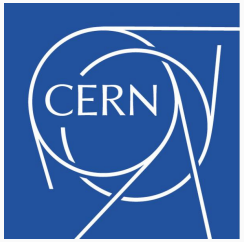


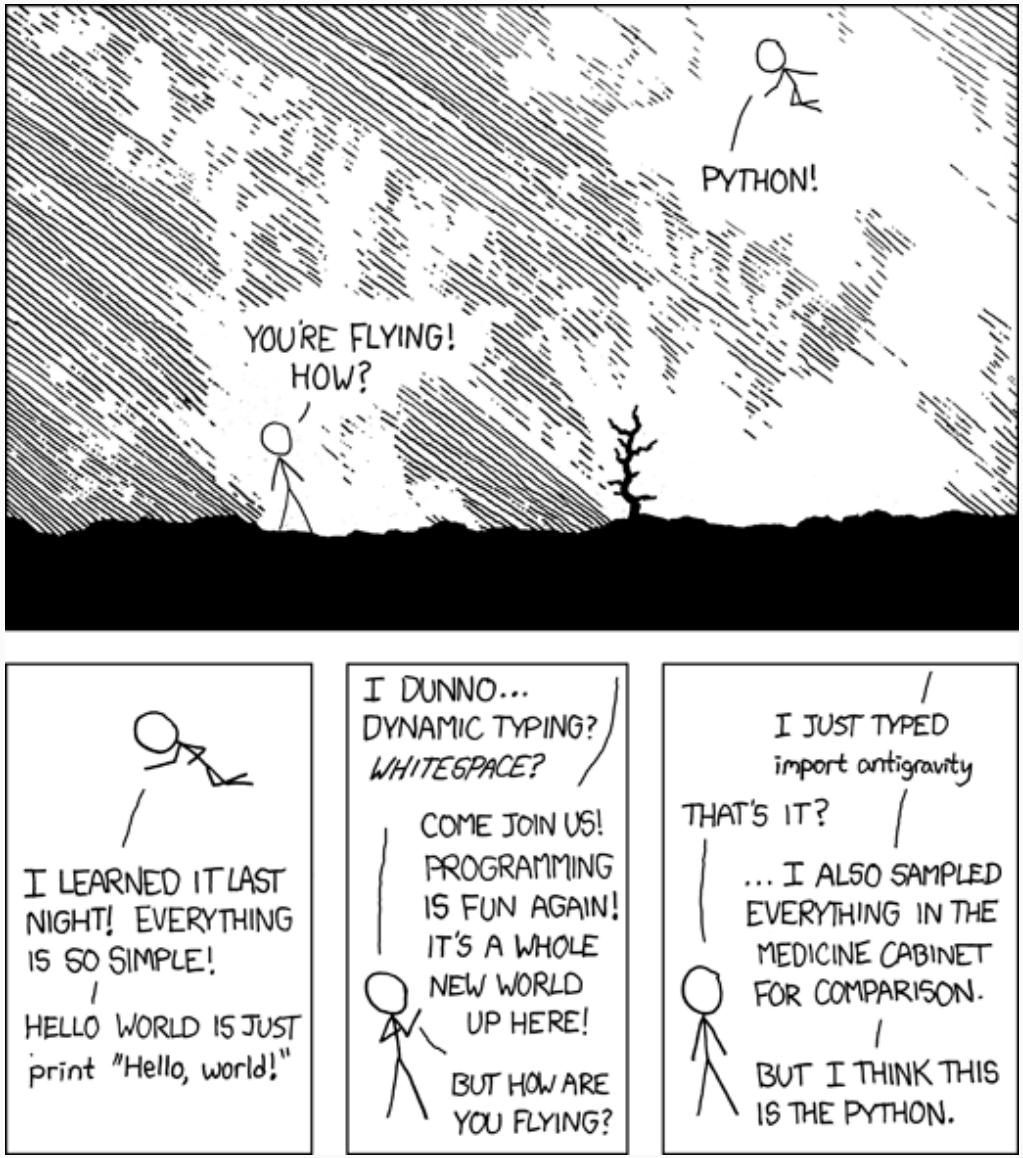
Distributing Python for the HEP environment

PyHEP 2018 Workshop

B. Couturier *on behalf of the LHCb collaboration*

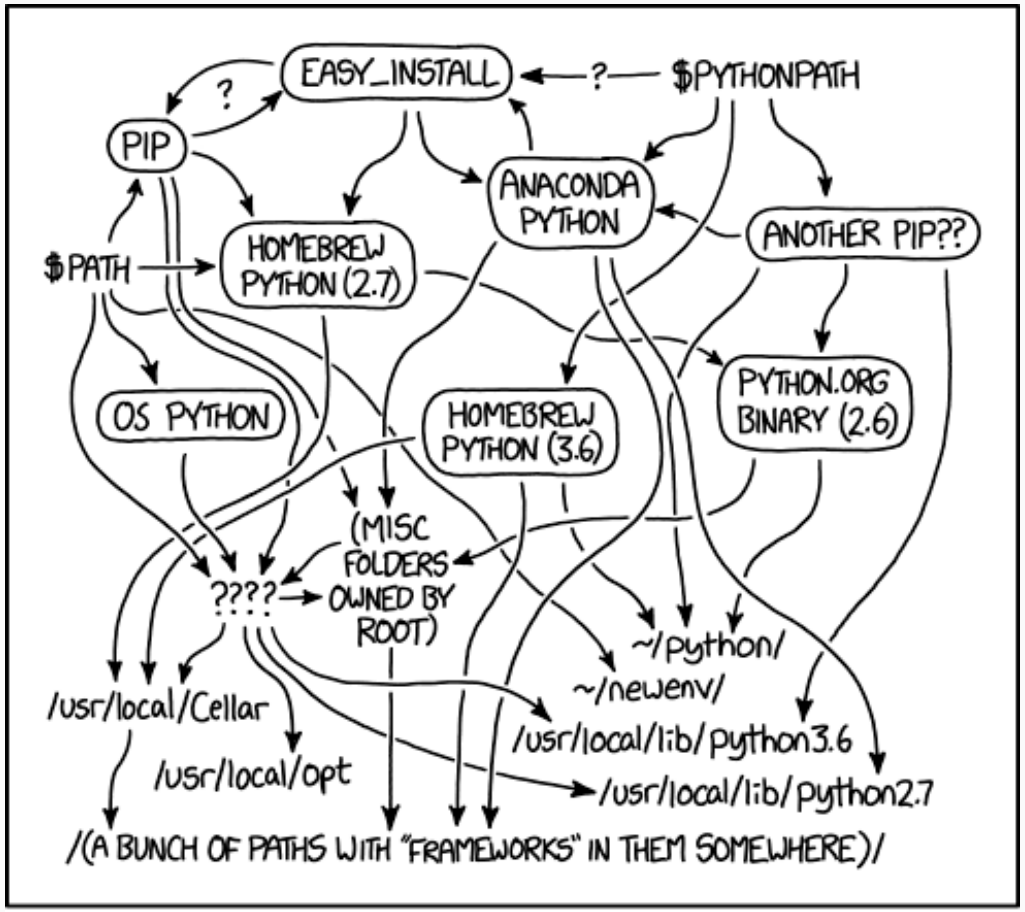
July 7, 2018

Python



https://xkcd.com/353/

Python environment



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987/

Python Packaging

- Not a completely solved problem !
- Definitely a **limiting factor**....

Disclaimer

I will present the (subjective) view of the *LHCb Software Librarian*

- I do not really know good practices
- But I definitely have tried a few bad ones...

This is not an exhaustive review of HEP Python release practices

The LHCb way to distribute python

The LHCb experiment relies heavily on Python
Two ways to get the python code to end users:

- with custom made rpms for the environment scripts and tools
- using the LCG stack as provided by EP-SFT for all the rest
  - including Python (2.7) itself

Very different use cases e.g.:

- environment/productivity tools for developers
- Python tools used in production stacks (or to run them)
- Python tools for analysts using ROOT ntuples

The simple case

Productivity tools for developers: *Python code without binary dependencies*

- **Simplest solution is to produce standard Python packages with setuptools configuration**
- easy enough to host your own PyPI server if needed
- continuous integration tools really help with release

Using standard distribution allows tight integration with development tools

A little bit harder...

Python tools for production: *using the LCG stack*

- Happy to use the versions provided in the LCG stack
- Can plan ahead for upgrades
- *stability and consistency are crucial* (we still use Python 2.7)

The hard case

Python tools for physics data analysis:

- Keen to try latest versions of all tools
- Keen to move to Python 3
- Analysts want to upgrade packages on their own terms:
  - e.g. when bug found or new feature needed
  - normally not security sensitive code
  - need for full reproducibility of the environment

The versions released in the LHCb stacks may not be recent enough...
and installations on top of the stack are not very practical...

Requirements

Common python analysis tools + *ROOT, PyROOT, RooFit, pyxrootd, root_numpy...*

- Need custom and fully reproducible consistent set of versions
- Do not need to compile C++ libraries with ROOT
- Do not need the multiple configurations as proposed by LCG

Python 3 needed in some cases:

- Some packages frozen for python 2, being improved for python 3 packages *(c.f. Python 2 to 3 talk tomorrow by Stefan)*
- New features of the language can be useful

Some tools require Python 2, e.g. LHCb Python tools

Swan and LCG views

Analysts can use LCG views (directly or through Swan)
Large choice of packages, but if some are missing:

- Issues encountered when trying virtual env
- Recommended way is to ”pip install –user package_name”

This works but...

- *Requires CVMFS and therefore maybe also containers to run locally*
- *Can you submit batch jobs with such environment/customizations ?*
- *How can you manage separate installations in case of incompatible packages*
- *How can you ensure reproducibility of the environment (c.f. integration with analysis preservation)*

Preparing your own environment

There are different solutions to use custom versions of Python packages:

- **virtualenvs** using system packages
- **Conda** environment (c.f.Anaconda)
- Containers of custom software stack
  (e.g. using system packages or your favourite build tool such as Nix, Spack, Portage...)

No solution is ideal and hassle free:

- ROOT installation cannot be done with standard python tools
- need to simplify as much as possible
- Integration with the batch systems / preservation tools is critical

Side note: Python distribution

A long (and turbulent) story ! But tools have simplified a lot!

Creation of the Python Packaging Authority working group
https://www.pypa.io/en/latest/

- With the role to create a consistent roadmap for the python packaging tools
- Looking after key tools documentation: **pip**, **setuptools**, **virtualenv**, and **wheel**.
- As well as after the python packaging guide https://packaging.python.org/

Side note: Python and linux binaries distribution

Python wheels portable across linux system has been an issue!

- Not an easy problem to solve...
- But huge efforts went into it !
- c.f. PEP-513, PEP-571
- Still not that easy to install/maintain installation of framework linking with GPU libraries
- Even large packages like Qt can be provided as wheels (within pyside2 in the example referenced)

Side note: Conda and its ecosystem

Conda is a package, dependency and environment management, popular for Python:

- The Anaconda distribution comes with many scientific packages
- conda-forge, a community driven collection of recipes, build infrastructure and distributions
- Very successful custom ”channels” for domain specific packages:
  - Astropy
  - Bioconda

Personal conclusions

Providing python environments for analysts is a hard task

- Some are happy with the python modules deployed centrally on CVMFS
- others want a more dynamic and python centric environment

HEP has its own way to distribute python tools

- Because of the difficulty to distribute binaries of course...
- We tend to distribute fully compiled/consistent environments

This limits adoption of HEP code by other fields...

Personal conclusions

Maybe we could also provide our software with ”standard” python tools
(be it as Wheels or Conda packages...)

- Would suit many installations, even with limitations
  (e.g. is there a need to use ACLiC to produce libraries from conda?)
- Could contribute to the community
- This may also help preservation of the code by improving reproducibility of the setup
- Effort already exist: NLeSC Conda)

Keen to hear other opinions on the topic...

Many thanks to C.Burr, M.Clemencic, V.Gligorov, D.Piparo, G.Stewart, E.Tejedor for the fruitful discussions