Tool Dependencies and Conda
Questions

- How can I connect tools to applications and libraries?
- What are the advantages of declaring dependencies for my tool?
- What are Conda and Bioconda?
- What are Conda recipes and environments?
- How do I find and use existing Conda recipes?
- How do I develop Conda recipes for use within Galaxy tools?
Objectives

- Learn to use existing Conda recipes to enable best practice tool dependency management in Galaxy.
- Learn the basics of building Conda recipes and contributing to Bioconda.
- Learn to use Planemo to assist in developing Galaxy tools from existing and new Conda recipes.
Planemo

These slides mirror the section on "Dependencies and Conda" in the Planemo Documentation.
Galaxy Dependencies
Example Tool (1 / 2)

From Planemo docs - the following example builds a tool for the seqtk seq command.

```bash
$ planemo tool_init --force \
    --id 'seqtk_seq' \
    --name 'Convert to FASTA (seqtk)' \
    --requirement seqtk@1.2 \
    --example_command 'seqtk seq -a 2.fastq > 2.fasta' \
    --example_input 2.fastq \
    --example_output 2.fasta \
    --test_case \
    --cite_url 'https://github.com/lh3/seqtk' \
    --help_from_command 'seqtk seq'
```

Notice the --requirement seqtk@1.2.
Example Tool (2 / 2)

The --requirement seqtk@1.2 gets translated into the following Galaxy tool XML:

```xml
<requirements>
    <requirement type="package" version="1.2">seqtk</requirement>
</requirements>
```
Dependency Resolution

ToolBox

- seqtk_seq.xml
- seqtk_subseq.xml
- seqtk_sample.xml
  ...
- bwa.xml
- samtools_flagstat.xml
  ...

Applications & Libraries

- seqtk
- bwa
- samtools

Dependency Resolution via Requirement Tags
CONDA

Package, dependency and environment management
Conda recipes build packages that are published to channels.
No compilation at install time - *binaries* with their dependencies, libraries...

Support for all operating systems Galaxy targets

Easy to manage *multiple versions* of the same recipe

HPC-ready: no root privileges needed

Easy-to-write YAML recipes

Vibrant Communities
• Packages through channels within Continuum.

• Conda channels searched by Galaxy for packages
  ◦ iuc
  ◦ bioconda
  ◦ conda-forge
  ◦ defaults

If you are interested in Natural Language Processing or Cheminformatics you may be asking if these channels can still work for your tools. Despite the name Bioconda - it is really more about community and a set of best practices than about bioinformatics purity - many diverse packages have been integrated.
Planemo installs Conda using miniconda and configured defaults designed to easy development.

This has already been done on Planemo machine.
Using Conda outside Planemo

• Install a package

$ conda install pyyaml

• Install some packages within an isolated environment

$ conda create -n yaml pyyaml
$ conda env list
yaml * ~/miniconda3/envs/yaml
root ~/miniconda3
$ source activate yaml
(yaml) $
Conda and Galaxy

Galaxy now automatically installs Conda when first launched and will use Bioconda and other channels for package resolution.
# Installing Tools with Conda

The following tool dependencies are required by the current repository:

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Version</th>
<th>Current Installation Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>hisat2</td>
<td>None</td>
<td>Not Installed</td>
</tr>
<tr>
<td>samtools</td>
<td>None</td>
<td>Not Installed</td>
</tr>
</tbody>
</table>

## Confirm dependency installation

By default Galaxy will install all needed dependencies for the repository. See the [dependency resolver documentation](#) .

You can control how dependencies are installed (this is an advanced option, if in doubt, use the default) [Display Details]

## Choose the tool panel section to contain the installed tools (optional)

### Add new tool panel section:

Add a new tool panel section to contain the installed tools (optional).

### Select existing tool panel section:

[Get Data](#)

Choose an existing section in your tool panel to contain the installed tools (optional).

[Install](#)

Clicking [Install] without selecting a tool panel section will load the installed tools into the tool panel outside of any sections.
Managing Tool Dependencies

This page gives an overview of all tool dependencies required by all tools currently loaded, including tools not installed through the Tool Shed.

Dependency details

<table>
<thead>
<tr>
<th>Select</th>
<th>Used by</th>
<th>Environment Path</th>
<th>Requirement</th>
<th>Version</th>
<th>Resolver</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Baspin</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._uucsc_tools@332</td>
<td>guspin</td>
<td>None</td>
<td>None</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Trimmomatic</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._trimmomatic@_uucsc_tools@332</td>
<td>trimmomatic</td>
<td>0.36</td>
<td>Conda</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>Stringtie</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._stringtie@_uucsc_tools@332</td>
<td>stringtie</td>
<td>1.1.0</td>
<td>Tool_Shed_Package</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Cufflinks</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._cufflinks@_uucsc_tools@332</td>
<td>cufflinks</td>
<td>None</td>
<td>None</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Wg_bedGraph-to-bigWig, Convert BedGraph to BigWig, BED-to-bigBed, Convert Wiggle to BigWig, Extract Genomic DNA</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>None</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>HSAAT2</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._hsaat2@_uucsc_tools@332</td>
<td>hsaat2</td>
<td>None</td>
<td>None</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Bar chart</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>None</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Convert FASTA to Bowtie base space index, Convert FASTA to Bowtie color space Index</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>None</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Cuffdiff</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>None</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Convert FASTA to fast file</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>None</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Trimmomatic</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>332</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Convert SAM to BigWig</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>332</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Convert genomic intervals to coverage, Convert BAM to BigWig, Convert BED, GFF, or VCF to BigWig</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>332</td>
<td>Conda</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>Convert FASTA to Zbit</td>
<td>/Users/john/workspace/galaxy/database/dependencies/conda/envi/._ucsc_tools@332</td>
<td>uucsc_tools</td>
<td>332</td>
<td>Conda</td>
<td>True</td>
</tr>
</tbody>
</table>
Conda and Planemo

Using Conda directly is generally package-centric, Planemo provides abstractions that are tool-centric.
The next few slides will use the seqtk example from Planemo's documentation - this can be downloaded to follow along using the following command:

```
$ planemo project_init --template=seqtk_complete seqtk_example
$ cd seqtk_example
```
Linting Conda Dependencies

Planemo can check if the requirements of a tool are available in best practice Conda channels using the `--conda_requirements` flag of `planemo lint`.

```bash
$ planemo lint --conda_requirements seqtk_seq.xml
Linting tool /Users/john/workspace/planemo/docs/writing/seqtk_seq_v6.xml
...
Applying linter requirements_in_conda... CHECK
.. INFO: Requirement [seqtk@1.2] matches target in best practice Conda channel [bioconductor]
```

Notice *Planemo indicates this tool is available and shows the channel it is available in.*
The Planemo conda_install command

$ planemo conda_install seqtk_seq.xml
Install conda target CondaTarget[seqtk,version=1.2]
/home/john/miniconda3/bin/conda create -y --name __seqtk@1.2 seqtk=1.2
Fetching package metadata ..............
Solving package specifications: ...........
Package plan for installation in environment /home/john/miniconda3/envs/__seqtk@1.2:
The following packages will be downloaded:
  package | build
  --------------- | ---------------
  seqtk-1.2 | 0 29 KB bioconda
The following NEW packages will be INSTALLED:
  seqtk: 1.2-0 bioconda
  zlib: 1.2.8-3
Fetching packages ...
....
#
# To deactivate this environment, use:
# > source deactivate __seqtk@1.2
#
$ which seqtk
seqtk not found

Notice seqtk hasn't been placed on the PATH, an environment has been setup that Galaxy (when used through Planemo) can leverage.
The Planemo `conda_env` command

$ . |<in.fa>

Options:  
- `-q INT` mask bases with quality lower than INT [0]
- `-X INT` mask bases with quality higher than INT [255]
- `-n CHAR` masked bases converted to CHAR; 0 for lowercase [ ]
- `-l INT` number of residues per line; 0 for 2^32-1 [0]

...  
- `-V` **shift** quality by `'(Q) - 33'`
- `-U` convert all bases to uppcases
- `-S` strip of white spaces in sequences

(seqtk_seq) $ conda_env_deactivate

$
Using the Tool Environment

Now that we have verified the Conda environment setup with conda_install works properly on the command-line, we can use our tool!

planemo test and planemo serve will use this environment by default now for this tool.
Planemo test

$ planemo test seqtk_seq.xml
...
INFO [galaxy.tools.actions] Handled output named output1 for tool seqtk_seq (20.136 ms)
INFO [galaxy.tools.actions] Added output datasets to history (12.782 ms)
INFO [galaxy.tools.actions] Verified access to datasets for Job[unflushed,tool_id=seqtk_seq] (10.954 ms)
INFO [galaxy.tools.actions] Setup for job Job[unflushed,tool_id=seqtk_seq] complete, ready to flush (21.053 ms)
INFO [galaxy.tools.actions] Flushed transaction for job Job[id=2,tool_id=seqtk_seq] (26.510 ms)
INFO [galaxy.jobs.handler] (2) Job dispatched
DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda
DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda
INFO [galaxy.jobs.command_factory] Built script [/tmp/tmpLvKwta/job_working_directory/000/2/tool_script.sh] for tool command
DEBUG [galaxy.tools.deps] Using dependency samtools version None of type conda
DEBUG [galaxy.tools.deps] Using dependency samtools version None of type conda
ok

--------------------
XML: /private/tmp/tmpLvKwta/xunit.xml
--------------------
Ran 1 test in 15.936s
OK

The following line indicates the seqtk package was found:

[galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda
Hands-on
Hands-on

The Goal

- Use the Planemo commands `conda_install`, `conda_env`, and test to practice the Galaxy tool dependency development lifecycle.
**Steps**

Run the following commands to practice working with Galaxy tools, Planemo, and Conda.

```sh
$ planemo project_init --template=seqtk_complete seqtk_example

$ cd seqtk_example

$ planemo conda_install seqtk_seq.xml

$. <(planemo conda_env seqtk_seq.xml)

$ planemo test seqtk_seq.xml
```
Finding the Correct Requirements & Packages

The previous example worked because a published Bioconda recipe named seqtk at version 1.2 was previously published, but how can these be found?

Two easy approaches are using `planemo conda_search` and using the Anaconda web search.
Using the Planemo conda_search Command

The Planemo conda_search command is a shortcut around conda_search that searches best practice channels that Galaxy is configured to work with:

```
$ planemo conda_search seqt
Fetching package metadata ...............  
seqtk              r75                       0  bioconda  
r82                0  bioconda  
r93                0  bioconda  
1.2                0  bioconda  
```

Alternatively, conda can be used directly:

```
$ $HOME/miniconda3/bin/conda search -c iuc -c bioconda -c conda-forge seqtk
```
Using Anaconda Search - https://anaconda.org

Why you'll love Anaconda Cloud
Making it easy to share packages, notebooks, and environments to be more collaborative.
Using Anaconda Search - https://anaconda.org
Hands-on
The Goal

- Find the correct package and version for a tool in a best practice channel.
- Add a requirement to a tool to allow Galaxy to find, install, and use a Conda package.
Hands-on

Steps

1. Run the following commands to download an example tool to modify.

```
$ planemo project_init --template conda_exercises conda_exercises
$ cd conda_exercises/exercise1
$ ls
pear.xml              test-data
```

2. Run `planemo test pear.xml` to verify the tool does not function without dependencies defined.
3. Use `--conda_requirements` flag with `planemo lint` to verify it does indeed lack requirements.
4. Use `planemo conda_search` or the Anaconda website to search for the correct package and version in a best practice channel.
5. Update `pear.xml` with the correct requirement tags.
6. Re-run the `lint` command from above to verify the tool now has the correct dependency definition.
7. Re-run the `test` command from above to verify the tool test now works properly.
If searching best practice channels fails, you may need to build a Conda recipe.

Documentation about how to write conda recipes from scratch
Bioconda guidelines
A Conda recipe is defined by a directory, the two most important files in this directory are:

- `meta.yaml`: contains all the metadata of the recipe
- `build.sh`: the Unix script that installs the files
meta.yaml contains basic metadata about the recipe.

```yaml
package:
  name: bwa
  version: "0.7.15"
about:
  license: MIT
  summary: The BWA read mapper.
build:
  number: 0
  skip: True # [osx]
source:
  fn: v0.7.15.tar.gz
  url: https://github.com/lh3/bwa/archive/v0.7.15.tar.gz
  md5: 54fdee953c5c256d36885a1c5c6b118c
```
Writing a Conda Recipe - requirements

```
requirements:
  build:
    - gcc  # [not osx]
    - llvm  # [osx]
    - zlib
  run:
    - libgcc  # [not osx]
    - zlib
    - bowtie2
```

- **build**: requirements needed during the build step (here compilation)
- **run**: requirements needed at runtime
Writing a Conda Recipe - Preprocessing Selectors

- Evaluated as Python expressions - feel free to use and, or, etc.
- Other common selectors include py2k, py3k.

```yaml
requirements:
  build:
    - gcc   # [not osx]
    - llvm  # [osx]
    - zlib

build:
  skip: True  # [not py27]
```

https://conda.io/docs/building/meta-yaml.html#preprocessing-selectors
**meta.yaml** should contain simple tests. These are commands executed at the end of `conda build` and expected to return 0 on success.

```yaml
test:
  commands:
    - bowtie2 --version

# another test

test:
  commands:
    - bwa 2>&1 | grep 'index sequences in the'

# R testing

test:
  commands:
    - '$R -e "library(''xcms'')"'
```

Please note that the Conda tests run inside the runtime environment and not in the build environment.

https://conda.io/docs/building/meta-yaml.html#test-section
From Bioconda Guidelines:

An adequate test must be included in the recipe. An "adequate" test depends on the recipe, but must be able to detect a successful installation. While many packages may ship their own test suite (unit tests or otherwise), including these in the recipe is not recommended since it may timeout the build system on CircleCI. We especially want to avoid including any kind of test data in the repository.

https://bioconda.github.io/guidelines.html
Writing a Conda Recipe - build.sh

```bash
#!/bin/bash
./configure --prefix=$PREFIX
make
make install
```

```bash
#!/bin/bash
mkdir -p $PREFIX/bin
cp *.py $PREFIX/bin
```

- `$PREFIX` is a variable defined by conda when building the package.
- `$PREFIX` usually contains bin/, lib/, and include/ that are filled with the package files.
- `$PREFIX` will be loaded in the user environment at runtime: PATH, LD_LIBRARY_PATH, PYTHONPATH, ...

http://training.galaxyproject.org/training-material/topics/dev/tutorials/conda_sys/slides.html
$ conda skeleton pypi <packagename>

$ conda skeleton cran <packagename>

$ bioconductor_skeleton.py <packagename>

$ conda skeleton cpan <packagename>`

These generate pre-filled recipes (not guaranteed to work out of the box) for specific programming environments.

Building conda packages with conda skeleton
Once the recipe is ready to go, the `conda build` command can be used to build it.

$ $HOME/miniconda3/bin/conda build .

1. BUILD START: Builds/Compiles the package
2. BUILD START: Provides a .tar.bz2
3. TEST START: Installs the .tar.bz2 previously generated
4. TEST START: Launches the functional tests
5. (Provides the .tar.bz2 path)

If miniconda wasn't configured with `planemo conda_init`, you may have to run `conda install conda-build` before using the above command.
A channel dedicated to bioinformatics (and other informatics) packages

- https://bioconda.github.io
- https://anaconda.org/bioconda
- Open to contribution
- GitHub repository: https://github.com/bioconda/bioconda-recipes
Fork Bioconda.

Clone your fork:

$ git clone https://github.com/ <myuser> /bioconda-recipes

Create a new branch package

$ git checkout -b package

Fill two files meta.yaml and build.sh in a new recipe directory

Build your new package and test it using conda build
Commit and push to GitHub

```bash
$ git add recipe
$ git commit -m 'my recipe description'
$ git push origin package
```

Create a Pull Request

After the PR is merged, wait for the functional tests to pass on the master branch

Enjoy your new Conda package at https://anaconda.org/bioconda

See Contributing with GitHub
Planemo and `--conda_use_local`

By default, Galaxy and Planemo will ignore locally built packages.

Simply pass `--conda_use_local` to various Planemo commands (e.g. test, conda_install, or serve) to use the local package cache.

*Enables developing Galaxy tools and Conda recipes in parallel.*
Hands-on
Hands-on

The Goal

- Implement and test a local Conda recipe.
- Use Planemo and Galaxy with a locally built package.
Before

If you have completed exercise1, open exercise2.

$ cd ../exercise2
$ ls
fleeqtk Seq.xml test-data

This directory contains the outline of a tool for `fleeqtk`. `fleeqtk` is a fork of the project seqtk that many Planemo tutorials are built around and the example tool `fleeqtk_seq.xml` should be fairly familiar.
Steps

2. Build a recipe for fleeqtk version 1.3. You may wish to use conda skeleton, start from scratch, or copy the recipe of seqtk and work from there - any of these strategies should work
   - fleeqtk 1.3 can be downloaded using the URL https://github.com/jmchilton/fleeqtk/archive/v1.3.tar.gz
   - fleeqtk can be built using make and installed with make install
3. Use conda build to build the recipe
4. Add a requirement for this new package in the example tool.
5. Run planemo conda_install --conda_use_local fleeqtk_seq.xml to install the package for Galaxy
6. Run planemo test fleeqtk_seq.xml to verify the tool and package work together
Advanced Topics in Conda Development
Jinja Templating

```{% set name = "seqtk" %}
{% set version = "1.15.1" %}

package:
    name: {{ name }}
    version: {{ version }}

source:
    fn: {{ name }}-{{ version }}.zip
    url: http://coolsoftware.com/{{ name }}/{{ version }}/{{ name }}-{{ version }}

https://conda.io/docs/building/meta-yaml.html#templating-with-jinja
Bioconda Variables

CONDA_PY:
  - 27
  - 35
  - 36
CONDA_HTSLIB: "1.7"
CONDA_BOOST: "1.64"
CONDA_R: "3.4.1"
CONDA_PERL: "5.22.0"
CONDA_NPY: "112"
CONDA_NCURSES: "5.9"
CONDA_GSL: "1.16"
CONDA_GMP: "5.1"
CONDA_HDF5: "1.8.17"
MACOSX_DEPLOYMENT_TARGET: "10.9"
CONDA_ZLIB: "1.2.11"
CONDA_BZIP2: "1.0"
CONDA_XZ: "5.2"
CONDA_BAMTOOLS: "2.4.1"

Pinning Libraries - Use jinja

If your package links dynamically against a particular library, it is often necessary to pin the version against which it was compiled, in order to avoid ABI incompatibilities. Instead of hardcoding a particular version in the recipe, we use jinja templates to achieve this. This helps ensure that all bioconda packages are binary-compatible with each other. For example, bioconda provides an environment variable CONDA_ZLIB that contains the current major version of Zlib.

```
requirements:
  build:
    - zlib {{ CONDA_ZLIB }}*
  run:
    - zlib {{ CONDA_ZLIB }}*
```

https://bioconda.github.io/guidelines.html#c-c
While supported by Conda, `git_url` and `git_rev` are not as stable as a git tarball. Ideally a github repo should have tagged releases that are accessible as tarballs from the “releases” section of the github repo. In addition tarballs can be easily mirrored and Bioconda is saving a copy of every tarball so the recipe can be rebuild at any time.
Python

For PyPI packages

```
conda skeleton pypi <package_name>
```

- Builds likely correct `build.sh` and `meta.yaml`.
- The test automatically added is probably sufficient for library, may need to write extra tests for command-line tools.
- Recipes requiring `python` should build on Python 2.7, 3.5, and 3.6 by default, preprocessing selectors can be used with `build: skip` to skip targets.
Python - pysam's `build.sh`

```bash
#!/bin/bash
# Remove gcc statements that do not work on older compilers for CentOS5
# support
sed -i'' -e 's/-Wno-error=declaration-after-statement","g' setup.py
sed -i'' -e 's/-Wno-error=declaration-after-statement"//g' setup.py
# linking htslib, see:
# https://pysam.readthedocs.org/en/latest/installation.html#external
# https://github.com/pysam-developers/pysam/blob/v0.9.0/setup.py#L79
export CFLAGS="-I$PREFIX/include"
export CPPFLAGS="-I$PREFIX/include"
export LDFLAGS="-L$PREFIX/lib"

export HTSLIB_LIBRARY_DIR=$PREFIX/lib
export HTSLIB_INCLUDE_DIR=$PREFIX/include
$PYTHON setup.py install
```
Python - pysam's meta.yaml

package:
  name: pysam
  version: 0.11.2.2

source:
  fn: pysam-0.11.2.2.tar.gz
  url: https://pypi.python.org/packages/a4/1b/b6dfd92aea876647d20d9a8bd8618e4f2af6380539426be83c8bb0912d6f/pysam-0.11.2.2.tar.gz
  md5: 5623cd5f55b503845915b76c22d620a
  patches:
    - osx_rpath.patch [osx]

build:
  number: 0
  skip: False

requirements:
  build:
    - gcc # [llinux]
    - llvm # [osx]
    - htslib >=1.4.1,<1.5
    - samtools >=1.4.1,<1.5
    - bcftools >=1.4.1,<1.5
    - python
    - setuptools
    - zlib
    - curl
  run:
    - llbgcc # [llinux]
    - htslib >=1.4.1,<1.5
    - samtools >=1.4.1,<1.5
    - bcftools >=1.4.1,<1.5
    - python
    - zlib
    - curl
  test:
    imports:
      - pysam
For CRAN packages

conda skeleton cran <packagename>

- Builds likely correct build.sh and meta.yaml.
- The recipe name will have an `r-` prefix and will be converted to lowercase.
- Typically can be used without modification, though dependencies may also need recipes.

https://bioconda.github.io/guidelines.html#r-cran
R-locfit's build.sh

#!/bin/bash

# R refuses to build packages that mark themselves as Priority: Recommended
mv DESCRIPTION DESCRIPTION.old
grep -v '^Priority: ' DESCRIPTION.old > DESCRIPTION

$R CMD INSTALL --build .
package:
  name: r-locfit
  # Note that conda versions cannot contain -, so any '-'s in the version have been replaced with '_'s.
  version: "1.5_9.1"

source:
  url: https://cran.r-project.org/src/contrib/locfit_1.5-9.1.tar.gz
  md5: 38af7791c9cda78e2804020ae65ac7fb4

build:
  number: 0
  rpaths:
    - lib/R/lib/
    - lib/

requirements:
  build:
    - gcc
    - r-base
    - r-lattice
  run:
    - libgcc
    - r-base
    - r-lattice

test:
  commands:
    - "$R -e 'library('locfit')'"

about:
  home: https://cran.rstudio.com/web/packages/locfit/index.html
  license: GPL (>= 2)
  summary: 'Local regression, likelihood and density estimation.'
Java

- New recipes should use the openjdk package from conda-forge.
- Add a wrapper script if the software is typically called via java -jar.
- JAR files should go in $PREFIX/share/$PKG_NAME-$PKG_VERSION-$PKG_BUILDNUM
- A wrapper script should be placed here as well, and symlinked to $PREFIX/bin.

https://bioconda.github.io/guidelines.html#java
Java - PeptideShaker's build.sh

```bash
#!/bin/bash
set -eu -o pipefail

outdir=${PREFIX}/share/${PKG_NAME}-${PKG_VERSION}-${PKG_BUILDNUM}
mkdir -p $outdir
mkdir -p ${PREFIX}/bin
cp -R * $outdir/
cp ${RECIPE_DIR}/peptide-shaker.py $outdir/peptide-shaker
ls -l $outdir
ln -s $outdir/peptide-shaker ${PREFIX}/bin
chmod 0755 "${PREFIX}/bin/peptide-shaker"
```
Java - PeptideShaker's meta.yaml

...
Perl
For CPAN packages

conda skeleton cpan <packagename>

- Builds likely correct build.sh and meta.yaml.
- The recipe will have the perl- prefix.
Perl - Module-Build

package:
  name: perl-module-build
  version: "0.4214"

source:
  fn: Module-Build-0.4214.tar.gz
  url: https://cpan.metacpan.org/authors/id/L/LE/LEONT/Module-Build-0.4214.tar.gz
  md5: 7b7ca5a47bef48c50c8b5906ca3ac7fb

build:
  number: 0

requirements:
  build:
    - gcc
    - perl
  run:
    - libgcc
    - perl

test:
  # Perl 'use' tests
  imports:
    - Module::Build
    - Module::Build::Base
    - Module::Build::Compat
    - Module::Build::Config
    - Module::Build::Cookbook
    - Module::Build::Dumper
    - Module::Build::Notes
    - Module::Build::PPMMaker

about:
  home: https://metacpan.org/pod/Module::Build
  license: perl_5
  summary: 'Build and install Perl modules'
Metapackages

Metapackages tie together other packages. All they do is define dependencies. For example, the hubward-all metapackage specifies the various other conda packages needed to get full hubward installation running just by installing one package.

Other metapackages might tie together conda packages with a theme. For example, all UCSC utilities related to bigBed files, or a set of packages useful for variant calling.

https://bioconda.github.io/guidelines.html#metapackages
CircleCI Continuous Building

- Lint recipes.
- Build and run tests.
- Build and publish Docker container.
- Publish to anaconda.org.
CircleCI Command Line Interface (CLI)

- Installation:

  ```bash
  $ curl -o /usr/local/bin/circleci https://circle-downloads.s3.amazonaws.com/releases/build_agent_wrapper/circleci
  $ chmod +x /usr/local/bin/circleci
  ```

- The extended building and testing done by CircleCI can be executed locally using the CircleCI CLI in the root directory of Bioconda.

- It should be run from the top-level dir of the repo.

- Build and test recipes:

  ```bash
  $ circleci build
  ```
Key points

- Conda and Bioconda are Galaxy best practices for connecting Galaxy tools to underlying applications and libraries.

- Leveraging Conda allows easy installation of your tool's dependencies by Galaxy deployers.

- The Planemo commands `conda_search`, `conda_init`, `conda_install`, `lint`, `test`, and `serve` make it easy to search and use existing Conda recipes when developing tools.

- Conda recipe skeletons, `conda_build`, and with `planemo conda_install --conda_use_local` allow easy development of new Conda recipes at the same time as Galaxy tools that wrap them.

- Bioconda is a Galaxy best practice Conda channel for recipe publication.

- Bioconda has easy to follow contribution guidelines and is very welcoming to new contributors.
Thank you!

This material is the result of a collaborative work. Thanks the Galaxy Training Network and all the contributors (Nicola Soranzo, John Chilton, Björn Grünling, Hervé Ménager)!

Found a typo? Something is wrong in this tutorial? Edit it on GitHub