Table of Contents

Python FAQ

Workshop: using Python at IAC

Slides of the *Using Python at IAC* workshop (11.12.2019)

General

⚠ Warning: do not automatically activate an environment in `.bashrc`

- Automatically activating an environment in (e.g. `source activate iacpy3_2019`) in your `.bashrc` can break your login
- If you cannot login and suspect it is because of this issue
    - ♦ open alternative (non-graphical) shell with `CTRL + ALT + F1`
    - ♦ edit your `.bashrc` and remove the line with `source activate environment`

Also check the docs on JupyterHub!

JupyterHub

Should I use Python?

- Absolutely.

Tutorials

- Python Scientific Lecture Notes: http://scipy-lectures.github.io
- Scientific Python Lectures: https://github.com/jrjohansson/scientific-python-lectures
- Book for Beginners "Dive Into Python": http://www.diveintopython.net

Which version should I use? python 2 or python 3?

- You should use python 3 for every new project.
- ⚠ Note: Python 2 has reached its end-of-life in 2019. It no longer receives bugfixes. Most projects (numpy, matplotlib, ...) stoped supporting python 2!

What's the recommended way to use python at IAC?

- We recommend to use `conda` to manage packages and environments. See below.

What about other solutions?

- **Python installed on the machines?**
  It is discouraged to use the python installation on the machines. Package versions can change unexpectedly which may lead to code incompatibilities. This solution is not viable for long-term reproducibility.

- **Virtual environments?**
  Conda is superior to virtual environments because it also handles non-python dependencies (such as the netCDF library) and can therefore offer a more stable environment.

What are Best Practices when using python?

- For each project you should decide on a conda environment and note which one you use. You could for example create a `startup` file that you call every time you work on the project:

```
#!/bin/bash
module load conda/2022
source activate iacpy3_2022
```

- This file has to be made executable (`chmod +x startup`) and needs to be invoked as `source startup`.
- See this gist for a more complete example

Why is it important to work with a fixed environment?

- Python packages undergo a rapid development and may become incompatible with your script. If you want to re-run your analysis in at a later stage you want to have the same versions for the packages you used. Therefore it is important that you know which environment you used.

Using Conda (on Linux)

What is conda?

- conda is a program that manages (python) packages and environments. It allows to use a centralized installation while still providing user flexibility in term of package installation.

What is mamba?

- mamba is a faster drop-in replacement for conda - it allows for much faster dependency solving. You can replace almost every `conda command` by `mamba command`

What is a conda environment?

- From the conda documentation: A conda environment is a directory that contains a specific collection of conda packages that you have installed. For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing. If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them. You can also share your environment with someone by giving them a copy of your environment.yaml file.

What is a conda package?

- A conda package is a compressed tarball file that contains system-level libraries, Python or other modules, executable programs and other components. Conda keeps track of the dependencies between packages and platforms. Thus, it can not only handle python packages but also other dependencies (e.g. the netCDF c library).

- See also conda documentation

How do I use conda?

- Load the module

```
module load conda
```

- View all environments

```
conda env list
```

- Work in an environment

```
source activate iacpy3_2023
```

  - ⚠ Warning: do not do this in your `.bashrc`, instead define an alias (see below)
- List installed python packages

```
conda list
```

- jupyter -> use JupyterHub)
- Start spyder

```
spyder
```

- Use ipython

```
ipython
```

- For a simpler and quicker usage, you can create an alias in your ~/.bashrc:

```
alias iacpy24='module load conda; source activate iacpy3_2024'
```

What environments are available?

- You can also make all environments available with `module load conda`
- **2024** environments

```
module load conda/2024
conda env list
```

  - **`iacpy3_2024`**: based on `iacpy3_2023`, uses python 3.11; updated all packages on 16.04.2024; Note that a number of packages have been removed from the environment - contact iac-linux@env.ethz.ch if you are missing something.

- **2023** environments

```
module load conda/2023
conda env list
```

  - **`iacpy3_2023`**: based on `iacpy3_2022`, uses python 3.11; updated all packages on 08.08.2023; Note that a number of packages have been removed from the environment - contact iac-linux@env.ethz.ch if you are missing something.

Older environments

- **2022** environments

```
module load conda/2022
conda env list
```

  - **`iacpy3_2022`**: based on `iacpy3_2021`, uses python 3.9; updated all packages on 16.03.2022;

- **2021** environments

```
module load conda/2021
conda env list
```

  - **`iacpy3_2021`**: based on `iacpy3_2020`, uses python 3.9; updated all packages on 22.04.2021;

- **2020** environments

```
module load conda/2020
conda env list
```

  - **`iacpy3_2020`**: based on `iacpy3_2019`, uses python 3.7; updated all packages on 15.04.2020;

- **2019** environments

```
module load conda/2019
conda env list
```

- **iacpy3_2019**: based on `iacpy3_2018`, uses python 3.7; updated all packages on 08.04.2019;
- `iacpy_cmip6_ng` environment used to create the cmip6 new generation archive, please use **iacpy3_2019**

- **2018** environments

```
module load conda/2018
conda env list
```

- **iacpy3_2018**: based on `dypy`, uses python 3.6; updated all packages on 03.04.2018;
- **iacpy2_2018**: based on `dypy`, but uses python 2.7; updated all packages on 03.04.2018

- **2017** environments

```
module load conda/2017
conda env list
```

- **dypy**: python3.5 environment with dypy (for LAGRANTO) and suitable for most users
- **cis_env**: python3.5 environment with cis tools in version 1.5.4
- **pyferret_env**: python3.5 environment with pyferret in version 7.0
- **pyn_env**: python2.7 environment with PyNgl and PyNio in version 1.5

- `module load miniconda3` is equivalent to **`module load conda`**

I am missing a package - what can I do?

- Write to iac-linux@env.ethz.ch - we can generally add single packages to the existing environments. If you are impatient, see below.

How can I install a single package?

- You will need to create your own environment (see below), and then add the package with

```
mamba install <package>
```

How can I create my own environment?

- conda allows to manage environments without being root!

Create a new environment

- You can create your own environment. E.g.:

```
mamba create -n myenv scipy
```

- NB: don't forget to include ipython in your new environment

Clone and tweak an existing environment

- If you want to have more control on your environment but still exiting one as a base, you can do the following:

```
mamba create -n analysis_2019 --clone iacpy3_2019
```

- You can now add packages to the new environment:

```
source activate analysis_2019
mamba install <package>
# - OR -
pip install <package> # installation with mamba is preferred!
```

Background information about the differences between pip, conda and anaconda

- Blogpost by Jake Vanderplas

How can I create an executable python script when using a conda environment?

- To run a python script directly from the command line (**`./script.py`**) you need to add the following at the top of your script

```
#!/usr/bin/env python
```

- However, the environment needs to be loaded, before it is executed.
- Of course the file needs to be executable (`chmod +x scripy.py`)

How to best run jupyter notebook on a server?

- We recommend using JupyterHub.

- See below how to run a notebook on a server from your personal computer.
- Here we to setup a jupyter notebook running on a server and use it from your computer.
- It will make use of tmux on the server to keep a session running even if you are not logged in anymore.
- For security purpose the jupyter notebook produces a token at start, you need to copy this token (password) to the login screen of the notebook.

- On the server

```
tmux new-session -s 'background jobs'
cd ~
module load conda/<year>
source activate <environment>
jupyter notebook --no-browser --port 55000
```

  - ♦ If the port is already in use jupyter will select the next higher available. Make sure that you use the right port in the next part.

- On your computer:

```
ssh -f -N -L localhost:8888:localhost:55000 SERVER
```

- Open the browser and go to: http://127.0.0.1:8888
- Copy the token given by the jupyter notebook on the server and paste it in the login field.
- This also works to tunnel a notebook from a linux machine to a windows machine
    - e.g. via (WSL [windows subsystem for linux]), or Start -> cmd
- You can recover the full address including the token on the SERVER by pressing `CTRL C` **once**.

Run a notebook on a server - from your personal computer

- We recommend using JupyterHub.

- Please don't run jupyter/ python on fog or fog2. These are login nodes that don't have many ressources.

You cannot directly `ssh` to our servers. Therefore, the above solution does not work from you personal computer. There are two possibilities.

1) **Configure fog as a 'jumphost'**

- Edit (or create) `~/.ssh/config` file as follows (on your personal computer)

```
Host fog
  User <username> # your ETH username
  Hostname fog.ethz.ch

Host atmos # change this according to the SERVER you want to use
  User <username> # your ETH username
  Hostname atmos.ethz.ch # change this as well
  ProxyCommand ssh -q -W %h:%p fog
```

- now the following command should work

```
ssh -f -N -L localhost:8888:localhost:55000 atmos
```

- if you try to access the server from Windows using putty, make sure that you add the forwarded port to the profile for your session (on the left under "Category" go to "Connection" -> "SSH" -> "Tunnels" and type "8888" into source port, "localhost:55000" into destination, then click "Add")
- do NOT write `atmos.ethz.ch`, only `atmos`

2) **Use `ssh` from the USYS VPN**

- Connect to the USYS-VPN
- You can now `ssh` to the following servers (i.e. the above script works directly)
    - litho, cfc
- If you want to access another server, please contact iac-linux@env.ethz.ch

Run spyder remotely

- Please don't run spyder on fog or fog2. These are login nodes that don't have many ressources.
- ssh to any of the servers to run your analyis, e.g.:

```
ssh -X <username>@fog.ethz.ch
ssh -X litho
module load conda
source activate iacpy3_2023
spyder
```

- If you don't want to log in twice (-> fog -> server), see tipps above (Run a notebook on a server)

Spyder 3 shows strange symbols

- Something goes wrong with the font in spyder3. The workaround is to use the symbols of spyder2.
- In spyder3 go to `Tools` > `General (Appearance)` > `Icon Theme`: Change to `Spyder 2` and restart spyder.

user packages no longer available in conda environment and JupyerHub

- Packages installed with `pip install <package> --user` (site packages) are no longer available in personal conda environments.
- This should not effect most users.

Why was this change introduced?

- Conda environments should be self-contained and reproducible. Site packages 'pollute' the clean workspace (i.e. there could be packages in an environment that were never installed into it).
- You don't need `--user` to install packages with conda+pip.
    - Most often you will be able to install packages directly with conda (`conda install <package>`)
    - When using conda+pip it is not necessary to install packages with `--user`; you can do this with `pip install <package>`

I'am missing my self-installed packages, what can I do?

- Recommended: You should re-install the packages into your enviroment:

```
module load conda
source activate <environment>
pip install <package>
```

- NB: Make sure that pip is included in the environment before installing extra packages into it.
- If you installed site packages into an existing environment (one managed by IAC-IT), you will have to clone this environment, see instructions above.
- Not recommended: You can get the old behaviour back by adding the following line to your `~/.bashrc` (for bash):

```
module load conda # as before
unset PYTHONNOUSERSITE
```

What has changed?

- `module load conda` now does the following: `export PYTHONNOUSERSITE=1`. This avoids putting the site-packages in the pythonpath, as explained in the documentation.

Packages

xarray

How do I prevent xarray from automatically adding `_FillValue` to coordinates or variables without `NaN`?

- Specify encoding when saving to disk (xarray version 10.2)

```
# Saving back to disk with encoding dictionary telling not to use _FillValues for sp
encoding = {'lat' : {'_FillValue' : None}, 'lon' : {'_FillValue' : None}, 'time' : {
xarray.Dataset.to_netcdf('./outfile.nc',format='NetCDF4',encoding=encoding)
```

Why did xarray change the type of an nc attribute to string, causing other software to crash on the resulting nc file?

- Probably there was a special character (e.g. 'ü') in the attribute. Two suggested workarounds. (xarray version 10.2)

```
# Workaround A) use 'NetCDF4_CLASSIC'
xarray.Dataset.to_netcdf('./outfile.nc',format='NetCDF4_CLASSIC')
# Workaround B) change the attribute name
xarray.Dataset.attrs['institution'] = 'IAC ETH Zuerich' # instead of 'IAC ETH Zürich
xarray.Dataset.to_netcdf('./outfile.nc',format='NetCDF4')
```

matplotlib

I don't see a figure when typing `plt.show()`?

- Per default the plot backend is set to "agg". The "agg" backend is for writing to file and does not show a plot. Type the following to see your backend:

```
import matplotlib as plt
plt.get_backend()
```

There are two ways to achieve this:

- Set the backend in your code:

```
import matplotlib
matplotlib.rcParams['backend'] = "Qt4Agg"
```

- Or define your backend inside your **~/.config/matplotlib/matplotlibrc** file

```
backend      : Qt4Agg
```

 Important If you have a ~/.matplotlib folder, first move it to the new location ~/.config:

```
mv ~/.matplotlib ~/.config/matplotlib
```

More information can be found here: http://matplotlib.sourceforge.net/users/customizing.html

- IT
- Log In
- **IT Web**
- Create New Topic
- Index
- Search
- Changes
- Notifications
- RSS Feed
- Statistics
- Preferences
- **Webs**
- Public
- System