

# Reproducible Scalable Workflows with Nix, Papermill and Renku

PyCon India 2020

Rohit Goswami MInstP AMIE AMIChemE  
[rog32@hi.is](mailto:rog32@hi.is)

October 3, 2020



UNIVERSITY OF ICELAND  
FACULTY OF PHYSICAL SCIENCES

# Outline

---

Backstory

Packaging

Nix

Setup

Reproducibility

Provenance and Clusters

Towards the Future



PyCon India

2020, Online

# Backstory

# Standard Approach

- Language Agnostic

## Workflow

- Write functions/objects
  - Refactor in modules
- Test
  - Unit
  - Integration
- Documentation
- Use after importing

Not interactive enough for data-files



# Modern Data Analysis

- Try before you buy
- Doesn't play nice with tests

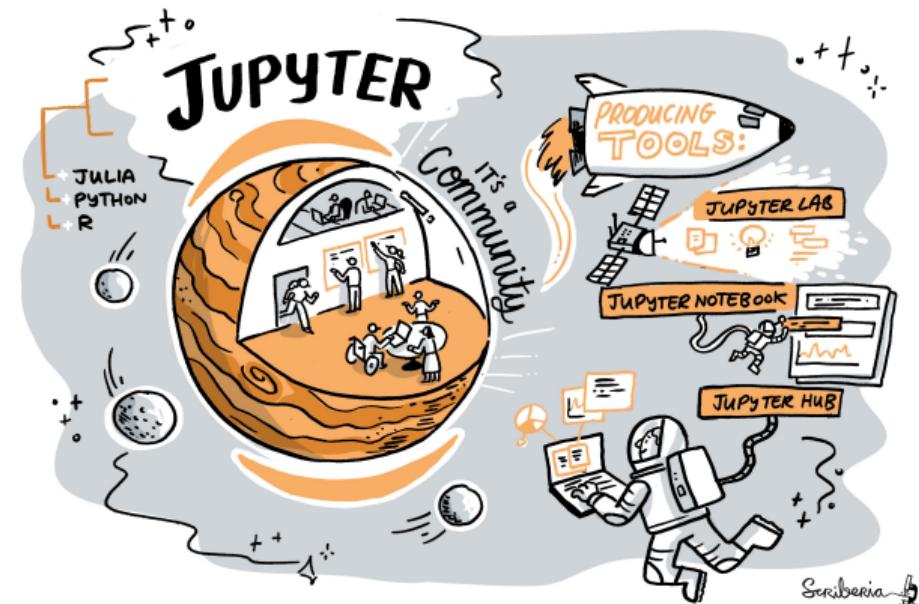
## Python Interactivity

- IPython (ipython)
- Jupyter (Lab/Notebook)
- Colab (Google)



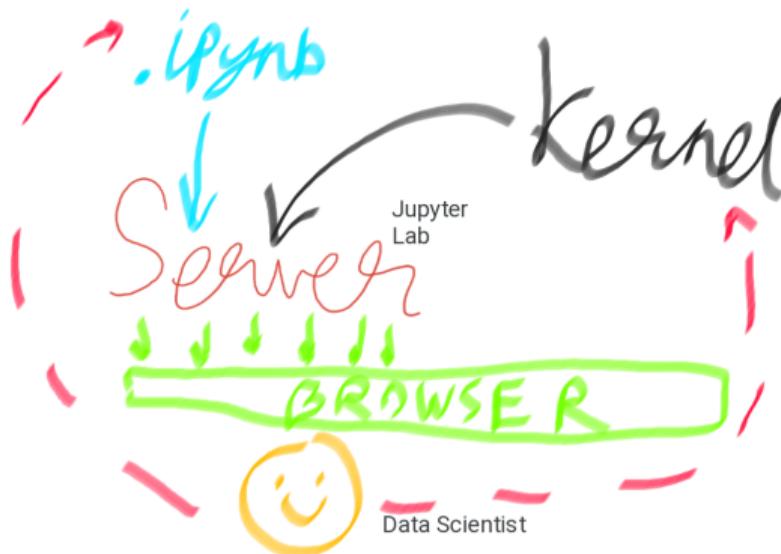
# Jupyter

The Turing Way Community et al.  
*The Turing Way: A Handbook for Reproducible Data Science.*  
Version v0.0.4. Zenodo, Mar. 25, 2019

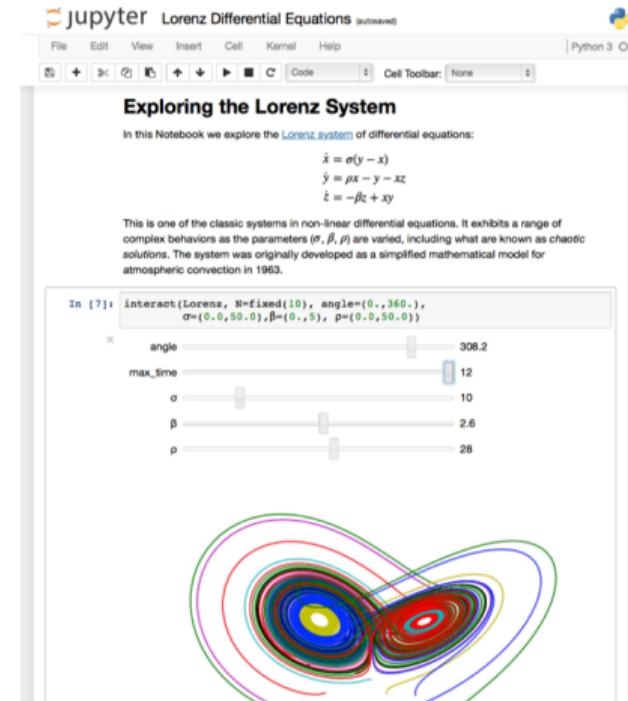


# Jupyter Notebooks

- ipynb files can be exported independently
  - Or via server extensions



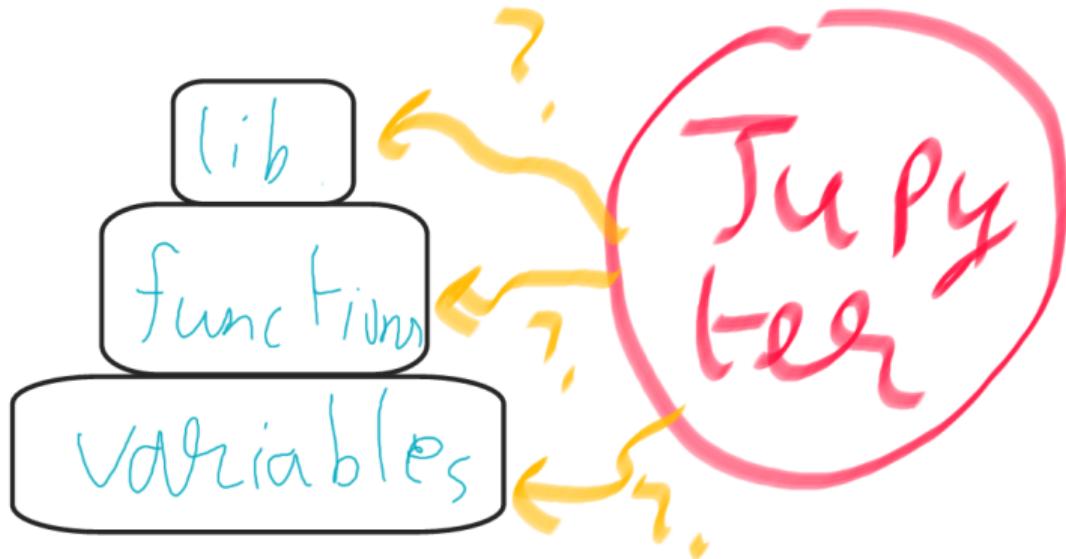
From the Jupyter documentation



# Reconciliation

---

- A crisis of faith
- Made worse by Colab



## Section II

---

# Packaging

# Python Modules

---

- A .py file is a **module**
- It is **standalone** if it only imports from the standard library

# Pure Python Packages

---

- A directory with `__init__.py` in it is a package
- Use pip

# Distributions

---

## Standard

- Built by `setuptools` with `setup.py`
- Simple source only `.tar.gz`
- Distributions have zero or more packages

## Binary

- `wheel`
  - For all your interoperable needs
  - Includes static libraries

# The Python Gradient

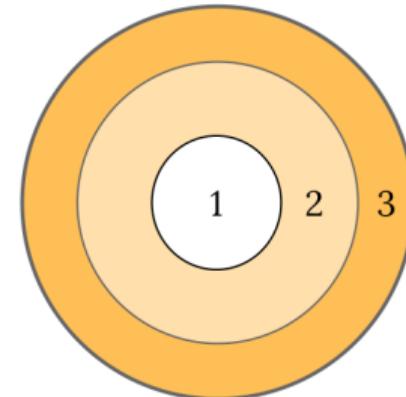
Consider the packaging gradient<sup>1</sup>

## Packaging for Python **libraries**



1. **.py** - standalone modules
2. **sdist** - Pure-Python packages
3. **wheel** - Python packages

(With room to spare for static vs. dynamic linking)



- Libraries and Dev tools are all we get (from PyPI)

<sup>1</sup> by Mahmoud Hashemi (PyBay'17): <https://www.youtube.com/watch?v=iLVNWfPWAC8>

# Pip Requirements

---

- Python
- System libraries
- Build tools
  - Wheels don't work for arbitrary distributions

# Dependency Resolution

---

- requirements.txt (pip)
- Poetry (pretty)
  - pyproject.toml
  - poetry.lock
- Pipenv (older)
  - Pipfile + lockfile
- Pipx (pip but for applications)
- Pyenv and friends



# System Dependencies

---

- Appimages
- Containers
  - docker, flatpak, snapcraft
- Impure filesystems
  - Anaconda



FLATPAK



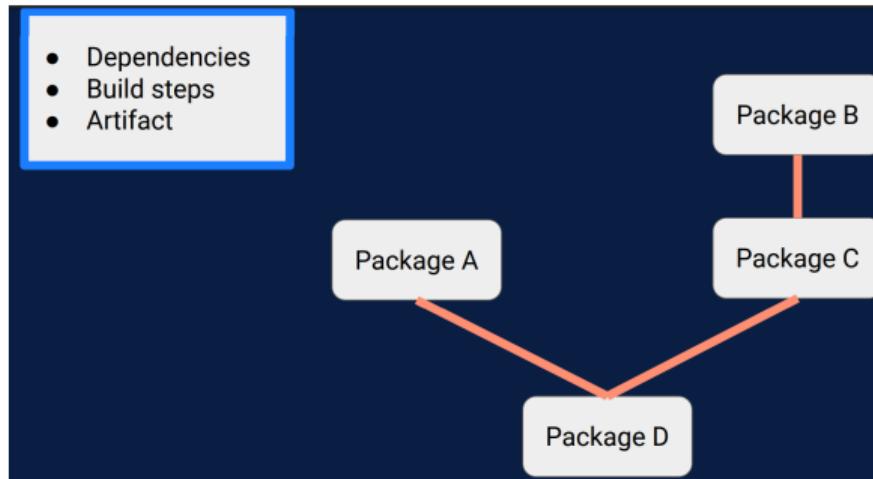
## Section III

---

# Nix

# What?

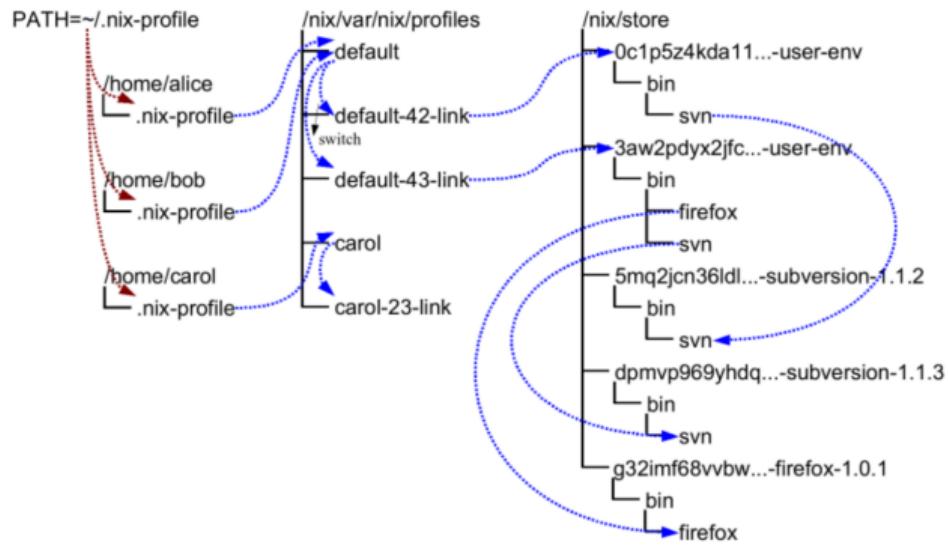
---



- from <https://brianmckenna.org/files/presentations/rootconf19-nix.pdf>

# Nix

Eelco Dolstra, Merijn de Jonge, and Eelco Visser. “Nix: A Safe and Policy-Free System for Software Deployment”. In: (2004), p. 15, Eelco Dolstra, Andres Löh, and Nicolas Pierron. “NixOS: A Purely Functional Linux Distribution”. In: *Journal of Functional Programming* 20.5-6 (Nov. 2010), pp. 577–615



# Why?

---

*Protects against self harm*

*Exposes things taken for granted*

*Enforces consistency*

**Reliable** Purely functional, no broken dependencies

**Reproducible** Each package is in isolation

**How?** store + hash + name + version

# Installation (Multi-User)

---

```
sh <(curl https://nixos.org/nix/install) --daemon
```

- Needs sudo but should not be run as root
- Will make build users with IDs between 30001 and 30032 along with a group ID 30000

# Nix Python - Trial I

---

```
nix-shell -p 'python38.withPackages(ps: with ps; [ numpy toolz ])'
```

- Check which python is loaded
- Check which modules are present

# Nix with Scripts

---

```
#!/usr/bin/env nix-shell
#! nix-shell -i python3 -p "python3.withPackages(ps: [ps.numpy])"

import numpy

print(numpy.__version__)
```

# An Aside into Purity

```
nix-shell --pure --run  
↳ 'bash'
```

- Why?
- What do we solve with this?

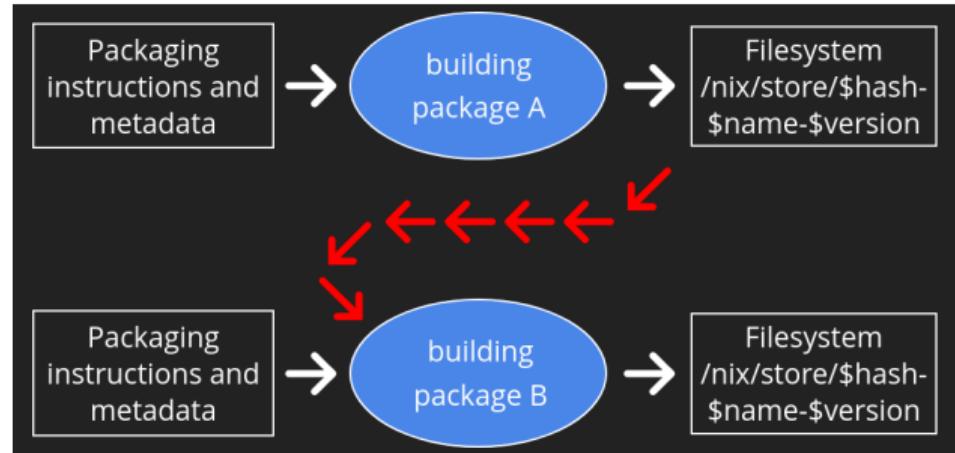


Figure: Stateless builds from <https://slides.com/garbas/mozilla-all-hands-london-2016#/7/0/3>

# Shell in a File

---

```
with import <nixpkgs> {};

let
  pythonEnv = python35.withPackages (ps:
  ↳  [
    ps.numpy
    ps.toolz
  ]);
in mkShell {
  buildInputs = [
    pythonEnv
    which
  ];
}
```

- What **tools** are we adding?
- What **environment** are we using?

# Nix Python Expressions I

---

```
f90wrap = self.buildPythonPackage rec {
  pname = "f90wrap";
  version = "0.2.3";
  src = pkgs.fetchFromGitHub {
    owner = "jameskermode";
    repo = "f90wrap";
    rev = "master";
    sha256 =
      "0d06nal4xzg8vv6sjdbmg2n88a8h8df5ajam72445mhzk08yin23";
  };
  buildInputs = with pkgs; [ gfortran
    stdenv ];
```

- The self portion is from overriding the python environment
- We will dispense with this later

# Nix Python Expressions II

---

```
propagatedBuildInputs = with self; [
    setuptools
    setuptools-git
    wheel
    numpy
];
preConfigure = ''
  export
  F90=${pkgs.gfortran}/bin/gfortran
';
doCheck = false;
doInstallCheck = false;
};
```

- More details here:  
<https://rgoswami.me/posts/ccon-tut-nix/>
- Note that the propagatedBuildInputs are for the python packages

# Friendly Nix

---

```
nix-env -i nox  
nox niv
```

- Niv** For pinning packages
- Nox** Interactive package management
- Lorri** For automatically reloading environments
- Mach-Nix** For working with Python
- Nix-Prefetch-Url** For obtaining SHA hashes

## Section IV

---

# Setup

# Replacing Conda I

---

```
let
  sources = import
    ./prjSource/nix/sources.nix;
  pkgs = import sources.nixpkgs {};
  mach-nix = import (builtins.fetchGit {
    url =
      "https://github.com/DavHau/mach-nix/";
    ref = "2.2.2";
  });
}
```

- Note our definition of `mach-nix`
- Best practices involve `niv` pinned sources

# Replacing Conda II

---

```
customPython = mach-nix.mkPython {
    requirements = builtins.readFile
        ./.requirements.txt;
    providers = {
        _default = "nixpkgs,wheel,sdist";
        pytest = "nixpkgs";
    };
    pkgs = pkgs;
};
in pkgs.mkShell { buildInputs = with
    pkgs; [ customPython ]; }
```

- More details here: <https://rgoswami.me/posts/mach-nix-niv-python/>

# Replacing Conda III

---

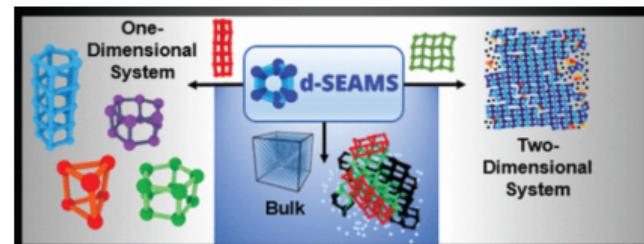
```
overrides_pre = [
    (pythonSelf: pythonSuper: {
        pytest =
            ↳ pythonSuper.pytest.overrideAttrs
            ↳ (oldAttrs: {
                doCheck = false;
            });
        f90wrap =
            ↳ pythonSelf.buildPythonPackage
            ↳ rec {...};
    })
];
];
```

- An important aspect of mkPython
- More details here: <https://rgoswami.me/posts/mach-nix-niv-python/>

# More Nix

---

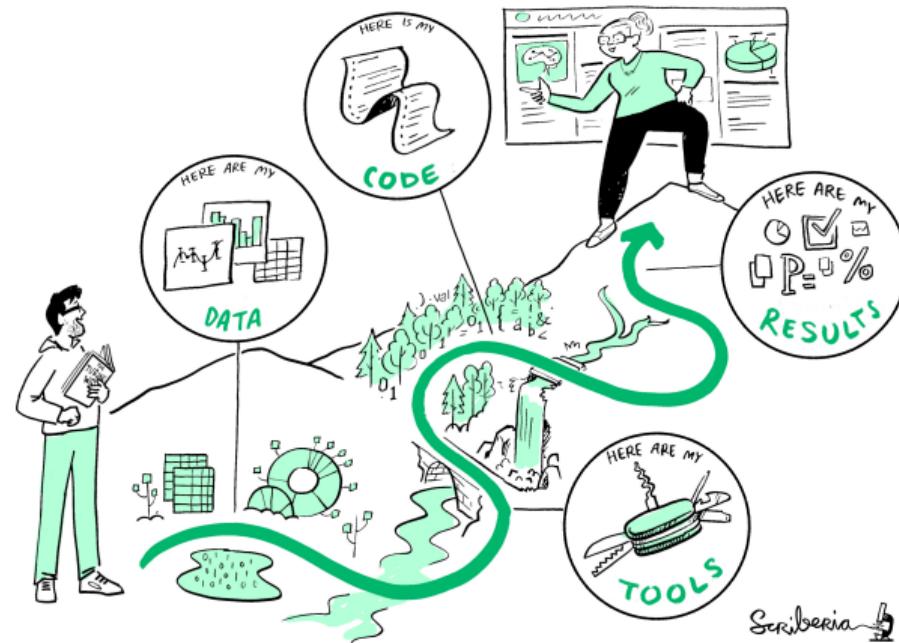
- Try Nix Pills
- Roll your own environment
- Make a docker image
- Try a more complex system (d-SEAMS [4])



# Reproducibility

# What Reproducibility?

The Turing Way Community et al.  
*The Turing Way: A Handbook for Reproducible Data Science.*  
Version vo.0.4. Zenodo, Mar. 25, 2019



# Data Science Woes

---

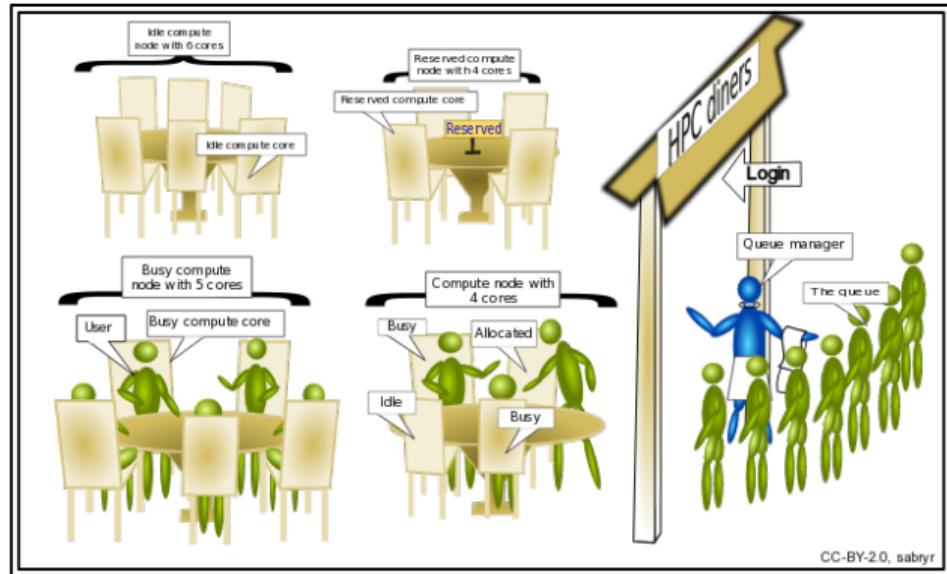
- Version Control
  - Git, SVN, Mercurial (hg)
- Collaboration
  - Overleaf, Google Drive, OneDrive
- Reproduce environments
  - Docker, Conda, Nix
- Re-run analysis
  - Luigi, any CWL runner

		Data	
		Same	Different
Analysis	Same	Reproducible	Replicable
	Different	Robust	Generalisable

# Provenance and Clusters

# Cluster Woes

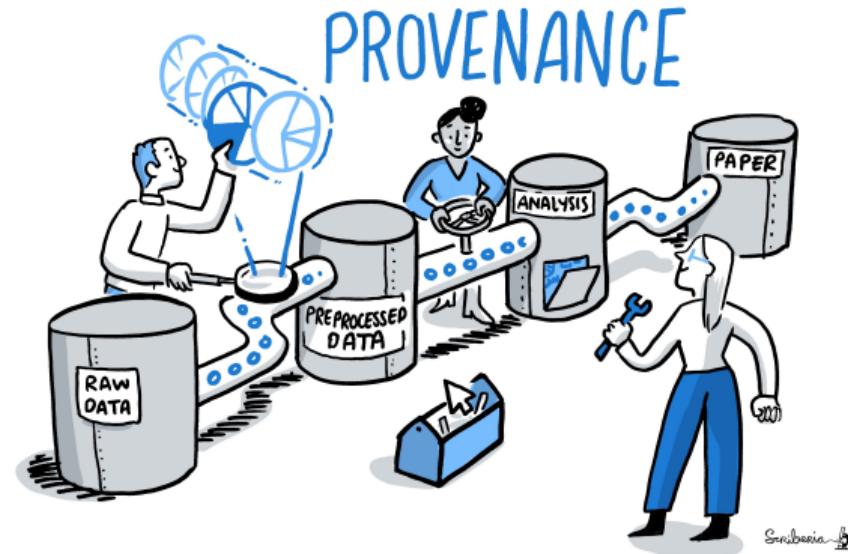
- No docker
  - If lucky, will have singularity
- No userspace support
- Probably runs CentOS or something
- Has a networked file system
- Uses a resource queue
  - Slurm, PBS
- Might have support for lmod



# Provenance

---

The Turing Way Community et al.  
*The Turing Way: A Handbook for Reproducible Data Science.*  
Version v0.0.4. Zenodo, Mar. 25, 2019



# Setup Jupyter

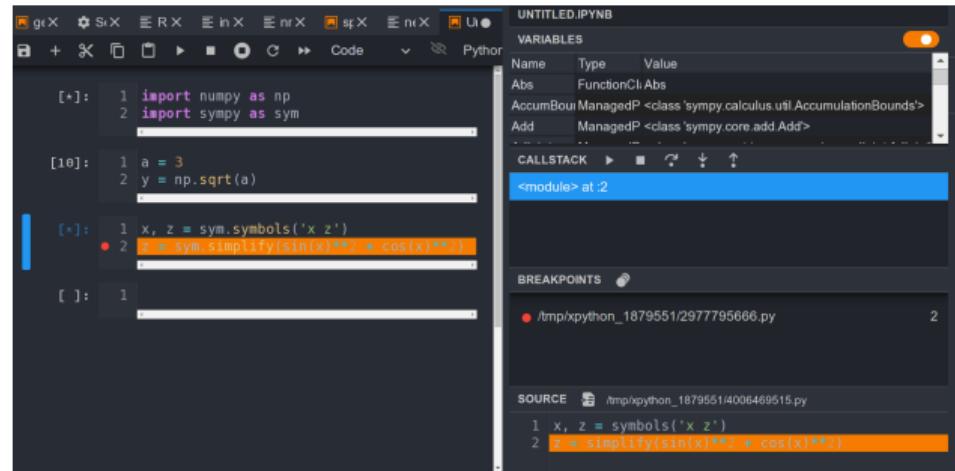
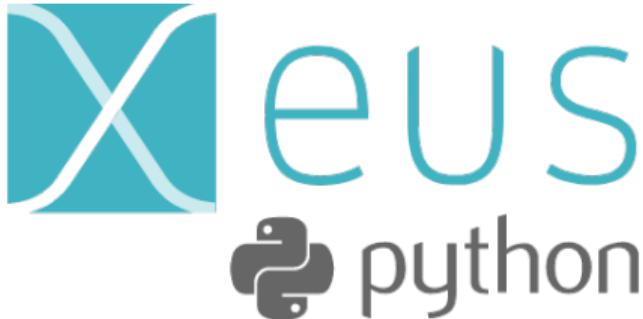
---

- Prefer conda
  - Export a yml with setup
- Use nvm
- Track provenance manually
  - For plugins and setup
- Consider direnv

```
1 jupyter lab --generate-config  
2 vim ~/.jupyter/jupyter_notebook_config.py  
3 # Change c.NotebookApp.notebook_dir to a  
4   ↵ full path
```

# Xeus Python

- Best Jupyter debugger
- Does not support all magics



A screenshot of the Xeus Python debugger interface. The interface is divided into several sections:

- Code Editor:** Shows a code cell with the following Python code:

```
[*]: 1 import numpy as np
      2 import sympy as sym
      3
[10]: 1 a = 3
      2 y = np.sqrt(a)
      3
[*]: 1 x, z = sym.symbols('x z')
      2 z = sym.simplify(sin(x)**2 + cos(x)**2)
      3
[ ]: 1
```
- VARIABLES:** A table showing variables and their types:

Name	Type	Value
Abs	FunctionClass	Abs
AccumBounds	ManagedP	<class 'sympy.calculus.util.AccumulationBounds'>
Add	ManagedP	<class 'sympy.core.add.Add'>
- CALLSTACK:** Shows the current call stack entry: <module> at :2
- BREAKPOINTS:** Shows a single breakpoint at /tmp/xpython\_1879551/2977795666.py:2
- SOURCE:** Shows the source code file with the breakpoint highlighted in orange:

```
1 x, z = symbols('x z')
2 z = simplify(sin(x)**2 + cos(x)**2)
```

# Reusing Notebooks

---



- jupy  
+ text

## Papermill

- Notebooks are functions
- Can be parameterized on the fly
  - No need to refactor
  - Cells become the analysis
- Mostly supports integration tests
- Actually these work best together, especially as papermill can be called in python directly

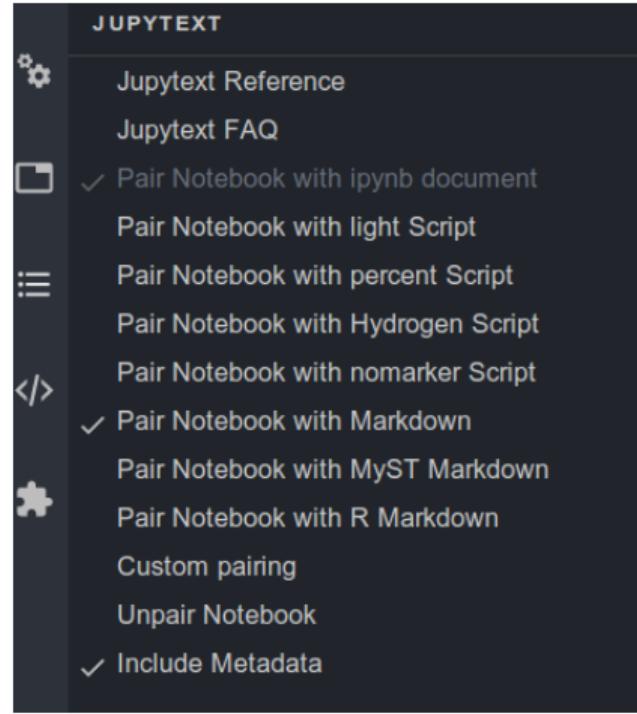
## Jupyter

- Notebooks are literate snippets
- Must refactor cells into functions
- Supports testing more transparently
  - Unit tests

# Jupytext I

---

- Works best with version control
  - Never commit an `.ipynb!`
- Encourages functions
  - Easier to unit-test later
- Is literate
  - Closer to `ipython` histories
  - Fits well with `orgmode` (via `pandoc`)



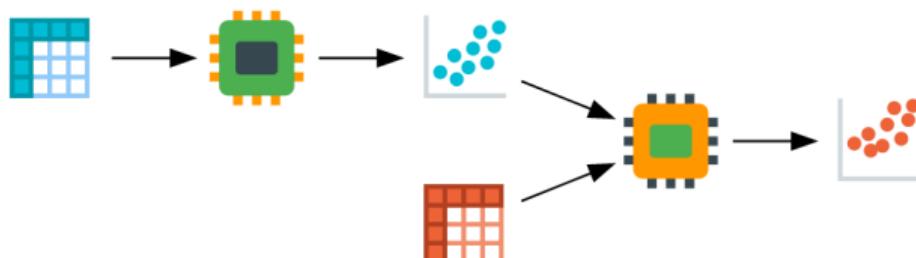
# Jupyter II

	File: getMBP.ipynb
1	{
2	"cells": [
3	{
4	"cell_type": "code",
5	"execution_count": 19,
6	"metadata": {},
7	"outputs": [],
8	"source": [
9	"from ase.io import read as asr\n",
10	"from ase.io import write as asw\n",
11	"from pprint import pprint as ppp\n",
12	"import itertools, collections\n",
13	"import os, shutil, tempfile\n",
14	"import sys, subprocess\n",
15	"from pathlib import Path"
16	]
17	},
18	{
19	"cell_type": "markdown",
20	"metadata": {},
21	"source": [
22	"# MBX Helper\n",
23	"This document describes the usage of the `mbx` software along with auxillary tasks. The following tasks are carried out:\n",
24	"- [ ] Splitting `xyz` trajectories\n",
25	"- [ ] Converting `xyz` files to `nrg` files\n",
26	"- [ ] Obtaining energies with the `single_point` binary\n",
27	]
28	},
29	}
	(stdin) 35

	File: getMBP.md
1	---
2	jupyter:
3	jupytext:
4	formats: ipynb,md
5	text_representation:
6	extension: .md
7	format_name: markdown
8	format_version: '1.2'
9	jupytext_version: 1.6.0
10	kernelspec:
11	display_name: Python 3
12	language: python
13	name: python3
14	---
15	```python
16	from ase.io import read as asr
17	from ase.io import write as asw
18	from pprint import pprint as ppp
19	import itertools, collections
20	import os, shutil, tempfile
21	import sys, subprocess
22	from pathlib import Path
23	```\n
24	# MBX Helper
25	This document describes the usage of the `mbx` software along with auxillary tasks. The following tasks are carried out:
26	- [ ] Splitting `xyz` trajectories
27	- [ ] Converting `xyz` files to `nrg` files
28	- [ ] Obtaining energies with the `single_point` binary
	(stdin) 35

# Renku

- Has a Web-UI
- Uses standard Git LFS under the hood
- Generates CWL files for each command
  - These become a provenance or lineage history
  - Image from renku docs



*Renku (連句 “linked verses”), is a Japanese form of popular collaborative linked verse poetry, written by more than one author working together.\**

—Wikipedia

```
renku run python  
↳ run_analysis.py -i  
↳ inputs -o outputs
```

## Section VII

---

# Towards the Future

# Parting Practicalities

---

- Keep Jupyter **impure**
- Do **not** rely on Colab
- Always keep a plain-text version
- Replace **functions** with parameterized Jupyter notebooks
- Reduce magics in parameterized notebooks
  - Maximize Xeus where possible
- Ask your sys-admins for **nix** or try a user-install
  - Unsupported but try: <https://rgoswami.me/posts/local-nix-no-root/>
- **conda** should be used for **global** installations
  - Like Jupyter
- Use **nix** derivations for actual environments
- Use **renku** to track provenance per project
  - Also tracks databases

# Conclusions

- Interactivity is here to stay
  - Especially in data science
- Jupyter notebooks are here to stay
  - Scalable development is still possible
- Nix ensures reproducible system dependencies



Sorabia

- Meeting old-school TDD developers halfway is best

## Tools

**Xeus Python** PDB on steroids for Notebooks

**Jupytext** Version Control and literate programming

**Papermill** Notebooks-are-functions

**Renku** Write CWL without tears

# References I

---

-  The Turing Way Community et al. *The Turing Way: A Handbook for Reproducible Data Science*. Version v0.0.4. Zenodo, Mar. 25, 2019.
-  Eelco Dolstra, Merijn de Jonge, and Eelco Visser. “Nix: A Safe and Policy-Free System for Software Deployment”. In: (2004), p. 15.
-  Eelco Dolstra, Andres Löh, and Nicolas Pierron. “NixOS: A Purely Functional Linux Distribution”. In: *Journal of Functional Programming* 20.5-6 (Nov. 2010), pp. 577–615.
-  Rohit Goswami, Amrita Goswami, and Jayant K. Singh. “D-SEAMS: Deferred Structural Elucidation Analysis for Molecular Simulations”. In: *Journal of Chemical Information and Modeling* 60.4 (Apr. 27, 2020), pp. 2169–2177.

End

---

Thank you