BIOS 259: The Art of Reproducible Science

A Hands-on Approach | A Stanford Biosciences Mini-course

Day 03: Dependency management using **Conda**

Aziz Khan <azizk@stanford.edu>

https://github.com/asntech/bios259-w24/

Course schedule

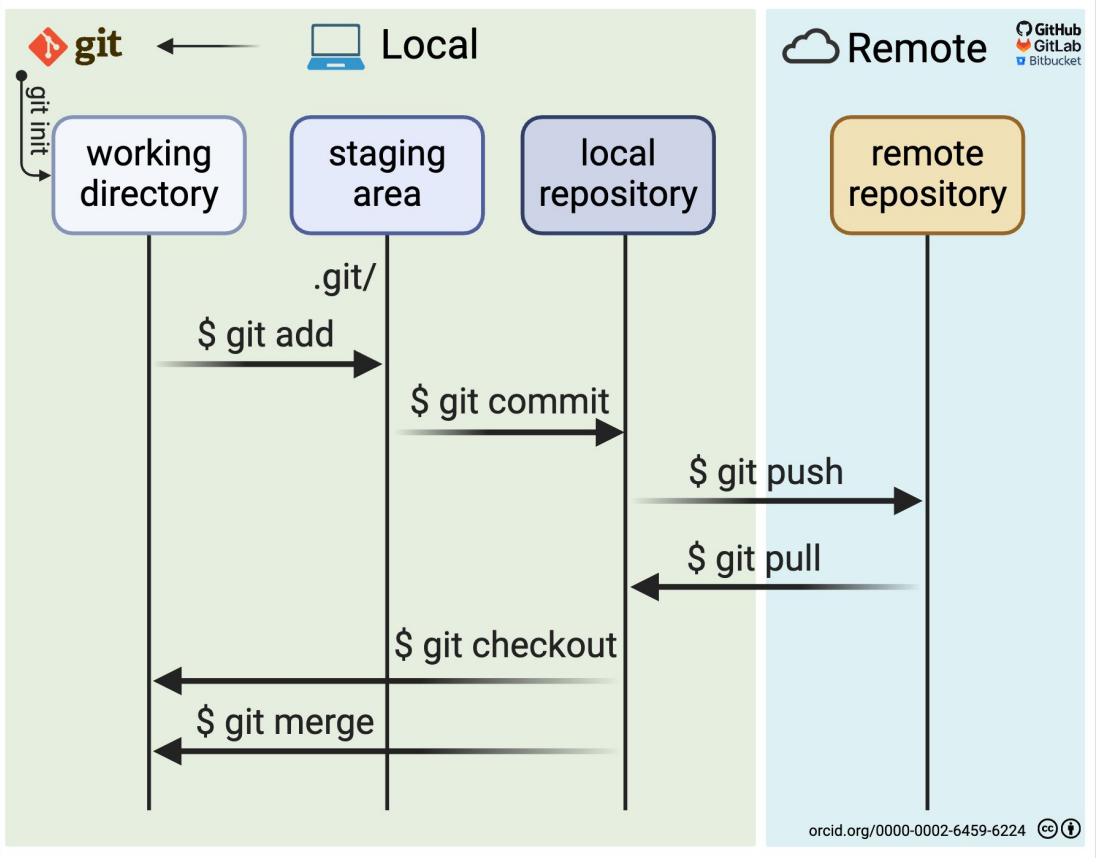
Date	Topic	Time	Location
02.26.2024 – Monday	Introduction to reproducibility and setting up	10:00-13:00	Alway Building M218A
02.28.2024 – Wednesday	Version Control (Git/GitHub)	10:00-13:00	M218A
03.01.2024 – Friday	Dependency management (Conda, Mamba, Bioconda)	10:00-13:00	M218A
03.04.2024 – Monday	Containerization (Docker, Singularity)	10:00-13:00	M218A
03.06.2024 – Wednesday	Workflows (Snakemake, Nextflow/nf-core)	10:00-13:00	M218A
03.08.2024 – Friday	Documentation (FAIR data and open code) and wrap up	10:00-13:00	Li Ka Shing LK208

M218A: https://25live.collegenet.com/pro/stanfordsom#!/home/location/1454/details
LK208: https://25live.collegenet.com/pro/stanfordsom#!/home/location/1149/details

Office hours

Date	Instructor name	Time	Location
02.**29**.2024 – Thursday	Aziz	10:00-11:00	BMI 4022
03.**05**.2024 – Tuesday	Aziz	13:00-14:00	BMI 4022
03.**07**.2024 – Thursday	Aziz	10:00-11:00	BMI 4022

Git Recap:


git
Local
Remote
GitHub
GitLab
Bitbucket
git init
working directory
staging area
local repository
remote repository
.git/
$ git add
$ git commit
$ git push
$ git pull
$ git checkout
$ git merge
orcid.org/0000-0002-6459-6224

Best practices to use Git

- **Commit frequently**
  - Make small, frequent commits to capture incremental changes and provide a clear history of your work.
- **Write descriptive commit messages**
  - Write clear and descriptive commit messages that explain the purpose of each change.
- **Use branches**
  - Use branches to work on different features, bug fixes. Avoid making changes directly on the main branch.
- ***git pull* before *git push***
  - Always pull changes from the remote repository before pushing your own changes to avoid conflicts.
- **Review changes before committing**
  - Review your changes before committing them to ensure code quality and consistency
- **Use .gitignore**
  - Use a .gitignore file to specify files or directories that should be ignored by Git
- **Avoid committing sensitive info**
  - Avoid committing sensitive information such as passwords, PHI, keys, or credentials. Add to .gitignore
- **Use tags for releases**
  - Use Git tags to mark important releases or milestones in your project.
- **Use remote repos to backup**
  - Host your repositories on a remote Git server (e.g., GitHub, GitLab) to facilitate collaboration and backup.
- **Collaborate via pull requests**
  - Use pull requests for code review and collaboration, especially when working with others.
- **Resolve conflicts carefully**
  - Handle merge conflicts carefully by reviewing and resolving them before merging changes.
- **Clean your repos often**
  - Regularly clean up your repository by removing unused branches, tags, and stale code.

Learning goals for today

- Explain the purpose and fundamental concepts of package dependency management
- Apply Conda commands to create, activate, manage and share environments with collaborators/papers
- Utilize Conda channels (e.g. Bioconda) to install and manage bioinformatics software packages
- Analyze dependency conflicts within Conda environments and propose resolution strategies
- Use Mamba for faster software installations and resolve conflicts
- Learn how conda recipes are build and shared

Definitions

Environment: An isolated directory containing a specific collection of softwares needed for a project

Package: Pre-built/configured software bundles that can be easily installed into Conda environments

Channels: Repositories **hosting Conda packages**; Conda searches channels for package installation by default

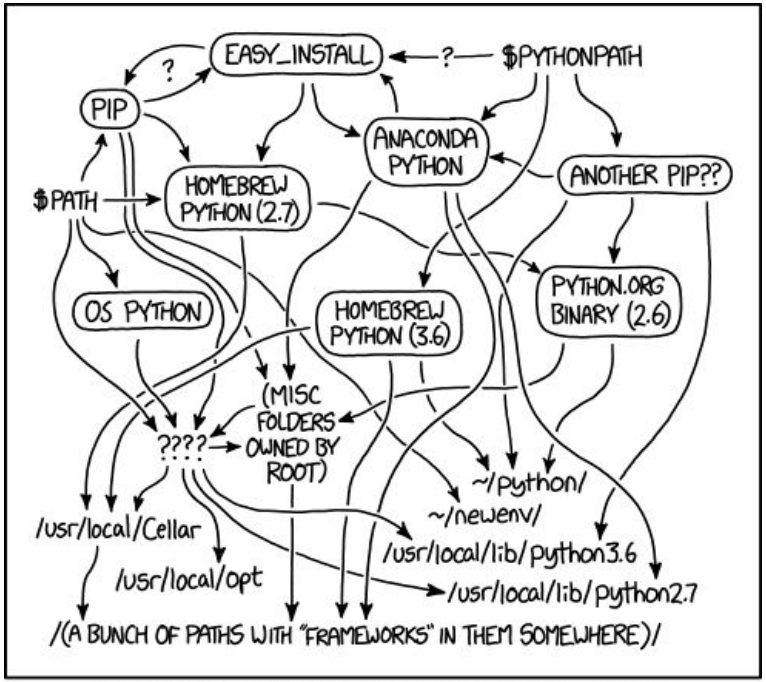
Dependency: A software package or library required by another package for proper functioning; Conda manages dependencies automatically

<env>.yaml: YAML is a human-readable data serialization language, commonly used for configuration files such as Conda environment – package dependencies and versions

Conda Recipe: Instructions in *meta.yaml* format on how to build a Conda package, including package name, version, dependencies, build scripts, and metadata

The dependency hell

- Our code often depends on others code that in turn depends on other code(s) ...
- We can control our code but how can we control dependencies?
- Different codes on the same environment can have conflicting dependencies



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

From xkcd.com

Tools for package dependency and environment management

What these tools do?

- **Install** a specific set of **dependencies**, possibly with well defined versions
- **Record** the **versions** for all dependencies
- **Isolate environments** on your computer **for projects** that have conflicting dependencies
- Isolate **environments** on computers **with many users**
- Use **different Python/R versions** per project
- Provide tools and services to share environments

Load environment modules with versions

A not reproducible way:

```
> module load samtools
```

A reproducible way:

```
> module load samtools/1.15
```

Find the latest version:

```
> module avail samtools
```

Install python modules using pip/requirements.txt

A not reproducible way:

```
> pip install intervene
```

A reproducible way:

```
> pip install intervene==0.6.5
```

Find the latest version:
https://pypi.org/project/intervene

Use requirements.txt:

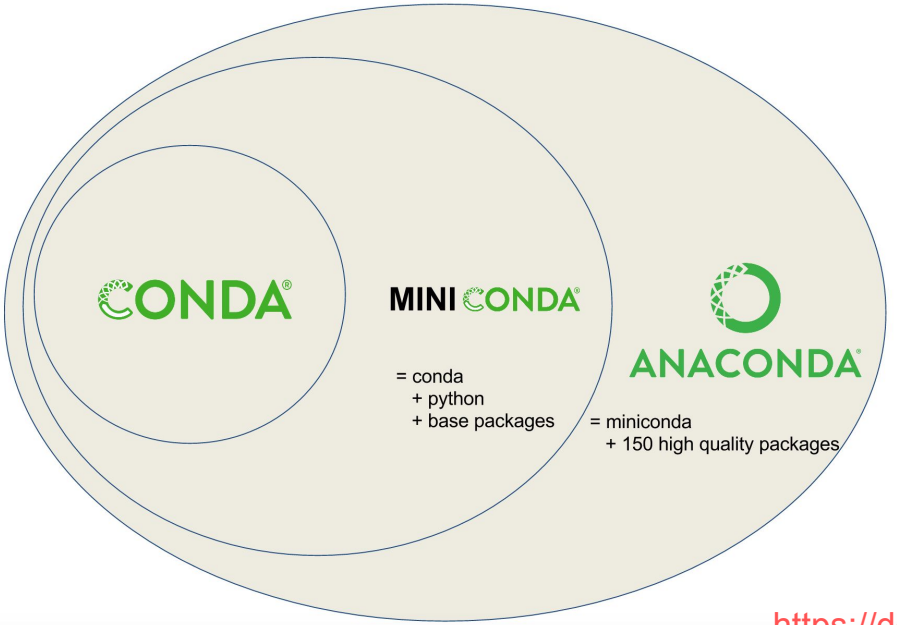
```
> pip install -r requirements.txt
```

What is Conda?

Conda is an **open-source** package and **environment management system** by Anaconda.

- Quickly install, run, and update packages and associated dependencies
- Create, save, load, and switch between project specific environments on your local computer
- Initially it was made for *Python* packages
    - Conda can package and distribute software for any language
    - R, Ruby, Java, JavaScript, C/C++, ...

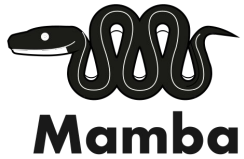
Conda vs. Miniconda vs. Anaconda



Source: Planemo documentation

https://docs.conda.io/projects/conda/
https://docs.conda.io/projects/miniconda

Mamba is more efficient than Conda



- **mamba** is a **reimplementation** of the *conda* in **C++** for **efficiency**
- **parallel downloading** of packages/data using multi-threading
- *libsolv* for **much faster dependency solving**, a state of the art library used in the RPM package manager of Red Hat, Fedora and OpenSUSE
- mamba utilizes the **same command line parser**, as conda to stay compatible

https://github.com/mamba-org/mamba
https://mamba.readthedocs.io/

Mamba vs. Conda

Feature	Mamba	Conda
Performance	Optimized for speed	Generally slower
Dependency solver	Uses *libsolv* a fast dependency solver	Uses its own dependency solver
Compatibility	Compatible with Conda	Original package manager, widely used
Language	Written in C++ for performance	Primarily written in Python
Installation	Requires separate installation	Included with Anaconda/Miniconda
Community Adoption	Gaining popularity	Established and widely used
Development	Actively developed with focus on speed	Receives updates and improvements

By design Mamba = Conda

Good news!! If learn/know conda, so you also know mamba!

Install Samtools using Conda:1.19-0

```
$ conda install -c bioconda samtools=1.19-0
```

Install Samtools using Mamba:

```
$ mamba install -c bioconda samtools=1.19-0
```

Install Mamba (Not recommended). Check installation guide

$ conda install -n base --override-channels -c conda-forge mamba 'python_abi==*cp*'*

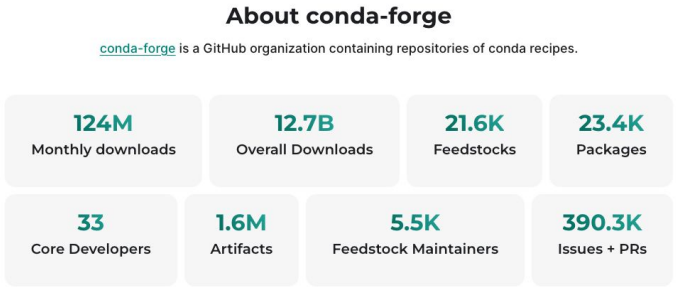
Conda channels

A channel is a location (a URL) where conda will look for packages. For example anaconda, r

Community led channels:

- *conda-forge*
- *bioconda*
- *defaults* - by Anaconda

You can create public/private channels



Bioconda channel

Bioconda is a community-lead, sustainable and comprehensive software distribution for the life sciences using conda

- Over 1400 contributors who add, modify, update, and maintain recipes and packages
- Over 8000 bioinformatics packages

Correspondence | Published: 02 July 2018

Bioconda: sustainable and comprehensive software distribution for the life sciences

Björn Grüning, Ryan Dale, Andreas Sjödin, Brad A. Chapman, Jillian Rowe, Christopher H. Tomkins-Tinch, Renan Valieris, Johannes Köster & The Bioconda Team

Nature Methods **15**, 475–476 (2018) | Cite this article

13k Accesses | **465** Citations | **231** Altmetric | Metrics

https://bioconda.github.io/
https://github.com/bioconda/bioconda-recipes

Use Bioconda – example

A not reproducible way:

```
> conda install -c bioconda pyjaspar
```

A reproducible way:

```
> conda install -c bioconda pyjaspar=2.1.0
```

Conda essential commands

$ conda <command> --help

--help is your best friend for most of the CLI tools

$ conda create: Create a new environment

$ conda activate: Load an environment

$ conda deactivate: Unload an environment

$ conda install: Install packages into environment

$ conda info: Show info about current env

$ conda list: List installed packages

$ conda remove: Remove a list of packages

$ conda update: Update packages to latest version(s)

$ conda rename: Rename existing env

$ conda search: Search for conda packages

Add channels and set priority

$ conda config --add channels <name>

(base) $ conda config --show channels
channels:
  - conda-forge
  - bioconda
  - defaults

Run these commands to modify your ~/.condarc file. Order is important from lowest to highest priority to avoid problems with solving dependencies:

$ conda config --add channels defaults

$ conda config --add channels bioconda

$ conda config --add channels conda-forge

$ conda config --set channel_priority strict

Conda loads base env by default

(base) $

$ conda config --set auto_activate_base false

Create an env

$ conda create --name <env name>

$ conda create --name bios259 -c conda-forge python=3.9

additional arguments

-y to bypass the prompt

--prefix to provide a new location

Activate an env

$ conda activate <env name>

$ conda activate bios259

(bios259)$ python --version

> Python 3.9.18

 # Deactivate environment

$ conda deactivate

$

Install/Unistall packages

$ conda install <package_name>

$ conda uninstall <package_name>

$ conda install pandas=2.1.4 -c conda-forge

(bios259)$ python

> import pandas as pd

> exit()

(bios259) $

Try to uninstall pandas and install matplotlib latest version?

(bios259) $ conda uninstall < >

(bios259) $ conda search < >

View your conda environments

 # View a list of all your environments

$ conda env list

$ conda info --envs #OR

 # View a list of the packages in an environment

$ conda list -n myenv

(myenv) $ conda list

 # Check a package is already installed

(myenv) $ conda list snakemake

Practice exercise

What are the most recent versions of pandas and seaborn?

```
$ conda search pandas | tail -n1

pandas       2.2.0          conda-forge
```

```
$ conda search pandas | tail -n1

snakemake       8.4.8          bioconda
```

Answer recorded [02.15.2024]

Discussion exercise

Situation: A student group uploaded their project code to GitHub 3 years ago and you are trying to re-run their code before adapting it.

Discussion prompts:

- Which option do you expect to be easiest to re-run? Why?
- Which option is more reproducible practice?
- What problems do you anticipate in each solution?

A: You find a couple of library imports across the code but that's it.

B: The README file lists which libraries were used but does not mention any versions.

C: You find a environment.yaml file with:

```
name: awesome-project
channels:
  - conda-forge
dependencies:
  - scipy
  - numpy
  - sympy
  - click
  - python
```

D: You find a environment.yaml file with:

```
name: awesome-project
channels:
  - conda-forge
dependencies:
  - scipy=1.3.1
  - numpy=1.16.4
  - sympy=1.4
  - click=7.0
  - python=3.8
```

Hands on exercise 1

~30 minutes

Pair in two and create a conda environment and install packages

- Create a conda env with name ***myenv*** and python v3.9
- Activate env and Install the following packages
  - seaborn
  - notebook
  - nextflow
  - snakemake
- What versions of these packages get installed?
  - numpy, pandas, nextflow, snakemake
- Find the location of your environment
- List all the packages installed
- Rename ***myenv*** to **bios259**
- Now view the conda info

Export and share environments

Creating an environment file manually

```
name: bios259
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.9
  - seaborn=0.13.2
  - notebook=0.7.11
  - <>=xx
  - <>=xx
```

Export to yaml

$ conda env export > environment.yaml

Import/create using yaml

$ conda env create -f environment.yaml

Think-pair share exercise

Take 2-3 minutes to think and write about:

Why creating separate Conda environments for each project is important?

Some discussion prompts:

- What are some challenges you've faced when managing dependencies in your projects?
- How has using separate Conda environments benefited your project?
- Can you think of any scenarios where sharing a Conda environment between projects might be advantageous or disadvantageous?

Hands on exercise 2

~60 minutes

Pair in two and export, share and import conda envs

- Export your **bios259** env into environment.yaml
- Share your environment.yaml with the pair through Github using one repo
- Import the environment.yaml to create new env with name ***bios259_colab***
- Find the location of new environment
- List all the packages installed
- What versions of these packages get installed?
    - numpy, pandas
- Delete the env ***bios259_colab***

Conda best practices for reproducibility

- Keep your base env clean! (only the package manager + its deps)
- Create **different environments** for different projects
- Create a general environment for general tasks and code testing
- Avoid creating environments for individual tools unless, you can escape the dependency hell
- Freeze and share the environment.yaml via GitHub with your publication
    - List dependencies and their versions. This enables others to recreate the environment easily and reproduce your results

slido

What should you include in your project repository to facilitate reproducibility with Conda?

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

What should you include in your project repository to facilitate reproducibility with Conda?

A. A list of all your favorite packages

B. A screenshot of your terminal session

C. A *requirements.txt* file

D. An *environment.yml* file containing environment specifications/versions

Building conda packages

What is a Conda Recipe?

A set of instructions for building a Conda package from source

Components of a Conda recipe:

- meta.yaml
- build scripts, and package files

Importance of creating accurate and well-documented Conda recipes for reproducibility and compatibility.

Recipe example - cutadapt

meta.yaml sections

package: name and version

source: url and SHA256/MD5 checksums

build: build number, platforms to skip, “noarch” information

requirements: packages for building, linking, running

test: commands/imports

about: webpage, license, summary of what the package does

extras: comments, maintainers, etc.

bioconda-recipes / recipes / cutadapt / meta.yaml

bernt-matthias cutadapt: require dnaio 1.2 (#44763)

Code Blame 47 lines (40 loc) · 956 Bytes

```
{% set version = "4.6" %}

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

source:
  url: https://files.pythonhosted.org/packages/35/b4/f1a7401c3503c17998fb9547b04353217a18323d5d85a3b957f1049ab800/cutadapt-4.6.tar.gz
  sha256: 924116f34569248035b16f58e73458ed4c0004e44823b80b07f4ab419272f591

build:
  number: 1
  script: "{{ PYTHON }} -m pip install . --no-deps -vv"
  skip: True  # [py27 or py36]
  run_exports:
    - {{ pin_subpackage("cutadapt", max_pin="x") }}

requirements:
  build:
    - {{ compiler('c') }}
  host:
    - pip
    - python
    - cython
    - setuptools-scm
  run:
    - python
    - xopen >=1.6.0
    - dnaio >=1.2.0

test:
  imports:
    - cutadapt
  commands:
    - cutadapt --version

about:
  home: https://cutadapt.readthedocs.io/
  license: MIT
  summary: Trim adapters from high-throughput sequencing reads

extra:
  recipe-maintainers:
    - marcelm
  identifiers:
    - biotools:cutadapt
    - doi:10.14806/ej.17.1.200
```

Install conda-build

$ conda-build --help

$ conda activate base

$ conda install conda-build

Building conda packages with conda skeleton

Create recipe for python package Click:

#Use skeleton
$ conda skeleton pypi click

#Builid the click recipe
$ conda-build click

# Install and test locally
$ conda install --use-local click

Post to a channel (Bioconda)

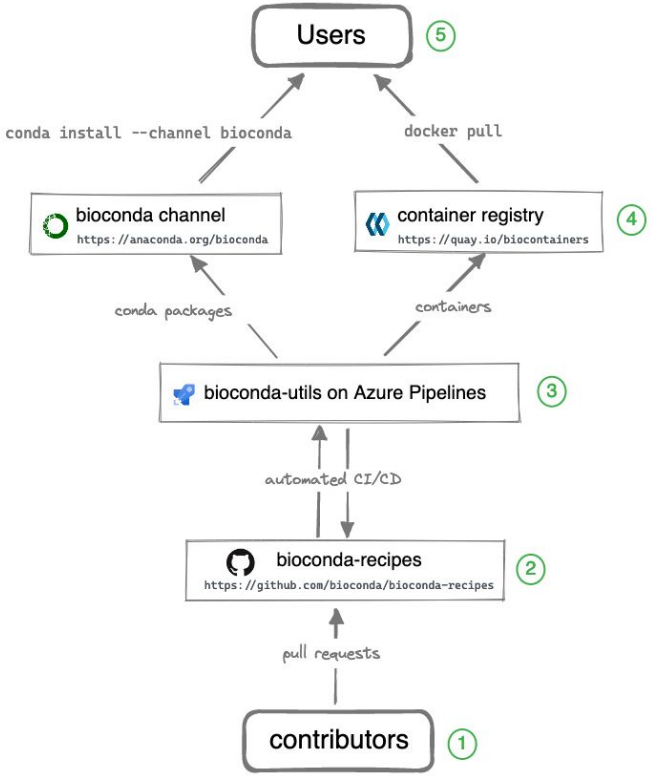
```
[redacted] conda skeleton --help
usage: conda skeleton [-h] {cpan,cran,luarocks,pypi,rpm} ...

Generates a boilerplate/skeleton recipe, which you can then edit to create a
full recipe. Some simple skeleton recipes may not even need edits.


positional arguments:
  {cpan,cran,luarocks,pypi,rpm}
    cpan                Create recipe skeleton for packages hosted on the
                        Comprehensive Perl Archive Network (CPAN) (cpan.org).
    cran                Create recipe skeleton for packages hosted on the
                        Comprehensive R Archive Network (CRAN)
                        (cran.r-project.org).
    luarocks            Create recipe skeleton for luarocks, hosted at
                        luarocks.org
    pypi                Create recipe skeleton for packages hosted on the
                        Python Packaging Index (PyPI) (pypi.io).
    rpm                 Create recipe skeleton for RPM files
```

https://docs.conda.io/projects/conda-build/en/latest/user-guide/tutorials/build-pkgs-skeleton.html

Building Bioconda recipes



① Over 1400 contributors who add, modify, update, and maintain recipes and packages

▸ *Details*

② A repository of recipes hosted on GitHub

▸ *Details*

③ A build system that turns each recipes into a conda package and a Docker container

▸ *Details*

④ A repository of packages and a registry of containers containing over 8000 bioinformatics packages

▸ *Details*

⑤ Users can then use the package with **`conda install`** or **`docker pull`**

▸ *Details*

https://github.com/bioconda/bioconda-recipes

Bioconda recipe build workflow

1. Fork the Bioconda repository on GitHub
    a. https://github.com/bioconda/bioconda-recipes
2. Create new branch with recipe name
3. Add the Conda recipe to the appropriate subdirectory in the recipes directory
4. Follow the Bioconda contribution guidelines and provide necessary documentation and metadata
5. Build and test the recipe
6. Create a pull request (PR) with the new recipe
7. Await review and feedback from the Bioconda maintainers
8. Address any requested changes and collaborate with maintainers to finalize the PR

Tips for successful contributions

- Follow the Bioconda contribution guidelines and formatting standards
- Provide clear and concise documentation for the package
- Specify version for all the dependencies
- Test the recipe thoroughly to ensure it builds and installs correctly
- Engage with the Bioconda community and collaborate with maintainers for feedback and assistance

Resources

Bioconda GitHub repository: https://github.com/bioconda/bioconda-recipes

Bioconda contribution guidelines: https://bioconda.github.io/contributor/index.html

Conda documentation: https://docs.conda.io/projects/conda-build/en/stable/

Conda cheat sheet: https://docs.conda.io/projects/conda/en/latest/_downloads/843d9e0198f2a193a3484886fa28163c/conda-cheatsheet.pdf