

How to live peacefully with Ana/Mini/Conda

or “How to completely wipe and reinstall your complete scientific Python stack in under 5 minutes” (if you have to)

K.-Michael Aye

Scripts at [this gist](#)

Conda vs Anaconda vs Miniconda

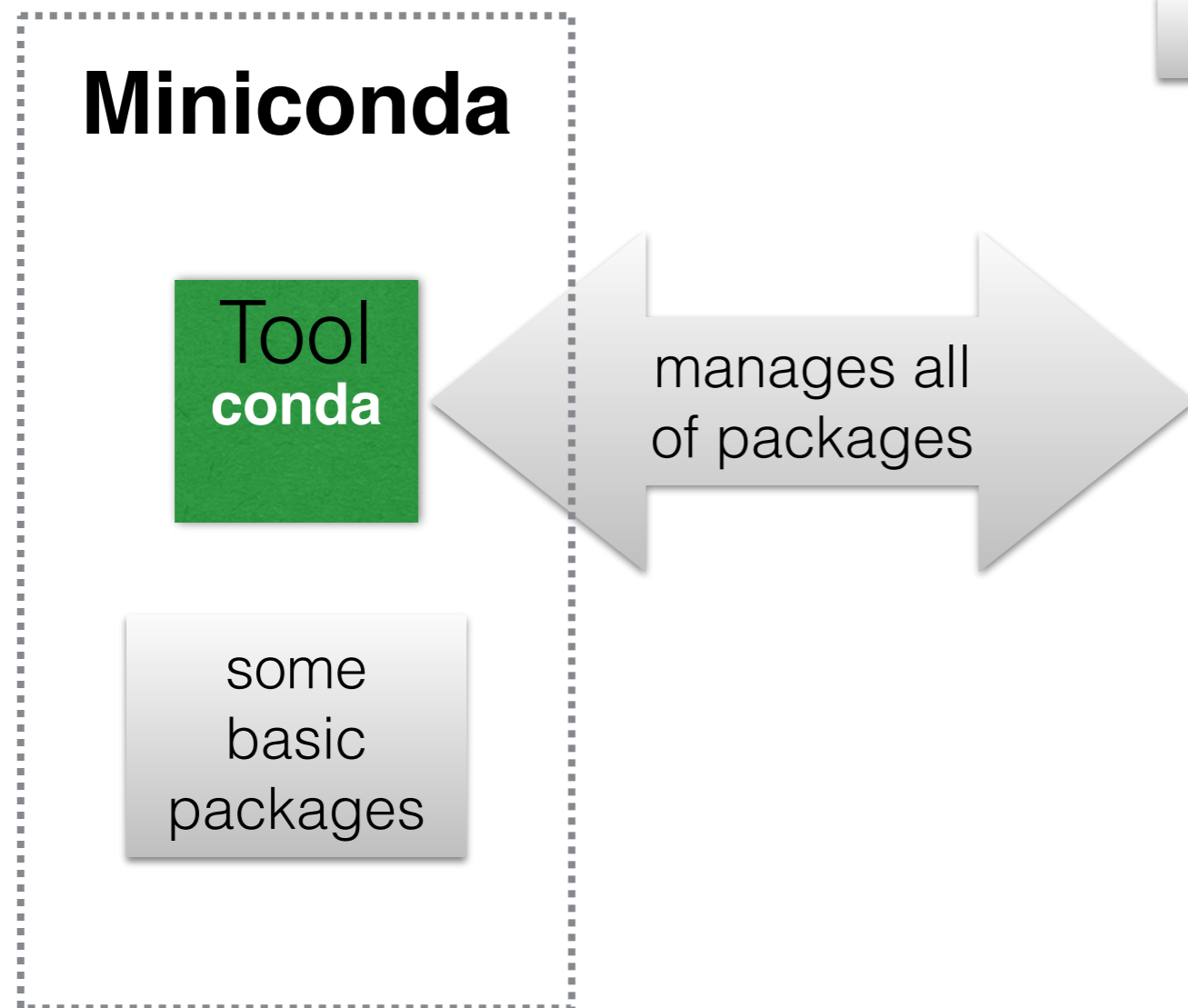
- Zoom polls:
- How many here use any conda-based system?
- How many understand the differences between the different conda terms in the title of this slide?
- Before we expunge/reinstall, let's understand it better.

The conda tool

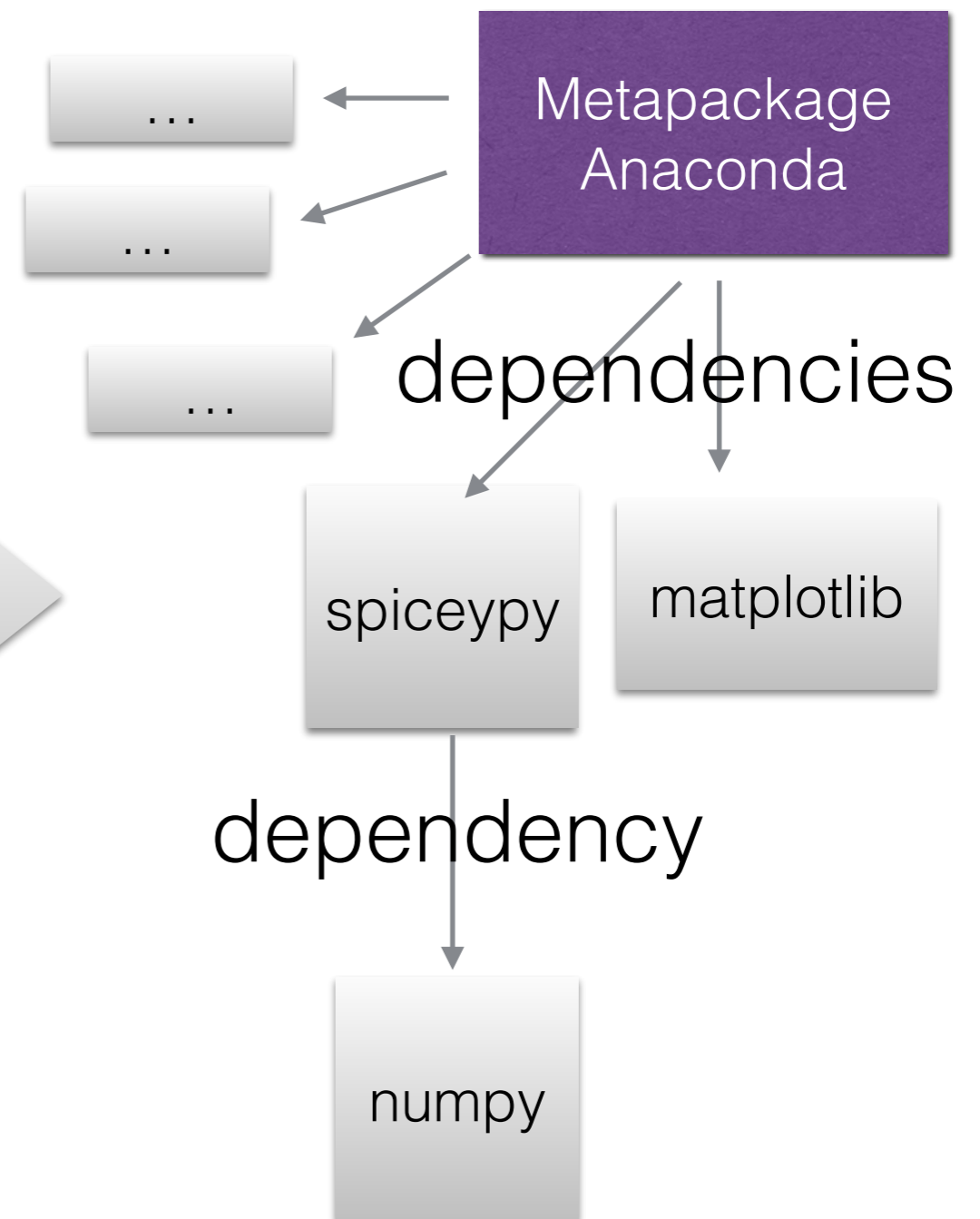
- Everything is centered around the `conda` tool.
- It's a command line tool, even when used from a GUI
- It is NOT focusing on Python, even though it's mostly used for that.
 - conda also manages/provides C++/FORTRAN library dependencies for a given Python package (e.g. for SciPy, GDAL, OpenCV)
 - There are also a lot of R conda packages now
- Completely removes the need for admin/root access

Conda vs Anaconda vs Miniconda (2)

Miniconda is a minimal set of pckgs, to be expanded by users that know what pckgs they need



Anaconda: best for beginners



Standard conda package (grey)

Conda vs Anaconda vs Miniconda (3)

- So, in summary:
 - conda is the **executable** that manages packages (not only Python, e.g. HDF binaries, FORTRAN, OpenCV, GDAL libraries etc.)
 - “miniconda” is a minimum set of packages for proper operation of conda, installed into a “base”. **Use this if you know which packages you need for your use case.**
 - “anaconda” is a meta-package with a huge list of scientific packages (dependencies) (**Recommended for beginners**)
 - Hence: **after installing miniconda and executing “conda install anaconda”, you would have the same python env as somebody that DL-ed the Anaconda distribution.**

conda environments

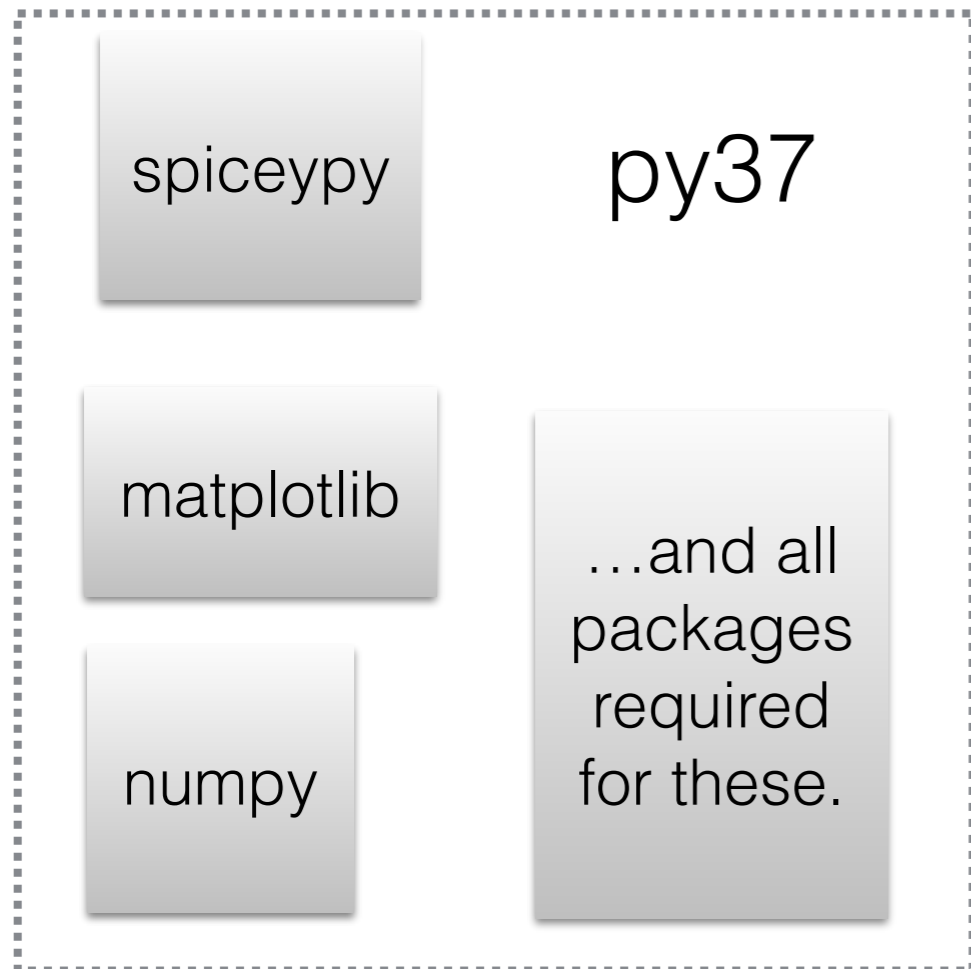
- conda's MO is based on (virtual) environments
- after installation, you have a "base" env.
 - if you still have an env named "root", I highly encourage you to wipe it all and reinstall
 - conda "suffered" from its own success:
 - rapid development led to breaking changes
 - these changes sometimes left incompatible remnants on your system
- same versions of pckgs are hardlinked between different envs, if possible
 - saves 50-70 % space, in my tests

Do you need envs?

- tl;dr : Yes
- Advanced (Python) hacking is NOT the criterium for using more than one env
- Not even "advanced Python usage" is.
- Piece of mind is the best criterium
 - ease of use is a supporting argument
- Why? Because the base environment is also the base for conda's functionality
 - That means, messing it up can render conda dysfunctional
 - -> reinstall of the whole ana/mini conda system might be required
 - While messing up one env (that isn't "base") only requires recreating the env

Creating envs

conda create -n py37 python=3.7 spiceypy matplotlib numpy



Legacy Python:

```
conda create -n py2 python=2  
python2_package
```

Use “**conda activate <env-name>**”
to switch between environments.

Demo time

- `conda env list`
- `conda search numpy`
- `conda search numpy=1.12 --info`
- `conda create -n anaconda_base python=3.7`
- `conda activate`
- `conda install numpy=1.14`

My python package search tree

- First conda: `conda install <pkg_name>`
 - The dependency resolver will tell if it would lead to downgrades of other packages, you can inspect and reject at this point.
- What if a Python package is not available conda?
 - `pip install pkg_name`
 - **NOTE: Always do conda activate <env_name> before this (or anything really).** Because otherwise a different “pip” command might be used on your computer and install goes somewhere else.
 - Pip ALWAYS depends on current active conda environment (or PATH if no conda)
 - If you ever did “pip install” and then Python couldn’t find it, it didn’t install where you think it did.
 - Always watch what pip is doing! It has a summary of changes at the end.
- What if pkg not even on Pypi server? Find it on GitHub:
 - `git clone <url_copied_from_GitHub> && cd <cloned_repo> && pip install (-e) .`
- I use this mix for many years successfully.... however `===>`

The conda-pip frenemies

- pip is good; it checks dependencies as well, and makes sure that you get the dependencies AS DEFINED BY THE PACKAGE AUTHOR
- pip is even recommended over "python setup.py install" for local (GitHub) package folders, because it keeps good records for uninstallation
 - "cd package_folder; pip install ." # note the dot!
- However, pip does NOT tell conda what it did.
- If a package author says it requires a lib version lower than you have, pip will replace it with the older version
 - While conda still thinks the newer one is installed!
- Lemma: NEVER run pip in the base env
 - Corrolary: You need envs! (proven theorem, or something like that...)

What if?

- So what do you do if pip replaced one library/package with an older version?
 - 1. `pip uninstall pck_name`
 - 2. `conda install pck_name --force-reinstall`
- If pip replaced several packages, i'd rather vote for env replacement
 - the dependency tree might be compromised beyond repair by conda

Summary of everyday conda tips

- If you have installed it before, and it's older than conda 4.6, remove and reinstall everything.
 - New version (now at 4.8.x) is much faster in adding a new package
 - Too many changes that make it better to delete “old cruft”
- If you still have changed PATH changes that point to your conda install in your ba/c/tc-sh configs, remove it!
 - Call of “conda init <shell_name>” configures things correctly, adding an init section to config files.
 - Leaving the manual PATH change in can create problems.
- Advice: Don't use the initial conda “base” environment for general work.
 - Eventually some of your installs (or Anaconda, Inc.) will mess up something.
- Always create a new default environment:
 - `conda create -n py37 python=3.7`
 - `conda activate py37`
- Find packages:
 - `conda search <package_name>`
 - If list shows what you need:
 - `conda install <package_name>` (will also drag in dependencies)

Keep your shell config clean

Below should be all there is related to conda in your shell config file (like .bashrc or similar)

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/maye/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/home/maye/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/home/maye/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/home/maye/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

Use this if you don't like the auto-activated environment:

```
conda config --set auto_activate_base false
```

Conda channels

- Channels are different locations/sources for packages.
- By default, an env is pointed to the default channel, you can confirm like so:
- The top-most channel has the highest priority for package searches.

```
$ conda config --show-sources
==> /Users/klay6683/.condarc <==
pinned_packages: []
report_errors: True
anaconda_upload: True

==> /Users/klay6683/miniconda3/envs/py37/.condarc <==
pinned_packages:
- conda-forge::bokeh
- conda-forge::opencv
- conda-forge::gdal
- conda-forge::numpy
channels:
- conda-forge
- defaults
```

Conda channels (2)

- What if you want to have one environment pointing to defaults and one to try out conda-forge?
 - -> ENV-dependent configuration!
- Activate the env you want to configure, then:
 - `conda config --env --add channels conda-forge`
- Good tip: Don't mix channels within one env. And pin packages to be sure.

Pinning packages

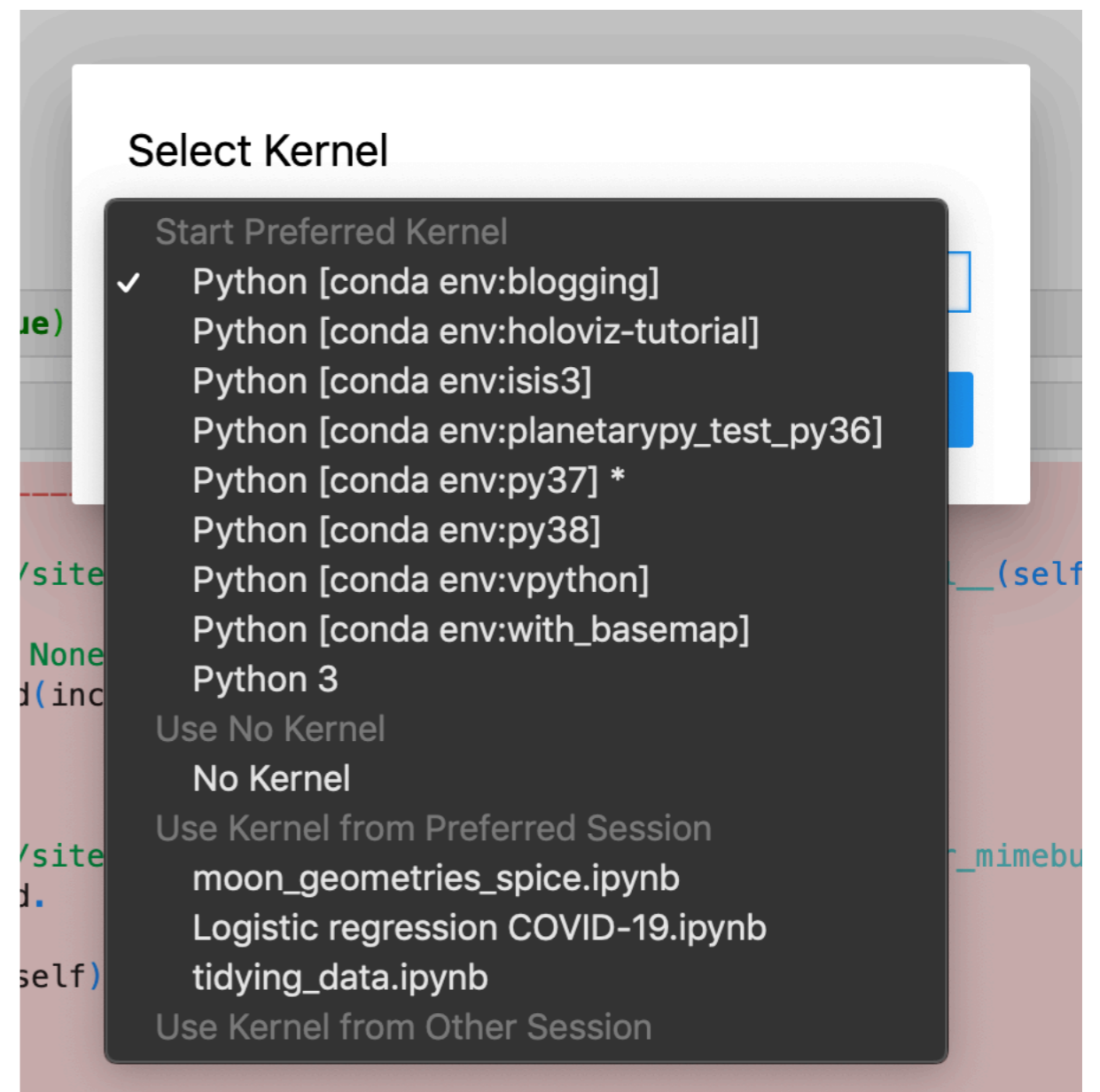
- Because conda sometimes finds “better” packages at default, conda did sometimes mix from different channels.
- To avoid this (catastrophic for gdal), pin packages per env to your desired channel:
 - `conda config --env --add pinned_packages conda-forge::gdal`

Demo time

- `conda create -n py38 python=3.8 -c conda-forge`
- `conda config --show-sources`
- `conda activate py38; conda config --show-sources`
- `conda config --env --add channels conda-config`
- `conda config --show-sources`
- `conda config --env --add pinned_packages conda-forge::gdal`
- `conda config --env --add pinned_packages "numpy>1.16"`
 - Sometimes needs `"numpy\>1.16"`, as some shells interpret `>` even inside `"`
- `conda config --show-sources`

nb_conda_kernels

- If you are working mostly in Jupyter and (anticipate to) have more than one conda env, this is the most important conda package.
 - It finds your existing conda envs at every launch of a Jupyter server
 - It then offers kernel for each conda env in the list
- JNotebook: find menu “Kernel->change kernel”
 - JLab: click on kernel name in the upper right



How to reinstall env in 5 min

- Even a “stable” env is rotting at some point
- The trick is to have:
 - File with a list of your conda packages
 - File with a list of your pip packages
 - If you develop new packages: file with a list of your own package folders and GH installs
- a (couple of) bash script(s)

How to reinstall env in 5 min(2)

```
#!/bin/bash
```

```
./Users/klay6683/miniconda3/etc/profile.d/conda.sh
```

```
if [ "$#" -ne 2 ]; then
    echo "Usage: $0 conda_env_name python_version_string (e.g. 3.7)" >&2
    exit 1
fi
if [ -n $CONDA_DEFAULT_ENV ]
then
    if [ "$CONDA_DEFAULT_ENV" == "$1" ]
    then
        echo "Deactive $1 environment first." >&2
        exit 1
    fi
fi
```

```
conda env remove -n "$1" -y
```

```
conda create -n "$1" -y -c conda-forge python="$2"
```

```
conda install -n "$1" -y -c conda-forge --file python_stuff/standard_py3_conda_packages.txt
```

- conda deactivate
- ./reinstall_env.sh py37 3.7
- conda activate py37
- ./install_my_libs.sh

```
1 v #!/bin/sh
2 # reinstall all my packages
3 cd ~/Dropbox/src
4 v for folder in 'pyciss' 'planet4' 'pyuvis' 'hirise_tools' 'nbtools' 'planetpy' 'pysis' 'p4terrains';
5     do cd $folder;
6         echo "Installing $folder";
7         echo;
8         pip install -e .;
9         cd .. ;
10        echo;
11    done
12 # reinstall packages from pip that are needed
13 pip install -r pip_packages_to_install.txt
14
```

Demo time (take my time!)

- `conda activate base`
- `conda env remove -n py37 # (my main dev env!)`
- `conda create -n py37 python=3.7 --file standard_py3_conda_packages.txt -c conda-forge`
- `conda activate py37`
- `./install_my_libs.sh # .fish in my case`