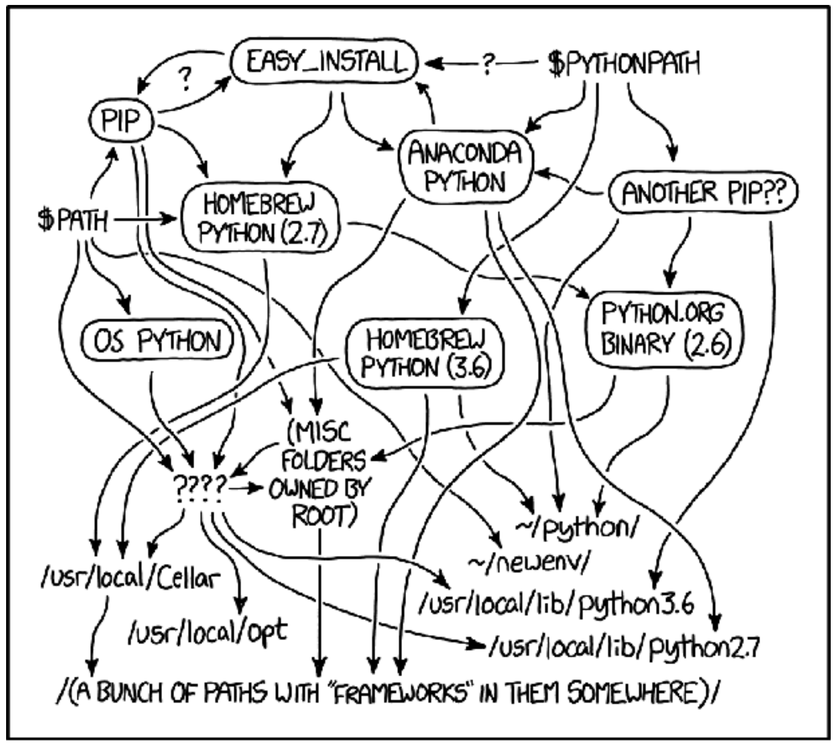
AST1501 - Introduction to Research

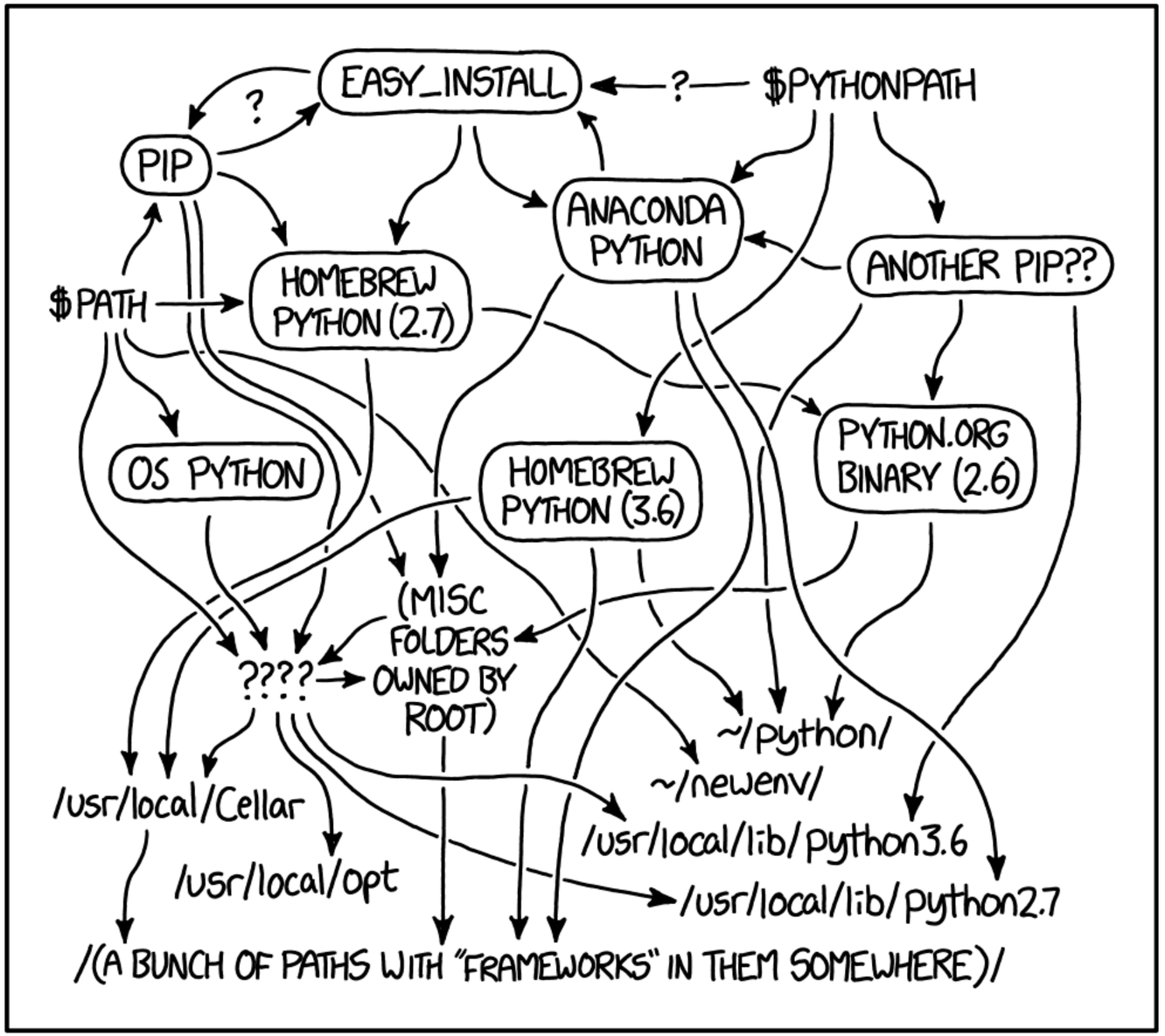
Jo Bovy

Intro to computing III



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

Python environments



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

xkcd

Python environments and package managers

Similar, but not the same

- Python is popular because it is simple yet powerful, but its weakest part is probably its lack of a unified system for managing packages and their versions
    - This is partly by design, because the loose system has allowed Python to do a very large variety of things
- Distinguish between two concepts:
    - Environments: An isolated Python+packages environment that you use to run programs in. Can (and will) have many of these on your computer
    - Package manager: a system for installing packages in your environment
- Generally a bad idea to use your main system's Python version as the main environment you use (e.g., often very out-of-date)

Python environments I

- Outside of (data-)science, the recommended way to set up Python environments is
  - Use Pyenv to manage different Python versions: https://github.com/pyenv/pyenv

    `pyenv install 3.12`
    `pyenv shell 3.12` (or `pyenv local 3.12` or `pyenv global 3.12` or use environment variable)

  - Use `venv` to manage different *virtual environments* (combination of Python+packages): https://docs.python.org/3/library/venv.html

```
python -m venv myenv
source myenv/bin/activate
<then install packages>
```

Python environments II

- In (data-)science, the recommended way is to use the Anaconda (‘conda’) package manager, which is a combined environment and package manager
- The main reason for this is that many powerful Python packages used in (data-)science depend on compiled libraries (both Python and more general libraries) and condo pre-compiles these for many operating systems, making them easy to install
- With conda installed, start new environment with

```
conda create -n myenv python=3.12 <any_additional_packages_you_want>
conda activate myenv
#install any additional packages, e.g.
conda install numpy
#or
pip install numpy
```

- Essential reading: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Conda recommendations

- Don’t install the full Anaconda distribution, instead install *miniconda:* https://docs.conda.io/projects/miniconda/en/latest/miniconda-install.html
- Install mamba as an alternative to the ‘conda’ command (much faster)
- Make conda-forge the default and only channel to find packages (using the main anaconda channel can lead to issue)
- Combine all three recommendations by installing *miniforge*: https://github.com/conda-forge/miniforge
  and `alias conda="mamba"` in your .bashrc

More environment recommendations

- Update your Python version once per year, stay at most one version behind (e.g., now use Python 3.11)
- Use project-specific environments in addition to a general-use one
- Make project-specific environments reproducible through use of an `environment.yml` file
    - Ideally, your supervisor should be able to come in, delete your project environment, and you rebuild your entire environment in five minutes :-)
    - Need to carefully track installed packages whenever you add them
- My personal recommendation is to use `pip` to install packages as much as possible, because conda packages are not always automatically updated to the latest package version (and modern `pip` often installs faster than conda, even mamba)
- However, conda and pip do not cleanly work together, so this can lead to issues. Ideally use conda to setup environment and any difficult-to-pip-install packages, then use pip going forward

Structure of `environment.yml`

```
name: myenv
channels:
  - conda-forge
dependencies:
  - python=3.11
  - ipython
  - jupyter
  - numpy
  - scipy
  - matplotlib
  - pip
  - conda-forge::astropy>=2
  - conda-forge::specutils
  - pip:
    - astroquery
```

Installing Python code

- Work in an environment (should never need root)
- `conda install <package>`
- Or `pip install <package>`
- Update using `conda update <package>` or `pip install -U <package>`; both support the `--no-deps` option to not update dependencies; remove with `conda remove <package>` or `pip uninstall <package>`
- If you need to install from source, do things like
  - Pip install `git+GITHUB-HTTPS-URL` (e.g., `pip install git+https://github.com/astropy/astropy.git`), can add `@BRANCH` to install a specific branch
  - Download or clone the source and do `pip install .` or `pip install -e .` for an editable installation (Python files updated automatically as you edit them)
  - Only in last resort do `python setup.py install`! [Shouldn't be used, hard to uninstall]
  - You will likely have to install non-Python dependencies yourself first (use conda! Or a system package manager like homebrew [Mac] or apt-get/yum [Linux])

What code can/should I use?

- Python has a large ecosystem of packages with various levels of trustworthiness
- Feel free to depend on numpy/scipy/matplotlib/pandas/astropy scientific stack (and scikit-learn and similar scikit- packages) —> well-tested, well-maintained, easy-to-install
- I found a useful package that I can use! But should I just dive in?
  Maybe, but perhaps check
    - Date of last commit on GitHub —> not updated for years is a bad sign (v likely to run into problems)
    - Are issues addressed and fixed?
    - Date of last release —> if old even if package active, hard to install latest version
    - Does a package have a test suite? Does it have good documentation?
- Even if a package doesn't pass all your criteria, it might still be a good thing to use, but expect issues that you will have to resolve yourself