Python Workshop Series Session 5: *Managing your Python Environment*

Mea Trahan
Research Computing

Slides: https://github.com/ResearchComputing/Python_Fall_2021

Outline

- PYTHONPATH and custom modules
- Package management with Conda
- Package management with Pip
- Managing multiple Python installations with Conda

Before we Begin

- No Jupyter notebooks today!
- Open a shell (do not activate your environment)

Recall: Where do modules live?

- Python places modules deep within its directory structure.
- Best not to place your custom modules here
- Let's have a quick look. (Bash commands follow)

```
which python
```

```
/custom/software/miniconda3/envs/idp/bin/python
```

```
export PYDIR=/custom/software/miniconda3/envs/idp
```

```
ls $PYDIR/lib/python3.6/site-packages/
```

The “Path” Concept

- Linux and macOS use special environment variables to manage system behavior.
- *Path* variables are one subset
  - Colon-separated list of directories
  - Searched from left to right until SOMETHING is found (error if not found)

Example: PATH variable

- PATH tells the OS where to search for executables
- In your terminal, try: echo $PATH
- You should see something similar to:
  - /usr/bin:/bin
- When we invoke a program name, the OS checks
  1. /usr/bin
  2. /bin
- If program not found within PATH directories, we get an error

Quick Exercise

- Open a FRESH terminal window
- Set PATH to a null value:
  - TYPE: export PATH=
- Try running the ‘ls’ command
- Close the window via ‘exit’

Quick Exercise

- Open a FRESH terminal window
- Print the value of your PATH variable:
  - TYPE: echo $PATH
- Which Python interpreter do you get?
  - TYPE: which python
- Activate your python environment and repeat the *echo* and *which* commands.
- Conda is an *environment manager*

The PYTHONPATH Variable

- PYTHONPATH
  - A Python-specific *path* variable
  - tells Python where to find modules
- Recommendation:
  - Use PYTHONPATH to manage modules that YOU create
  - Use Conda or Pip to manage 3rd party software
- When importing a module, Python will check:
  1. directory from which script was run
  2. directories in PYTHONPATH
  3. Installation-dependent defaults (including site-packages directory)

PYTHONPATH Example

- Activate your Python environment
- DO NOT start a Jupyter notebook
- Change to the session 7 directory

- Two directories:
  - modules1 – contains mod1.py
  - modules1 – contains mod1.py and mod2.py
- Copy/paste one of the *export* commands at the top of test_path.py (Omit the hastag #)

- Run: python test_path.py

- Rerun after copy/pasting the other export command

Introspection via sys.path

- Can access list of module directories within program via the path *list* (sys module)

```
import sys
print(sys.path)
```

- Path list:
  - Can be manipulated like any other list
  - populated as:
    [script directory, PYTHONPATH, installation dependent defaults]
  - path[0] is null string " " when running interactively

- Windows note:
  - I don't usually work in Windows
  - When I do, this is how I manipulate PYTHONPATH

Package Management with Conda

- General environment manager (not just for Python)
- Do not confuse with Anaconda (full-blown Python distro)
- Manages packages *within a Conda* environment
- Packages downloaded from remote channels
  - *Try*: conda config --get channels (we added Intel)
- Manages & tracks non-python dependencies (e.g., LAPack)
- Very useful for managing multiple python installations

- Advice/Opinions (I prefer Conda):
  - *In my experience*: a bit more intuitive than PIP
  - *In my experience*: great for complicated package installs
  - Recommend use when managing your own python installation

Conda Install

- Let's install the **twisted** network-programming package
- First, check for existence:

```
conda search twisted
```

- If the package is found, we can install it:

```
conda install twisted
```

- Conda will resolve dependencies for us.
- We can now see that the package is installed:

```
conda list
```

```
ls $PYDIR/lib/python3.6/site-packages/
```

Conda Uninstall

- First, restart python and verify you can **import twisted**
- Let's remove the package (reinstall later if you want).

```
conda uninstall twisted
```

- Be careful. Conda tries to prevent broken packages.
- If other packages depend on the one being removed, they may be downgraded or removed as well.

Package Management with PIP

- Pip Installs Packages (Recursive Acronym)
- Installs packages within *any* environment
  - can work alongside Conda
- Packages provided by Python Package Index (PyPI)
- Does not manage non-Python dependencies like Conda
- Advice: Use when:
  - working with Python installation you do not administer
  - working with non-conda Python installations
  - installing simple packages without complex dependency trees

Installation with PIP

- Works similarly to conda (can run pip search).
- Let's try installing h5py for next week…
- Activate your environment
- Recommend you specify non-system directory via --user flag (installs to ~/.local) :

```
pip install h5py --user
```

- Now go ahead and uninstall h5py via:

```
pip uninstall h5py
```

Installation with PIP

- Can run into conflicts if we have multiple Python installs all sharing ~/.local
- Specify custom directory via --prefix flag:

```
pip install h5py --prefix=~/my_modules
```

- More robust, but requires setting PYTHONPATH:

```
export PYTHONPATH=~/my_modules/lib/python3.6/site-packages/
```

- Cumbersome. Why not just use Conda?

PIP uninstall

- Go ahead and uninstall h5py again

```
pip uninstall h5py
```

Final Note on PIP

- Always use --user or --prefix
- Avoid running pip as root (sudo pip install)
- Best not to modify your system python's site-packages directory

Multiple Python Installs with Conda

- Activate your python environment (if needed)
- Start to install h5py, BUT DO NOT CONFIRM

```
conda install h5py
```

- Many packages will be downgraded.
- Is that what we want? Not sure?
- Let's set up a separate Python installation for h5py
- Enter 'n' to cancel h5py install

Multiple Python Installs with Conda

- Open a fresh terminal
- Create a *new* python 3 environment:

```
conda create -n h5py intelpython3_core python=3
```

- Can view available environments via:

```
conda-env --list
```

- Delete environment via:

```
conda remove –name env_name --all
```

Multiple Python Installs with Conda

- Now we can install h5py

```
source activate h5py
conda install h5py
```

- Downgrades no longer worry us.
- Original install (idp) remains unchanged