easy-to-deploy version of mamba**
  - ➡ Install instructions sent by email (next slides)
  - ➡ Doc: https://mamba.readthedocs.io/en/latest/user_guide/micromamba.html

Install micromamba

- Execute the installation script in your preferred shell terminal:
  - ➡ `"${SHELL}" <(curl -L micro.mamba.pm/install.sh)`

- You will be asked a couple of questions:
  - ➡ `Micromamba binary folder ? [~/.local/bin]`
    - Press enter - Define where the micromamba binary executable will be stored. The default is a hidden .local/bin in your home folder.
  - ➡ `Init shell (bash)? [Y/n]`
    - Answer: Y . It will automatically add a couple of lines to your .bashrc to define environment variables. Same as running the command `micromamba shell init`
  - ➡ `Configure conga-forge? [Y/n]`
    - Answer: Y . Configure conga-forge as default package provider.
  - ➡ `Prefix location? [~/micromamba]`
    - Press enter - Define where the environments you will create will be stored. Each environment created will have its package stored in this folder. For the bootcamp, preferably keep it in your home folder. When working on data centers, other choices might be preferred because environments can get pretty large.

Test if micromamba is working

- Source your .bashrc to have micromamba env. variables setup:
  - ➡ `source ~/.bashrc`

- Test if everything is ready by executing:
  - ➡ `micromamba -h`
    - It should return the list of micromamba options and subcommands
      - ✳ if YES: You are ready for the bootcamp !
      - ✳ If NO: There is an install/load problem ! Contact us.

Create the working environment

- Get the file defining the environment specifications for the bootcamp :
  - ➡ `wget https://gitlab.in2p3.fr/eurolabs-os-school/hands-on-2026/micromamba/-/raw/main/root_py12.yaml`

- Create the “root_py12” environment using micromamba and the file you just downloaded:
  - ➡ `micromamba env create -f root_py12.yaml`

- Test that you can activate the “root_py12” environment and test if root is working for example:
  - ➡ `micromamba activate root_py12`
  - ➡ `root`

Create another env (no input files)

- Create the “dummy_env” environment using micromamba and the file you just downloaded:
  - ➡ `micromamba env create –n dummy_env`
  - ➡ `Micromamba activate dummy_env`

- Ok, now install something, python v3.9:
  - ➡ `micromamba install python==3.9`

- Ok, now I need snamekake as well, but version >=9:
  - ➡ `micromamba install “bioconda::snakemake>=9”`

    - ✳ `Does it work ? Why ?`

The channels

- Channels are repositories on which packages are provided
  - ➡ Contain packages names, versions, dependencies
  - ➡ Most common one is **conda-forge**
  - ➡ Other exists like **bioconda** for snakemake
  - ➡ Channel name can be provided at install:
    - for all target with: -c <channel>
    - for a specific package with: <channel>::<package>

Environment file

- The definition of an environment can be provided from a yaml file:

  ➡ `micromamba env create -f <spec_file>.yaml`

```yaml
name: root_py12

channels:
  - conda-forge

dependencies:
  - python>=3.12
  - root>=6.30
  - make
  - ninja
  - cmake
  - git-lfs
  - git
  - uproot
  - pip
  - bioconda::snakemake>=9
  - docopt
  - boost-histogram
  - pip:
    - graphviz
```

Keeping track of changes

- Within my existing environment (like root_py12), I need a new package:

  ➡ `micromamba install bioconda::snakemake-executor-plugin-slurm`

- Now I want to update my environment file:

  ➡ `micromamba env export --from-history > my_new_env.yaml`