Tools for reproducible research

ISMB 2019
Devon Ryan, Björn Grüning, and Johannes Köster
Agenda

● Introduction to conda
  ○ Packages & channels
  ○ Environments
  ○ Writing recipes

● Introduction to Snakemake
  ○ Workflow definition
  ○ Workflow execution
  ○ Live demo
A “typical” irreproducible analysis

```
$ tree
.
├── analysis.2.sh
├── analysis.B.sh
├── analysis.sh
├── final_results.really_final.txt
├── final_results.txt
├── final_results.version2.txt
├── results
│   └── alignments.bam
├── results2
│   └── alignments.bam
├── results2b
│   └── alignments.bam
└── results3
    └── alignments.bam
```

```
for sample in `cut -f 1 sample_sheet.txt | sed "s#_1#_R1#" | uniq`; do
    bowtie2 -x $GENOME \
        -U $sample \n        -p 10 \n        $BT2PARAMS \n    | samtools view -o tmp/$sample.bam
done
```

- Which result is the “correct” one?
- How was it produced?
- What bowtie 2 version was used?
- What were the parameters?
- What does the analysis flow look like?
Reproducible research?

- The idea that all code and software needed to produce a given set of results are available and usable by others.
- “Crisis of reproducibility”
What’s needed for “reproducible research”?

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirement</th>
</tr>
</thead>
</table>
| Software  | - List of all required packages/versions  
            - Recreate environments on demand |
| Analysis  | - Tools for each step  
            - Parameters in each step  
            - Documentation of order, input, and output |
Why is this hard?

- Packages are hard to install
  - Dependency hell
  - Need root?
- Multiple versions
- Incompatible packages
- How do we recreate an analysis environment elsewhere?
- Automatic cloud/cluster deployment?
- Non-hack workflows
- Workflow portability?
- Workflow management of packages?
- Transparent scaling to cluster/cloud
How do Conda & Snakemake fit in?

<table>
<thead>
<tr>
<th>Component</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>- List of all required packages</td>
</tr>
<tr>
<td></td>
<td>- Recreate environments on demand</td>
</tr>
<tr>
<td></td>
<td>- Multiple versions/environments simultaneously</td>
</tr>
<tr>
<td>Analysis</td>
<td>- Tools for each step</td>
</tr>
<tr>
<td></td>
<td>- Parameters in each step</td>
</tr>
<tr>
<td></td>
<td>- Documentation of order, input, and output</td>
</tr>
<tr>
<td></td>
<td>- Archive environments</td>
</tr>
<tr>
<td></td>
<td>- Direct interaction with conda!</td>
</tr>
</tbody>
</table>
What exactly is Conda?

- A package and environment manager
  - Like apt/yum, but MUCH better
  - Environments are isolated from each other
- User-contributed package recipes
  - Different “channels”, can create your own
  - Updated constantly
- Prebuilt binaries
  - Linked to libraries in the same environment
Conda packages

- Specific versions
- Various sources (“channels”)
- Defined requirements
  - Usually from the same or predefined other channels

```
├── info
│   └── recipe
│       ├── conda_build_config.yaml
│       ├── meta.yaml
│       └── meta.yaml.template
├── python-scripts
└── site-packages
    ├── deeptools
    └── deepTools-3.3.0.dist-info
        └── LICENSE.txt
        └── zip-safe
```
Conda channels

Channel 1
- package-1.2.3
- package-1.2.2
- package-1.2.1
- package-1.2.0
- package-1.0.0alpha2
- package-1.0.0alpha1

Channel 2
- dependency-1.1.2
- package-1.2.3
- package-1.2.2
- package-1.2.1
- package-1.2.0
- salmon-0.14.0
- samtools-1.9
- snakemake-5.5.2

Channel 3
- gcc-4.7
- libcurl-7.6.41
- zlib-1.2.11
- numpy-1.16.4
- scipy-1.3.0
Conda channels

- Conda-forge: Most dependencies (numpy, scipy, zlib, CRAN packages, etc.)
- Bioconda: Most bioinfo packages (salmon, STAR, samtools, DESeq2, etc.)
- Defaults: Packages built by Anaconda Inc.

```bash
$ conda config --show channels
channels:
  - conda-forge
  - bioconda
  - defaults

$ conda config --add channels bioconda
```

- Order matters, use this one!
Finding packages

- Search on [http://anaconda.org](http://anaconda.org)
- Use `conda search`

```
$ conda search deepTools
Loading channels: done
# Name                       Version           Build  Channel
deeptools                      3.1.3  py36h14c3975_1  bioconda
deeptools                      3.1.3  py36h470a237_0  bioconda
deeptools                      3.1.3  py37h14c3975_1  bioconda
deeptools                      3.2.0            py_0  bioconda
deeptools                      3.2.1            py_0  bioconda
deeptools                      3.3.0            py_0  bioconda
```

- Packages have versions, build numbers and build hashes
  - Build hashes include dependency information
Practical 1

What are the most recent versions of STAR and Salmon?

```bash
$ conda search STAR | tail -n 1
star                2.7.1a               0  bioconda

$ conda search salmon | tail -n 1
salmon              0.14.1      h86b0361_1  bioconda
```
Conda environments

- A (mostly) self-contained directory with a set of compatible packages
- Often use links to reduce disk space

```
$ ls -i envs/*/bin/salmon pkgs/*/bin/salmon
20844181 envs/foo/bin/salmon
20844181 envs/salmon/bin/salmon
20844181 pkgs/salmon-0.14.1-h86b0361_1/bin/salmon
```
Conda environments

- Linking is relative to packages!
- No more conflicting dependencies between versions!

```bash
~/m3/envs/t/lib/python3.7/site-packages$ ldd -r pyBigWig.*.so
linux-vdso.so.1
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6
libz.so.1 => ~/m3/envs/t/lib/python3.7/site-packages/../../../../libz.so.1
libcurl.so.4 => ~/m3/envs/t/lib/python3.7/site-packages/../../../../libcurl.so.4
```
Conda environments - common commands

- conda env list
  - Lists available environments

```bash
$ conda env list
# conda environments:
#
base                  *  /home/dpryan/miniconda3
bioconda                 /home/dpryan/miniconda3/envs/bioconda
build                    /home/dpryan/miniconda3/envs/build
deepTools                /home/dpryan/miniconda3/envs/deepTools
foo                      /home/dpryan/miniconda3/envs/foo
salmon                   /home/dpryan/miniconda3/envs/salmon
twine                    /home/dpryan/miniconda3/envs/twine
```

- You start in “base”, a * indicates an active env
Conda environments - common commands

- conda create/conda env remove
  - Create/remove environments

$ conda create -n myenv python=3.7 numpy deepTools>=3.3.0
$ conda env remove -n myenv

- Packages can have versions specified
- Min/max versions can be specified

Tip: Specifying versions makes env creation faster!
Conda environments - common commands

- conda activate/deactivate
  - Activates/deactivates an environment

$ which deeptools
$ conda activate myenv
$ which deeptools
/home/dpryan/miniconda3/envs/myenv/bin/deeptools
$ conda deactivate
$ which deeptools

- You can “stack” environments with `--stack`
Conda environments - common commands

- conda install/uninstall
- conda list

```bash
$ conda activate myenv
$ conda install snakemake
... a lot of status output ...
$ conda list
... many packages ...
$ conda uninstall snakemake
```

Tip: Keep your “base” env clean, it will prevent headaches!
Practical 2

Create a new environment named “fondue” with hisat2, samtools and deepTools. What version of numpy got installed in it?

```bash
$ conda create -n fondue hisat2 samtools deepTools
$ conda activate fondue  # Yum!
$ conda list | grep numpy
numpy                     1.16.4           py27h95a1406_0    conda-forge
```
Conda environments - common commands

- conda env export/create
  - Exports an env to or creates an env from a YAML file

$ conda activate fondue
$ conda env export > environment.yaml
$ conda env create -f environment.yaml -n moreFondue
$ head environment.yaml
name: "fondue"
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - asn1crypto=0.24.0=py27_1003
  - attrs=19.1.0=py_0
  - backports=1.0=py_2
  - backports.functools_lru_cache=1.5=py_1
  - backports_abc=0.5=py_1
Common pitfalls

- Wrong channel order
- Installing packages in your base env
- Manually manipulating $PYTHONPATH
- Avoid manually installed packages
Conda package recipes

# Bioconda example
$ tree recipes/methyldackel
recipes/methyldackel/
  ├── build.sh
  └── meta.yaml

- meta.yaml is required
- optional:
  - build.sh
  - (small) test files
  - license
**meta.yaml sections**

- package: name and version
- source: url and sha256/md5
- build: build number, platforms to skip, “noarch” information
- requirements: packages for building, linking, running
- test: commands/imports
- about: Webpage, license, summary of what it does
- extras: Comments, maintainers, etc.
meta.yaml - package

```yaml
package:
  name: methyldackel
  version: 0.4.0
```

Can also use jinja2 variables:

```jinja
{% set name = "MethylDackel" %}
{% set version = "0.4.0" %}

package:
  name: {{ name|lower }}
  version: {{ version }}
```
Avoid `git_url` or `svn_url`

- Ony actual releases, no alpha/beta!
meta.yaml - build

- Reset to 0 with new releases
- Increment with each change
- Can skip conditions:

```yaml
build:
  number: 0
  skip: True  # [osx or py != 37]
```
meta.yaml - build

- "noarch" is useful, but confusing

build:
  noarch: generic
  number: 0

- generic: No platform-specific code (java, pure perl, pure R, etc.)

build:
  noarch: python
  number: 0

- python: Pure python packages, one build -> all versions
meta.yaml - build

- You can include the entire build command

```yaml
build:
  number: 0
  noarch: python
  script: "{{ PYTHON }} -m pip install . --no-deps --ignore-installed -vvv"
```

- Alternatively, use `build.sh`

```bash
$ cat build.sh
#!/bin/bash
$PYTHON -m pip install . --no-deps --ignore-installed -vvv
```

- `build.sh` is useful for "more involved" installs
- There are many environment variables: [https://bit.ly/2KUiDpd](https://bit.ly/2KUiDpd)
meta.yaml - requirements

```
requirements:
  build:
    - {{ compiler('c') }}
  host:
    - htslib
    - zlib
  run:
    - htslib
    - zlib
```

- build: Compilers, preprocessors, etc.
- host: Anything linked against
- run: All other dependencies
- Use compiler functions (`{{ compiler('cxx') }}` and such)
A note on "pinnings"

- Packages need to be compatible
  - Same compiler per-platform
  - Same htslib/numpy range/zlib/libcurl/etc. versions
  - Above are "pinned"

Wondered about the hash? It's the pinnings' sha256.

- Version ranges are great!
  - python
  - pybigwig >=0.2.3
  - numpy >=1.9.0
  - scipy >=0.17.0
  - matplotlib >=2.1.1
meta.yaml - test

```
test:
  imports:
    - deeptools
  commands:
    - bamCompare --version
```

- Keep it simple/quick but functional
- No large test files
- "imports" works for python/perl
meta.yml - about/extra

about:
  home: https://github.com/awesome/awesomeTools
  license: GPL3
  license_file: LICENSE
  summary: Awesome tools

extra:
  identifiers:
    - doi:10.1093/nar/gkw257
  recipe-maintainer:
    - your github handle

- Do try to package the license!
Don't fear skeletons

- Making recipes manually takes time
- Many common sources are automated

$ mkdir foo
$ cd foo
$ conda skeleton pypi deeptools

- Skeletons for: pypi, cpan, CRAN
  - For CRAN: https://github.com/bgruening/conda_r_skeleton_helper
  - CRAN packages belong on conda-forge if possible!
- We (bioconda) already make ALL bioconductor package
- On bioconda, recipes are in `recipes/`
Practical 3

Make a new cutadapt recipe

```bash
$ mkdir foo
$ cd foo
$ conda skeleton pypi cutadapt
$ vi cutadapt/meta.yaml  # edit noarch, host requirements, license_file
```
Common problems

- Compiling C/C++ packages are hard
- The compiler is NOT called "gcc"
- Installed into $PREFIX
- Dependencies are in $PREFIX/lib

```
$ cat build.sh
#!/bin/bash
make install CC=$CC \
  CXX=$CXX \n  CFLAGS="$CFLAGS" \n  CXXFLAGS="$CXXFLAGS" \n  prefix=$PREFIX
```