UCB/LBL Training *Anaconda*

Robert Sinkovits
San Diego Supercomputer Center
May 2, 2018

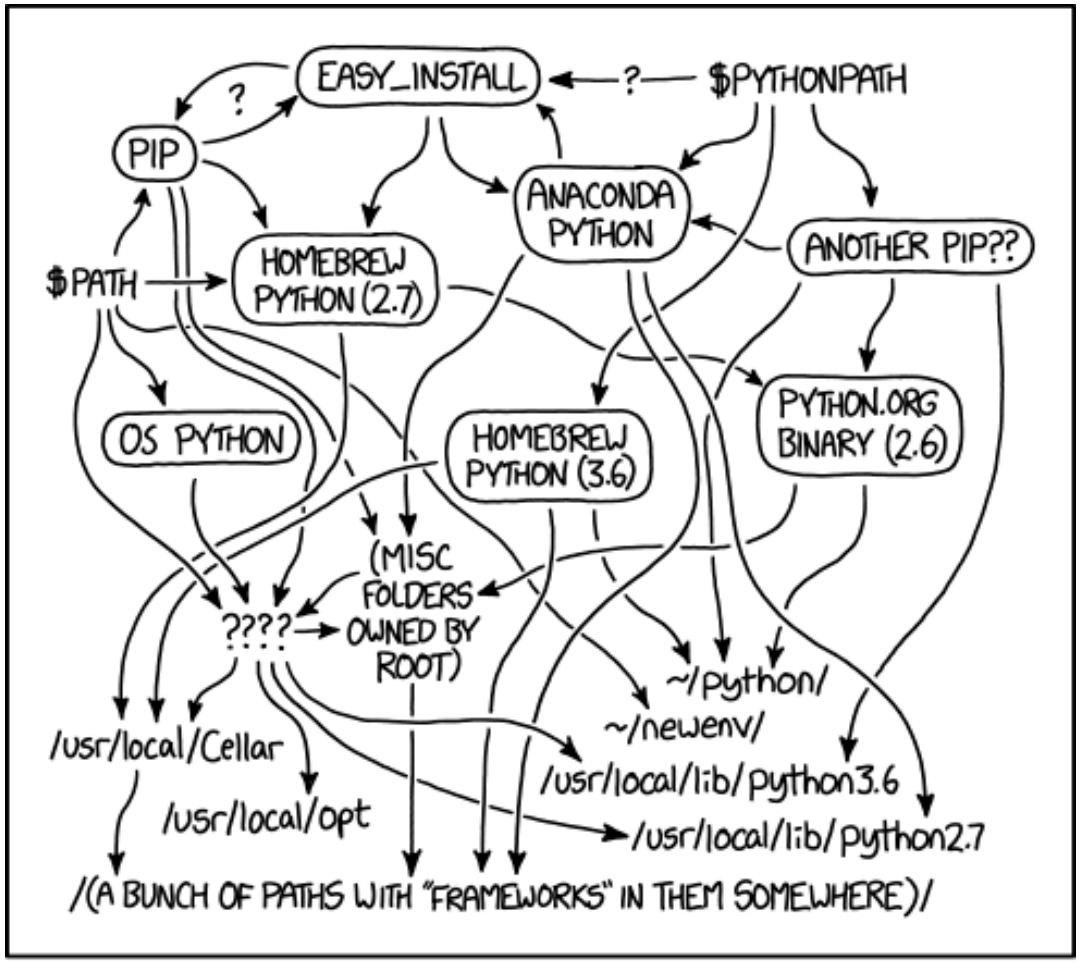
SDSC SAN DIEGO SUPERCOMPUTER CENTER

UNIVERSITY OF CALIFORNIA

Why Anaconda

- Users want us to provide a stable, yet flexible Python environment
- From a support point of view, this can be challenging
  - Users will want different Python versions (some still need Python 2!)
  - The number of packages and modules seems almost infinite
  - Some users will require access to old versions of modules. This seems to be common among grad students nearing the end of their thesis projects who don't want subtle changes in results from bioinformatics pipelines.
  - Some Python packages can have dependencies on systems libraries
- Using Anaconda, users can easily manage their own Python installations, with the added bonus that they can maintain consistency from laptop to cluster

How some view the Python ecosystem



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

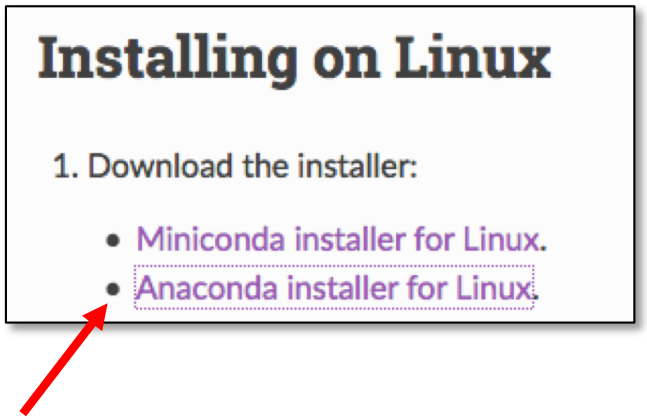
https://xkcd.com/1987/

Anaconda installation

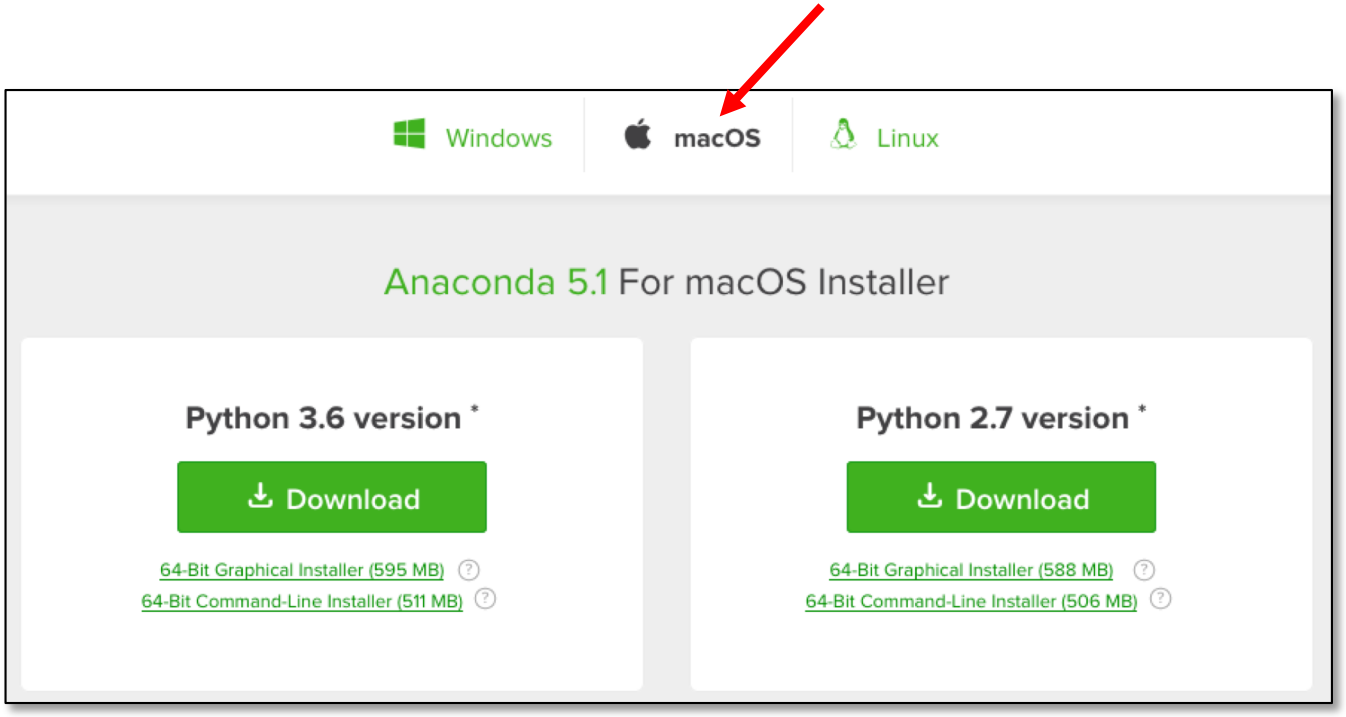
Installing Anaconda on the cluster is easy and doesn't require root access or any special knowledge. Nonetheless, there are a few things that you can do to make it easier for your users.

- Direct users to the installation page
  https://conda.io/docs/user-guide/install/index.html
- Encourage users to download directly to cluster rather than installing on laptop and copying to cluster (right click download button to copy link location)
  wget https://repo.anaconda.com/archive/Anaconda3-5.1.0-Linux-x86_64.sh
- Let users know that the download page will likely direct them to the download for the architecture where their browser is running (see next slide)

Anaconda install gotcha



Using the browser on my Mac and following link to Linux install took me to the macOS download instead of Linux download. Click the penguin to remedy.



Anaconda installation

Download to cluster (Comet) is fast and takes about 10 seconds

```
$ wget https://repo.anaconda.com/archive/Anaconda3-5.1.0-Linux-x86_64.sh

--2018-04-30 10:41:45--  https://repo.anaconda.com/archive/Anaconda3-5.1.0-
Linux-x86_64.sh
Resolving repo.anaconda.com... 104.17.108.77, 104.17.109.77, 104.17.111.77,
...
Connecting to repo.anaconda.com|104.17.108.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 577996269 (551M) [application/x-sh]
Saving to: "Anaconda3-5.1.0-Linux-x86_64.sh"


100%[=======================================================================
===========>] 577,996,269 58.0M/s   in 9.4s
```

Anaconda installation

Full anaconda installation in home directory takes just under six minutes and requires 3.4 GB of disk space

```
$ time bash Anaconda3-5.1.0-Linux-x86_64.sh
Welcome to Anaconda3 5.1.0
                …
Thank you for installing Anaconda3!
real 5m50.775s
user 1m39.212s
sys 0m42.248s

$ du -hs anaconda3/
3.4G anaconda3/
```

Anaconda installation

The Anaconda installer can automatically modify the .bashrc to point to the newly installed Python location. Beware of possible conflicts due to existing PYTHONPATH.

```
WARNING:
    You currently have a PYTHONPATH environment variable set. This may cause
    unexpected behavior when running the Python interpreter in Anaconda3.
    For best results, please verify that your PYTHONPATH only points to
    directories of packages that are compatible with the Python interpreter
    in Anaconda3: /home/etrain109/anaconda3
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /home/etrain109/.bashrc ? [yes|no]
[no] >>> yes


Appending source /home/etrain109/anaconda3/bin/activate to
/home/etrain109/.bashrc
A backup will be made to: /home/etrain109/.bashrc-anaconda3.bak

$ which python3
~/anaconda3/bin/python3
```

Installing packages

Although Anaconda comes with a large number of standard packages, users may still need to install additional packages. This can be done using conda

```
$ python3
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
>>> from Bio import SeqIO
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'Bio'

$ conda install BioPython

$ python3
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
>>> from Bio import SeqIO
>>>
```

Installing packages

But beware that sometimes special installations may be needed
https://stackoverflow.com/questions/33433274/anaconda-graphviz-cant-import-after-installation

```
$ python3
$ conda install graphviz
Solving environment: done
    graphviz:        2.40.1-h25d223c_0
Downloading and Extracting Packages
graphviz

$ python3
>>> import graphviz
Traceback (most recent call last):
ModuleNotFoundError: No module named 'graphviz'

$ pip install graphviz
Installing collected packages: graphviz
Successfully installed graphviz-0.8.3

$ python3
>>> import graphviz
>>> graphviz.__version__
'0.8.3'
```

Miniconda vs. Anaconda

- Miniconda includes only conda (open source package manager and environment management system) and its dependencies. Anaconda includes conda plus about 720 widely used packages.
- Installing Miniconda is quick (~ one minute) and requires little space (348 MB)
- Which one is right for you? It depends. According to https://stackoverflow.com/questions/45421163/anaconda-vs-miniconda

Choose Anaconda if you:

- Are new to conda or Python
- Like the convenience of having Python and over 150 scientific packages automatically installed at once
- Have the time and disk space (a few minutes and 3 GB), and/or
- Don't want to install each of the packages you want to use individually.

Choose Miniconda if you:

- Do not mind installing each of the packages you want to use individually.
- Do not have time or disk space to install over 150 packages at once, and/or
- Just want fast access to Python and the conda commands, and wish to sort out the other programs later.

I use Miniconda myself. Anaconda is bloated. Many of the packages are never used and could still be easily installed if and when needed.

Listing installed packages

Installed packages and their versions can be listed using “conda list”. Of course, you can still get this information from the Python prompt using the appropriate dunder (e.g. numpy.__version__)

```
$ conda list
# packages in environment at /home/etrain109/anaconda3:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0           py36he11e457_0
alabaster                 0.7.10          py36h306e16b_0
anaconda                  5.1.0                   py36_2
anaconda-client           1.6.9                   py36_0
anaconda-navigator        1.7.0                   py36_0
…
yaml                      0.1.7               had09818_2
zeromq                    4.2.2               hbedb6e5_2
zict                      0.1.3           py36h3a3bf81_0
zlib                      1.2.11              ha838bed_2
```

A few words about MKL

To get the best performance for numerically intensive applications, we recommend using the Intel MKL library of threaded and vectorized math routines. Anaconda 2.5 (February 2016) and later come with MKL-powered versions of NumPy, SciPy and Scikit-learn.

https://docs.anaconda.com/mkl-optimizations/

It is possible to opt out of using MKL

https://docs.anaconda.com/mkl-optimizations/#uninstalling-mkl

Not sure why anyone would want to do that when running on a modern cluster or supercomputer other than for benchmarking purposes when testing new math libraries.

A few words about MKL

Can confirm that NumPy, SciPy and scikit-learn were built using MKL with the show_config() method

```
>>> import numpy as np
>>> np.show_config()
mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/etrain109/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/etrain109/anaconda3/include']
blas_mkl_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/etrain109/anaconda3/lib']
    define_macros = [('SCIPY_MKL_H', None), ('HAVE_CBLAS', None)]
    include_dirs = ['/home/etrain109/anaconda3/include']
blas_opt_info:
    libraries = ['mkl_rt', 'pthread']
    library_dirs = ['/home/etrain109/anaconda3/lib']
...
```

A few words about MKL

```
>>> a = np.random.rand(20000,20000)
>>> ainv = np.linalg.inv(a)
```

```
top - 16:48:24 up 43 days,  2:23,  1 user,  load average: 5.11, 3.85, 3.16
Tasks: 1351 total,  26 running, 1325 sleeping,   0 stopped,   0 zombie
Cpu(s): 95.5%us,  2.8%sy,  0.0%ni,  1.5%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  131777004k total, 16082468k used, 115694536k free,    21496k buffers
Swap:        0k total,        0k used,        0k free,   143224k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
23082 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:49.38 python3
23089 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:50.25 python3
23091 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:47.83 python3
23096 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:48.01 python3
23097 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:47.69 python3
23102 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:47.15 python3
23104 etrain10  20   0 17.5g  12g  16m R 100.5 10.0   0:49.19 python3
22809 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   1:01.06 python3
23084 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:49.32 python3
23086 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:50.35 python3
23087 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:50.22 python3
23088 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:50.14 python3
23090 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:48.63 python3
23093 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:47.59 python3
23098 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:46.22 python3
23100 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:47.04 python3
23103 etrain10  20   0 17.5g  12g  16m R 100.1 10.0   0:47.41 python3
23083 etrain10  20   0 17.5g  12g  16m R 99.8 10.0    0:49.35 python3
23092 etrain10  20   0 17.5g  12g  16m R 99.8 10.0    0:46.16 python3
23085 etrain10  20   0 17.5g  12g  16m R 92.6 10.0    0:48.50 python3
23101 etrain10  20   0 17.5g  12g  16m R 80.0 10.0    0:46.93 python3
23094 etrain10  20   0 17.5g  12g  16m R 79.7 10.0    0:45.18 python3
23095 etrain10  20   0 17.5g  12g  16m R 75.1 10.0    0:44.29 python3
23099 etrain10  20   0 17.5g  12g  16m R 74.4 10.0    0:44.57 python3
22318 etrain10  20   0  113m 5216 1464 S  0.0  0.0   0:00.08 bash
```

Working with environments

Environments give you the flexibility to use different versions of Python and/or Python packages without altering your default environment (i.e. the one created when installing Anaconda or Miniconda). Before creating new environments, you may need to update conda to the latest version

```
$ conda update -n base conda
Solving environment: done

## Package Plan ##

  environment location: /home/etrain109/anaconda3

  added / updated specs:
    - conda

The following packages will be downloaded:
    package                    |            build
    ---------------------------|-----------------
    conda-4.5.1                |           py36_0         1.0 MB
```

Creating environments

Environments are created using the conda create command. A few examples are shown below for creating environments based on different Python versions and/or packages

```
# Environment with different Python version (3.4)
$ conda create -n py34 python=3.4

# Environment with older zlib version
$ conda create -n ozl zlib=1.2.8

# Environment with older Python and zlib versions
$ conda create -n py35ozl python=3.5 zlib=1.2.8
```

Creating environments

The new environment builds on the base environment that was created during the Anaconda installation and can be found under anaconda3/env. It encompasses the minimal set of changes that are necessary to deal with dependencies

```
$ ls anaconda3/envs/ozl/lib/
libz.a libz.so  libz.so.1  libz.so.1.2.8  pkgconfig

$ ls anaconda3/lib
cairo   libharfbuzz-gobject.so     libpng.so libQt5XmlPatterns.so
cmake   libharfbuzz-gobject.so.0     libpython3.6m.a
libQt5XmlPatterns.so.5
[-- 903 lines not shown --]
libharfbuzz-gobject.a   libpng.a     libQt5XmlPatterns.la
xsltConf.sh
libharfbuzz-gobject.la   libpng.la     libQt5XmlPatterns.prl
```

Listing and removing environments

Environments are listed and removed with the conda info and remove commands. In the example below, we list the environments before and after removing the ”ozl” environment.

```
$ conda info --envs
base                  *  /home/etrain109/anaconda3
ozl                      /home/etrain109/anaconda3/envs/ozl
py35ozl                  /home/etrain109/anaconda3/envs/py35ozl
py34                     /home/etrain109/anaconda3/envs/py34

$ conda remove -n ozl --all

$ conda info --envs
base                  *  /home/etrain109/anaconda3
py35ozl                  /home/etrain109/anaconda3/envs/py35ozl
py34                     /home/etrain109/anaconda3/envs/py34
```

Activating/deactivating environments

Use source activate and source deactivate to change environments. Note that the prompt changes to reflect the new environment and that repeated calls to activate create a stack of environments.

```
$ python3 -V
Python 3.6.4 :: Anaconda, Inc.
$ source activate py34

(py34) $ python3 -V
Python 3.4.5 :: Continuum Analytics, Inc.
(py34) $ source activate p35_oldzlib

(p35_oldzlib) $ python3 -V
Python 3.5.5 :: Anaconda, Inc.

(p35_oldzlib) $ source deactivate
(py34) $ source deactivate
$
```

Managing R with conda

Although Anaconda and Jupyter are usually associated with the Python community, keep in mind that conda is language agnostic.

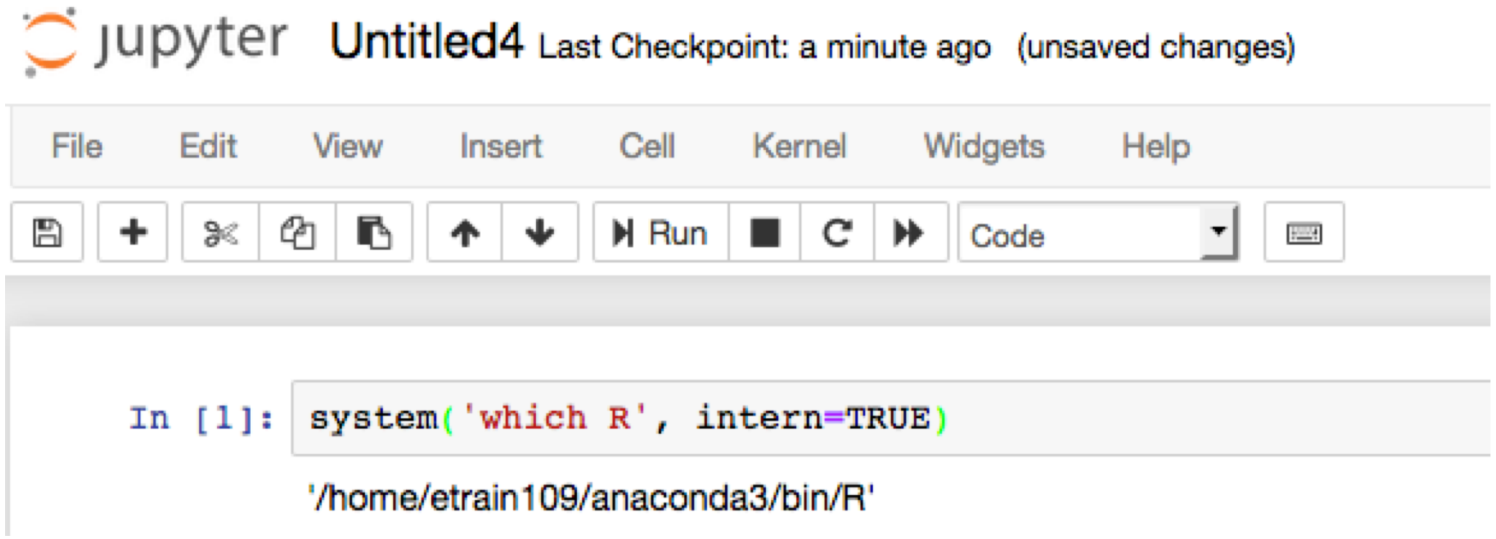
```
$ which R
/opt/R/bin/R

$ conda install -c r r-irkernel
Solving environment: done
## Package Plan ##
  environment location: /home/etrain109/anaconda3

$ which R
/home/etrain109/anaconda3/bin/R
```

Managing R with conda

R notebooks executed in Jupyter will use the R version found in Anaconda and packages will be installed in anaconda3/lib/R/library. Personally, I find this more convenient than dealing with a mix of system wide and local installations.



Exercises (page 1)

- Install Anaconda Python 3.6 version (https://conda.io/docs/user-guide/install/index.html). Use wget to download directly to Comet
- Source .bashrc and confirm paths to python3 and conda (~/anaconda3/bin)
- Edit .bashrc to unset PYTHONPATH (unset PYTHONPATH)
- List info for all Python packages (conda list) and for a few specific packages (conda list pkg)
- Install Miniconda, but don't update your .bashrc (answer 'no' to prepend path question)
- Get an interactive node
  srun --pty --nodes=1 --ntasks-per-node=24 -p compute --reservation=UCB2018Res -t 01:00:00 --wait 0 /bin/bash
- Launch Python3, import numpy as np, and use show_config() to confirm MKL build
- Open a second terminal window, ssh to your interactive node (e.g. ssh comet-12-31), launch top and show threads (Shift-H)
- Execute following at Python prompt and monitor CPU usage in second window
  a = np.random.rand(20000,20000)
  ainv = np.linalg.inv(a)

Exercises (page 2)

- Update your conda version – I had to do this to create new environments
  conda update -n base conda
- Create a new environment using an older version of Python 3
- Create another new environment using an older version of zlib (1.2.8 works)
- List your environments (conda info --envs) and explore the the anaconda3/envs directory
- Activate these new environments (source activate myenv) and confirm the Python version and zlib version either from the Python prompt or using the conda list command
- Deactivate the new environments (source deactivate)
- Remove at least one of the new environments
  conda remove -n myenv --all
- Check out conda documentation
  conda -h
  conda info -h

Exercises (page 3)

If you plan to tinker with R notebooks …

- Get the R version and location before installing R kernel
- Install R kernel
- Get the R version and location after installing R kernel
- Install ggplott2: install.packages("ggplot2")

Intermission

Using Jupyter Notebooks

Notebooks provide an excellent way to integrate code, documentation and output into a single file. As one of my colleagues described it, notebooks are essentially executable documentation. Although the Notebook was originally developed for Python (IPython), the concept has been expanded to other languages (we'll also see some examples in R).

Even if you're comfortable developing code the old fashioned way (edit, save, run, repeat), the Jupyter notebook can make you more productive and allows you to execute blocks of code (cells) rather than the entire application. Graphics, results and readable comments can be interspersed between executable statements.

Using Jupyter Notebooks

Notebooks provide an excellent way to integrate code, documentation and output into a single file. As one of my colleagues described it, notebooks are essentially executable documentation. Although the Notebook was originally developed for Python (IPython), the concept has been expanded to other languages (we'll also see some examples in R).

Even if you're comfortable developing code the old fashioned way (edit, save, run, repeat), the Jupyter notebook can make you more productive and allows you to execute blocks of code (cells) rather than the entire application. Graphics, results and readable comments can be interspersed between executable statements.

OK, Notebooks aren't completely new. They've been a staple of Mathematica for a long time but the Jupyter notebooks take things a step further by allowing the application to other languages and breaking the stand-alone model of Mathematica.

Using Jupyter Notebooks

https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/

Using Jupyter Notebooks

- **Grab an interactive node**

  srun --pty --nodes=1 --ntasks-per-node=24 -p compute --reservation=UCB2018Res -t 01:00:00 --wait 0 /bin/bash

- **Load singularity image**

  module load singularity
  singularity shell /share/apps/gpu/singularity/sdsc_ubuntu_tf1.1_keras_R.img

- **Clone repos**

  git clone https://github.com/sinkovit/PythonSeries
  git clone https://github.com/sinkovit/RSeries

- **Launch notebook**

Using environments with Jupyter

We showed earlier how to create new Python environments that use different versions of Python and/or packages. When working from the command line, it was easy to change environment

$ source activate myenv
(myenv) $

When working with Jupyter, a few more steps are needed

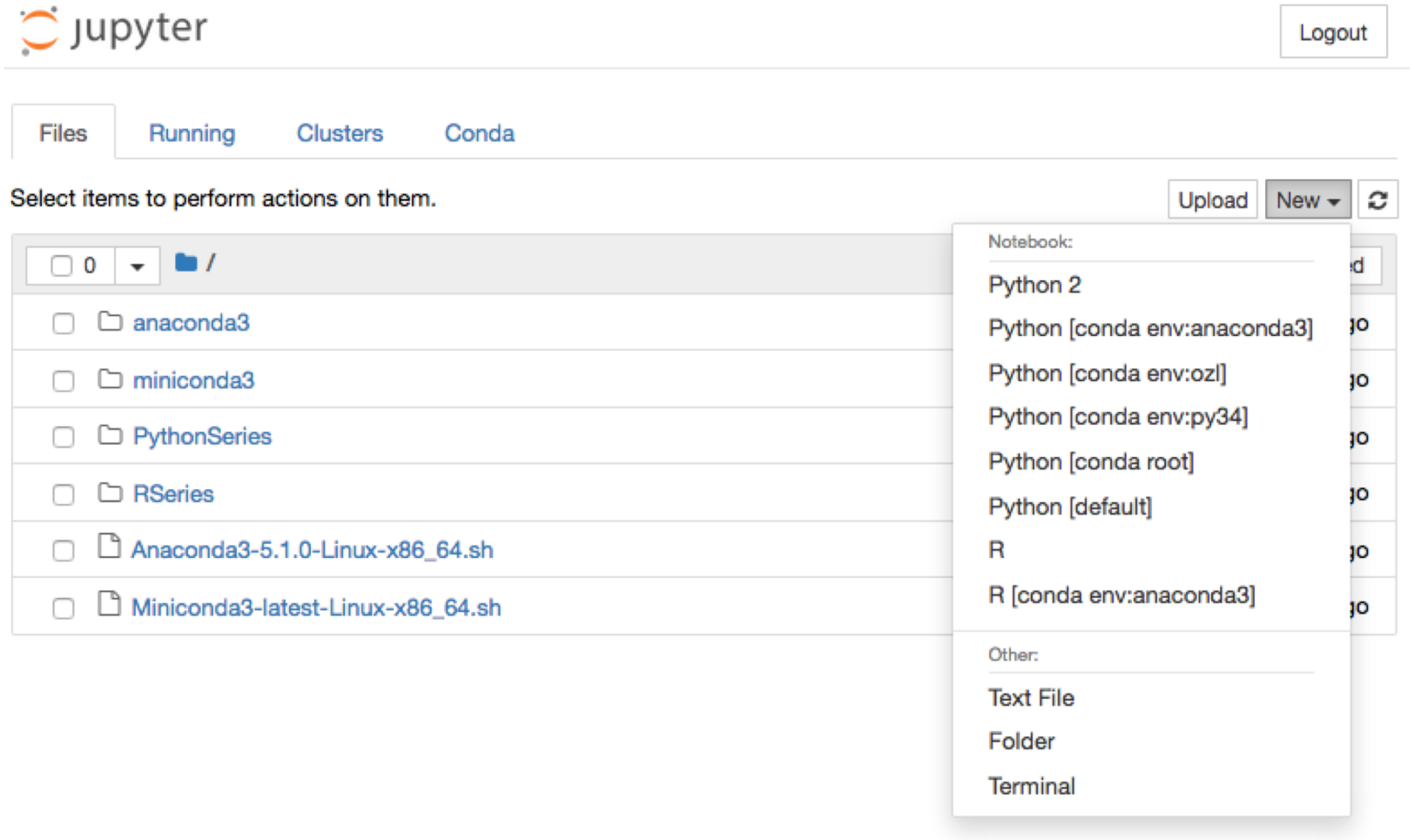
(1) conda install nb_conda_kernels

(2) Then from each environment
conda install ipykernel

The first step adds the functionality to the Jupyter notebooks to list the kernels, while the second step adds the environment to the list of kernels

Using environments with Jupyter

We can select a non-default kernel for our new notebooks



Using environments with Jupyter

We can also change the kernel for an open notebook