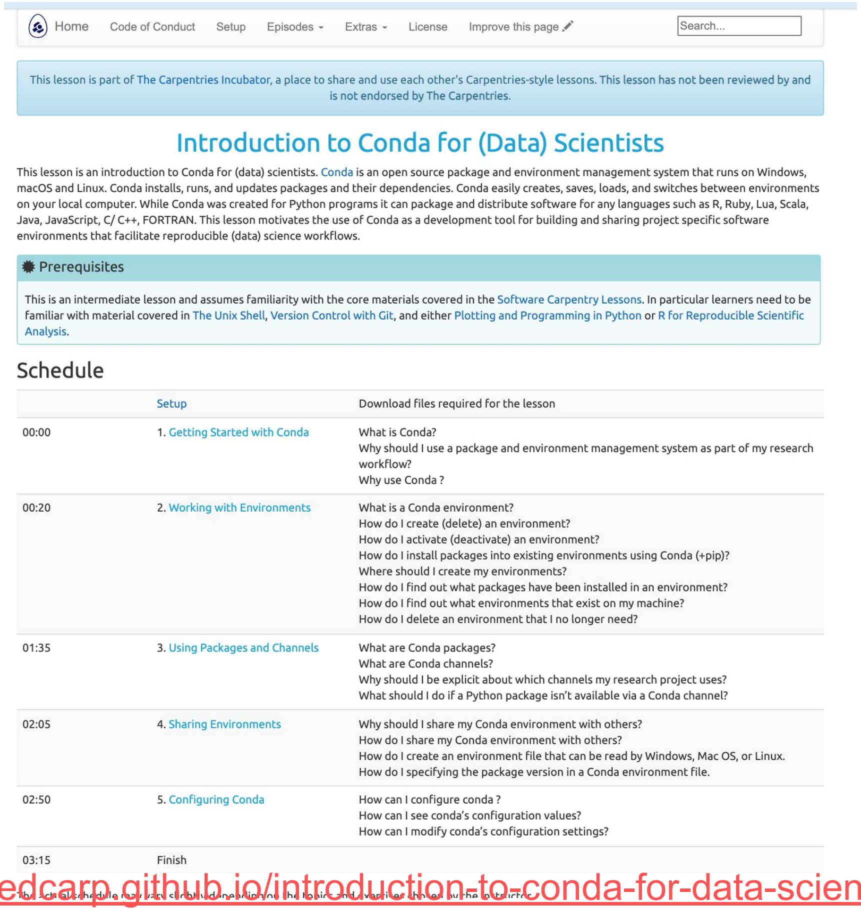
Introduction to Conda for Biologists

23rd November 2021

Introduction to Conda for Biologists



This lesson is part of The Carpentries Incubator, a place to share and use each other's Carpentries-style lessons. This lesson has not been reviewed by and is not endorsed by The Carpentries.

Introduction to Conda for (Data) Scientists

This lesson is an introduction to Conda for (data) scientists. Conda is an open source package and environment management system that runs on Windows, macOS and Linux. Conda installs, runs, and updates packages and their dependencies. Conda easily creates, saves, loads, and switches between environments on your local computer. While Conda was created for Python programs it can package and distribute software for any languages such as R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN. This lesson motivates the use of Conda as a development tool for building and sharing project specific software environments that facilitate reproducible (data) science workflows.

Prerequisites

This is an intermediate lesson and assumes familiarity with the core materials covered in the Software Carpentry Lessons. In particular learners need to be familiar with material covered in The Unix Shell, Version Control with Git, and either Plotting and Programming in Python or R for Reproducible Scientific Analysis.

Schedule

	Setup	Download files required for the lesson
00:00	1. Getting Started with Conda	What is Conda? Why should I use a package and environment management system as part of my research workflow? Why use Conda ?
00:20	2. Working with Environments	What is a Conda environment? How do I create (delete) an environment? How do I activate (deactivate) an environment? How do I install packages into existing environments using Conda (+pip)? Where should I create my environments? How do I find out what packages have been installed in an environment? How do I find out what environments that exist on my machine? How do I delete an environment that I no longer need?
01:35	3. Using Packages and Channels	What are Conda packages? What are Conda channels? Why should I be explicit about which channels my research project uses? What should I do if a Python package isn't available via a Conda channel?
02:05	4. Sharing Environments	Why should I share my Conda environment with others? How do I share my Conda environment with others? How do I create an environment file that can be read by Windows, Mac OS, or Linux. How do I specifying the package version in a Conda environment file.
02:50	5. Configuring Conda	How can I configure conda ? How can I see conda's configuration values? How can I modify conda's configuration settings?
03:15	Finish	

The actual schedule may vary slightly depending on the topics and exercises chosen by the instructor.

https://edcarp.github.io/introduction-to-conda-for-data-scientists

Introduction to Conda for Biologists

```
# Working with Environments

## Creating environments
conda create --name python3-env python
conda create --name python36-env python=3.6
conda search scikit-learn
conda create --name basic-scipy-env ipython=7.13 matplotlib=3.1 numpy=1.18 scipy=

## Exercise: Creating a new environment
conda create --name machine-learning-env \
 ipython=7.19 \
 matplotlib=3.3 \
 pandas=1.2 \
 python=3.8 \
 scikit-learn=0.23 \
 numba=0.51

## Activating an existing environment
conda activate basic-scipy-env

##  Deactivate the active environment
conda deactivate

## Exercise: Activate an existing environment by name
conda activate machine-learning-env
conda deactivate

## Installing a package into an existing environment
conda activate basic-scipy-env
conda install numba


conda install scikit-learn=0.22

## Remove a package from an environment
conda uninstall scikit-learn -n basic-scipy-env

## Exercise: Installing a package into a specific environment
conda install --name machine-learning-env dask=2020.12


## Where do Conda environments live?
conda config --show envs_dirs

## How do I specify a location for a Conda environment?
conda create --prefix ./env ipython=7.13 matplotlib=3.1 pandas=1.0 python=3.6
conda activate ./env
```

https://raw.githubusercontent.com/edcarp/introduction-to-conda-for-data-scientists/gh-pages/files/commands.txt

Introduction to Conda for Biologists: Timetable

Episode	Time	Website
Getting Started with Conda	9:30-10:00	https://edcarp.github.io/introduction-to-conda-for-data-scientists/01-getting-started-with-conda
Working with Environments	10:00-10:45	https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments
Break	10:45-11:00	
Using Packages and Channels	11:00-12:00	https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html
Break	12:00-13:00	
Sharing Environments	13:00-13:45	https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments
Configuring Conda	13:45-12:30	https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration/index.html

Introduction to Conda for Biologists

Getting Started with Conda

https://edcarp.github.io/introduction-to-conda-for-data-scientists/01-getting-started-with-conda

Getting Started with Conda: Questions

- What is conda?
- Why use conda?

What is Conda?

Conda is an open source package management system and environment management system

https://docs.conda.io/en/latest/

Getting Started with Conda: Questions

- Why should I use a package and environment management system as part of my research workflow?

Packages

What are Packages?

Python

What is Python used for?

- AI and machine learning. ...
- Bioinformatics analysis ..
- Data visualisation. ...
- Programming applications. ...
- Web development. ...
- Game development. ...
- Finance.

What are Packages?

Python

Package

Extend functionality

- Package: a collection of modules in a directory
    a. Module: a collection of functions and variables, as in a script

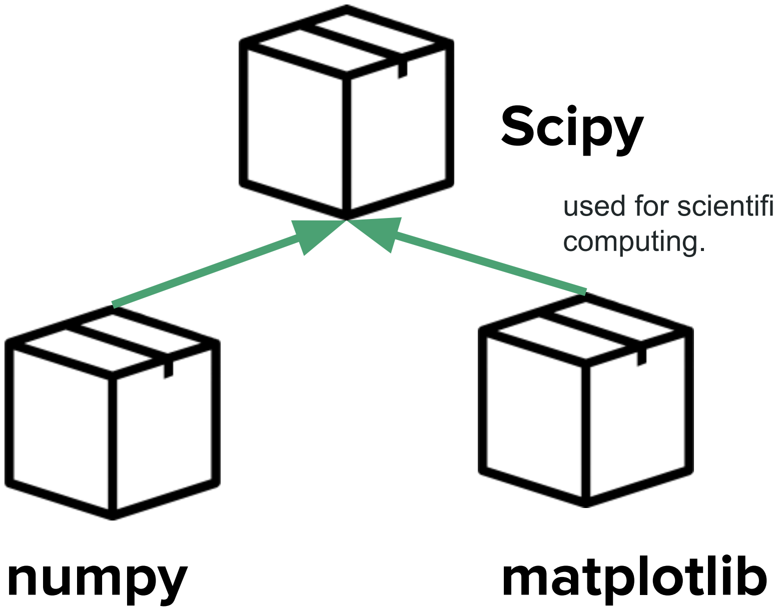
What are Packages?

Python

Scipy

used for scientific computing and technical computing.

Dependencies

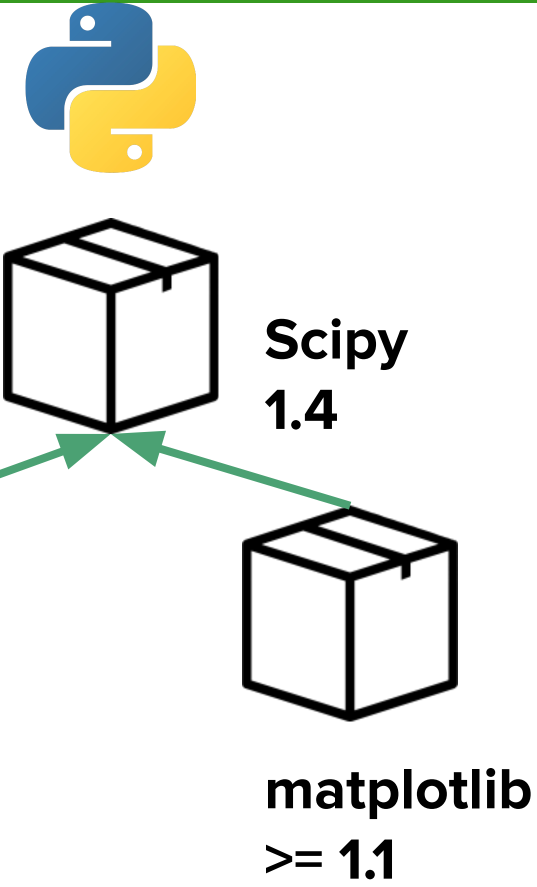


used for scientific computing and technical computing.

(numerical python)

(plotting)

Package Dependencies



Environment

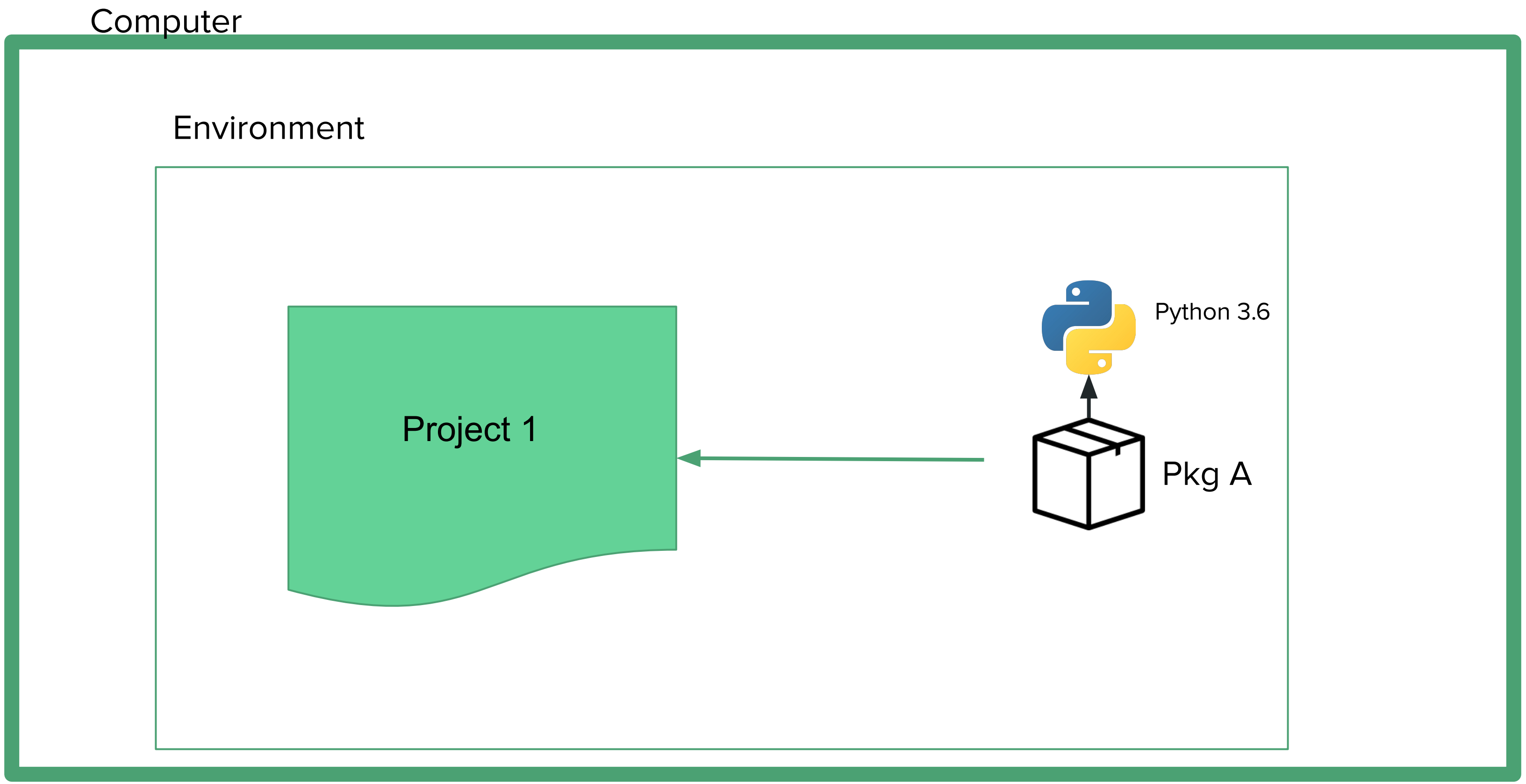
Environment

Computer

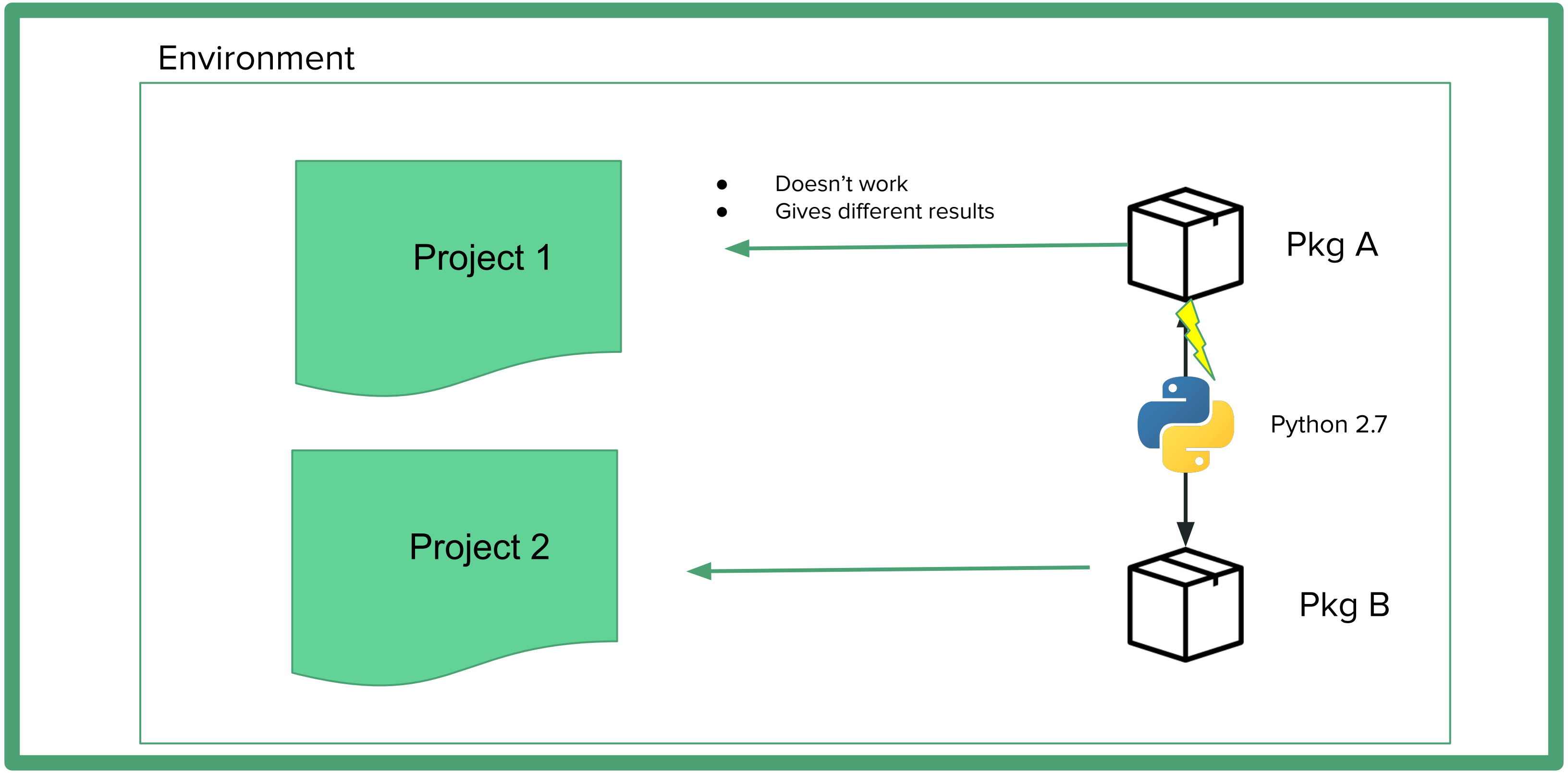
Environment

- All the Software
  - R, Python, C/C++, Java, sequence aligners ...
- Packages/Libraries
  - DESeq2, pysam, htslib
- Environment variables
  - PATH , RSTUDIO_WHICH_R

Environment

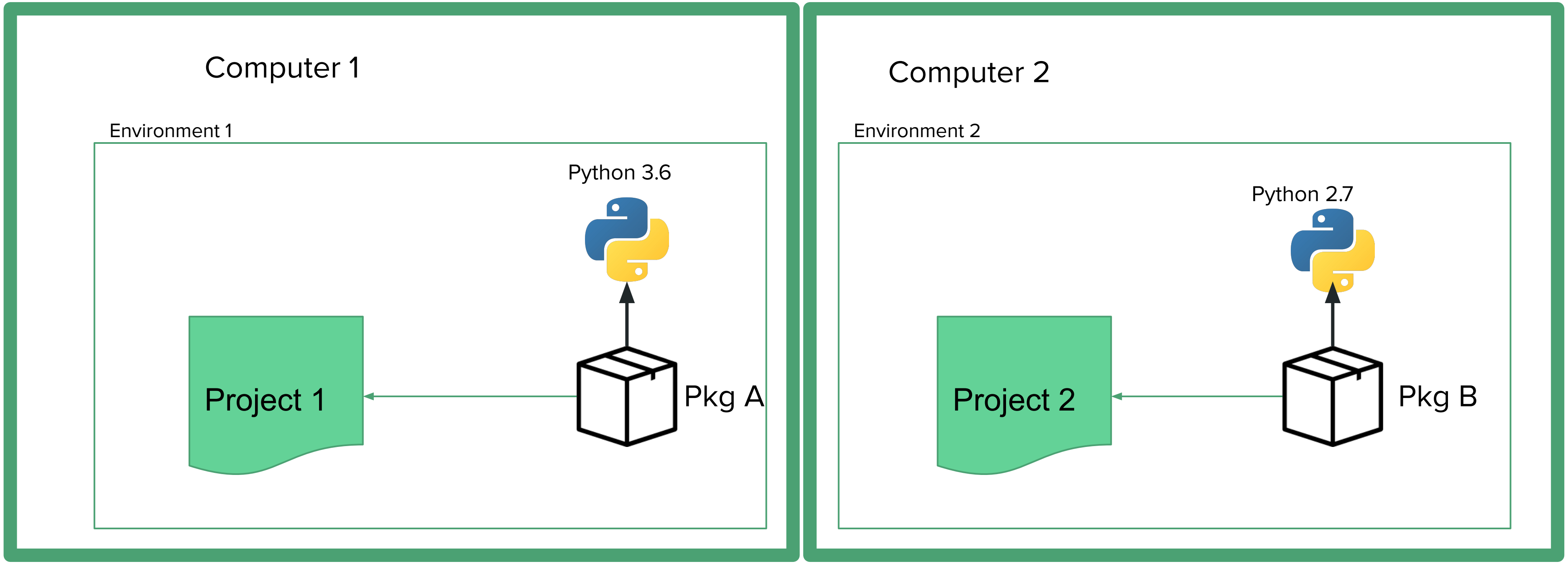


Environment: Update issues

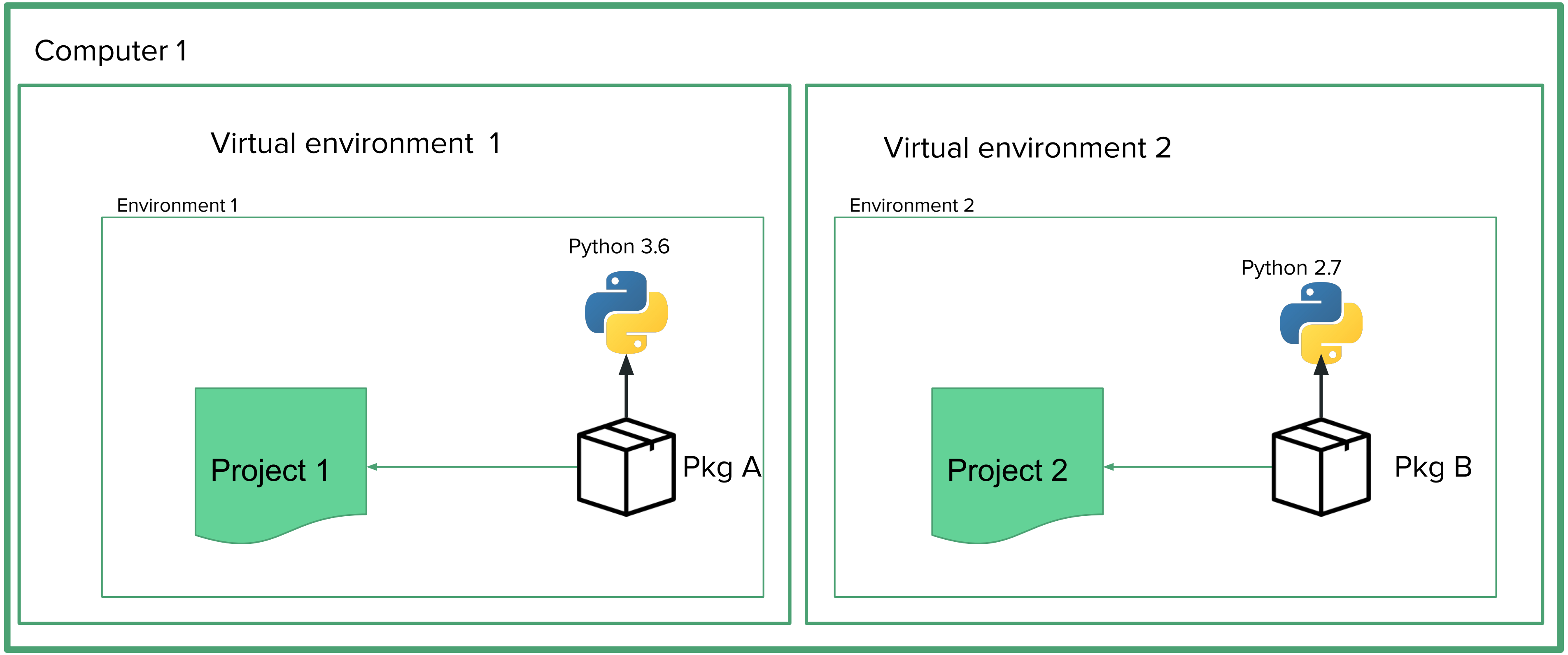


- Doesn't work
- Gives different results

Solution: Separate Computers /VMs



Solution: Virtual Environments



Environment Managers

Environment Management System

- Allow you to use different versions of a package for different projects.
- Make your projects self-contained and reproducible.
    - Capture packages and dependencies in a single file
- Can allow you to install packages on a host on which you do not have admin privilege

Solution: Environments Management

Language specific		Any Language
Python	R	Linux
venv, pipenv	renv,packrat	modules

Package management

Why should I use a Package Management System?

- Installing software can be hard.
  - Dependency hell

- **Installing bioinformatic software can take weeks!**

Good Package Management tools

- Identifies and installing compatible versions of packages (software and all required dependencies).

- Handling the process of updating packages as more recent versions become available.

Package Management Tools

Operating System		
Linux	osx	Windows
apt, yum..	homebrew	Chocolately

Packagement tools limitations

Installing software system wide has issues;

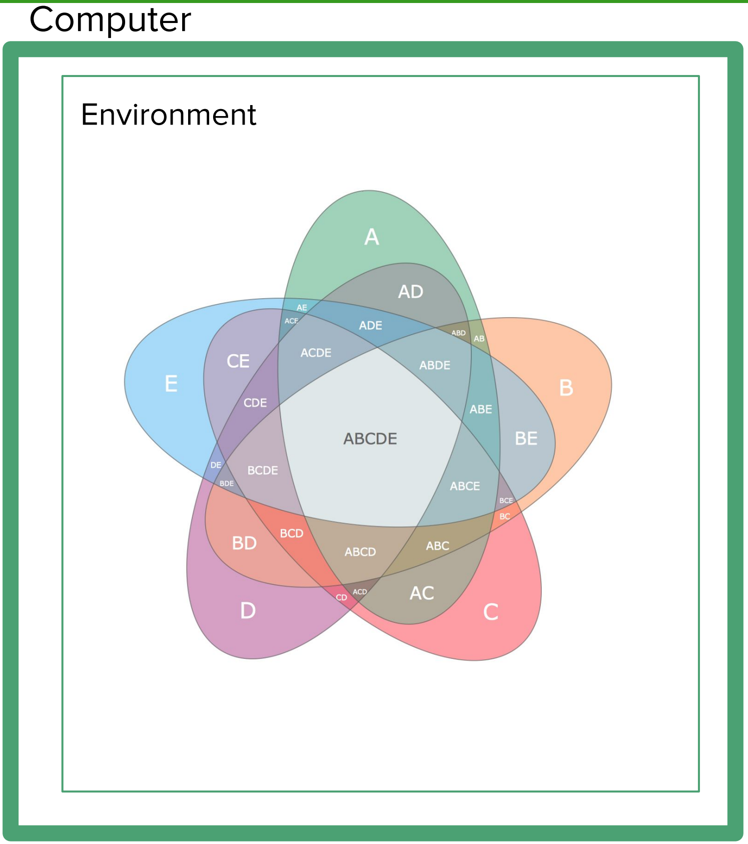
- It can be difficult to figure out what software is required for any particular research project.
- It is often impossible to install different versions of the same software package at the same time.
- Updating software required for one project can often "break" the software installed for another project.

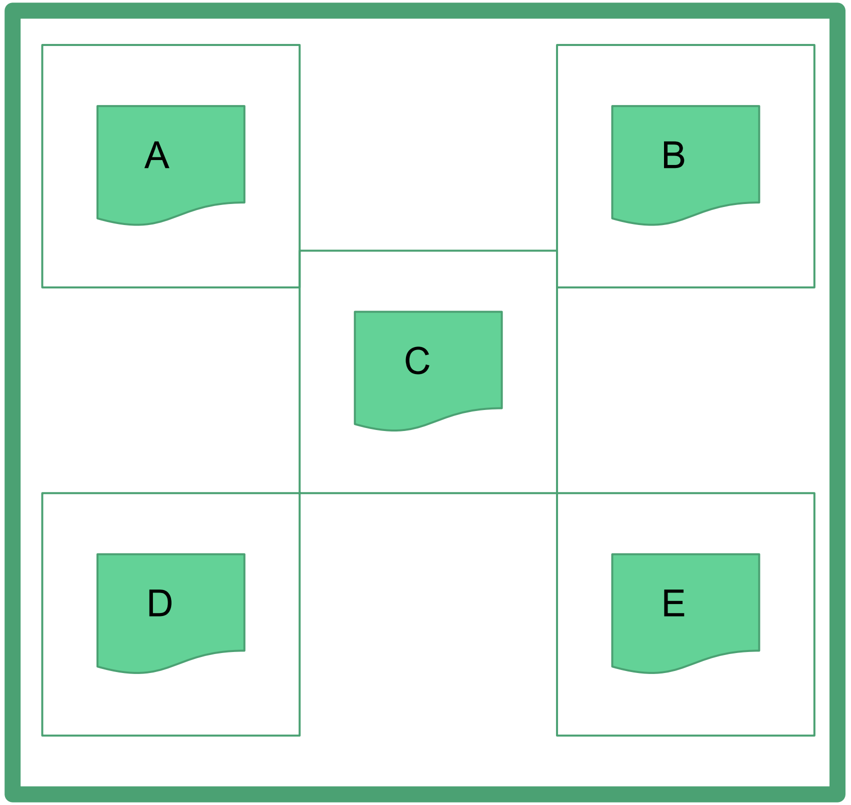
Packagement tools limitations

- As researcher you typically multiple scripting languages
  - R, Python, SQL

- A project may require a different version of the scripting language than the one installed on your system.
  - Python v2 or v3

Project dependencies





Discussion

- In the etherpad write a potential benefits from installing software separately for each project?

- And write a potential costs?

Conda: Package & Environment Manager

Conda: Package Manager

- Package Manager:
  - find,
  - install,
  - run,
  - update packages and their dependencies.

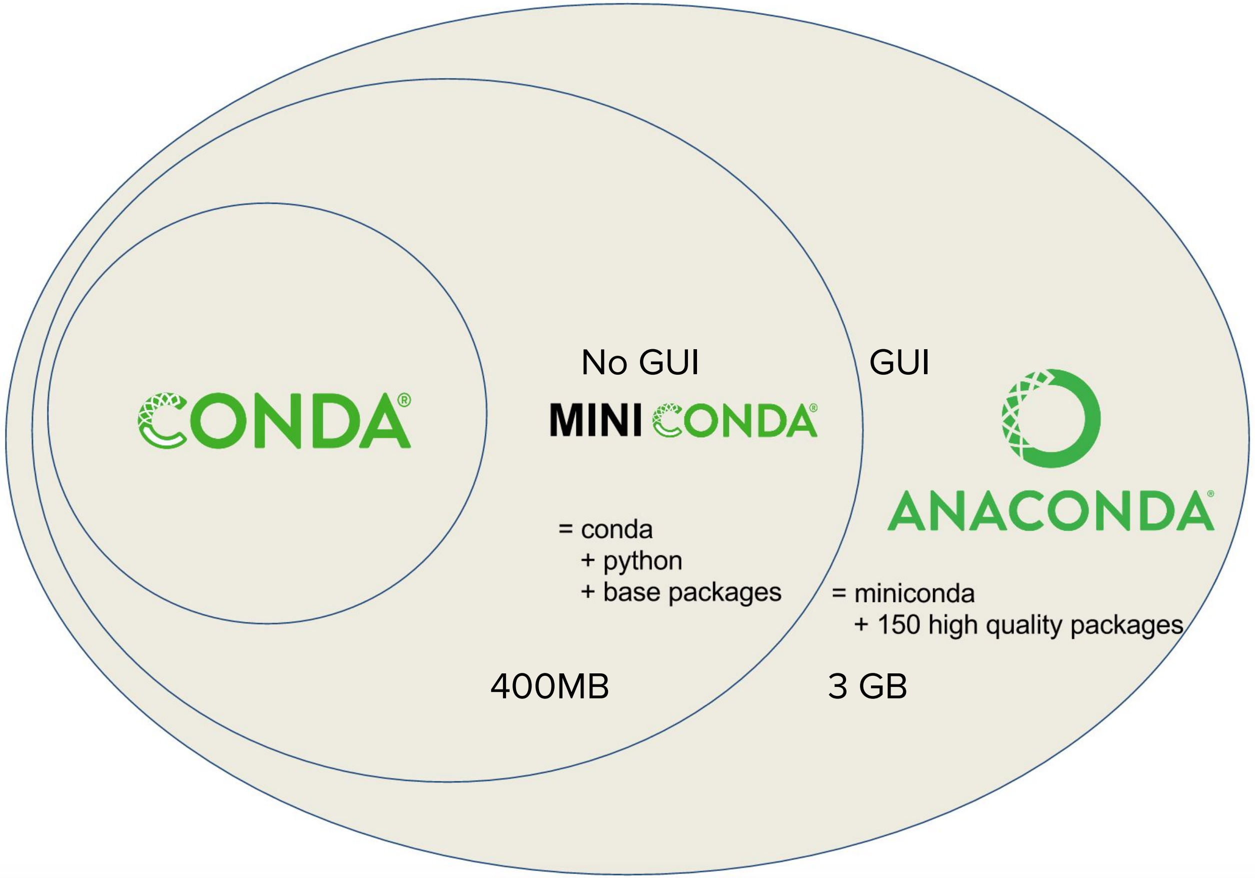
Conda: Environment Manager

- Environment Manager:
    - create,
    - save,
    - load,
    - switch between project specific software environments on your local computer or HPC.

Conda: Package any software

- Although Conda was created for Python programs, Conda can package and distribute software for any language such as
  - R,
  - Ruby,
  - Lua,
  - Scala,
  - Java,
  - JavaScript,

Conda, Anaconda & Miniconda



Why use Conda?

- Conda solves both a **package** and **environment manager** and explicitly targeted at science use cases.
- Conda is an open source and free
- Conda is cross platform
- Conda can access dedicated repositories (channels) with thousands of prebuilt packages packages (no need to compile software).
    - Including a dedicated Bioinformatics package repository.

Why use Conda?

- Unlike some package managers e.g. pip, conda provides prebuilt packages don’t need to be compiled.

- Conda makes sure all packages dependencies installed in an environment are met **simultaneously** before installing them.

Conda: Reproducibility

- Conda environments support reproducibility via a single environment files that describe their installation state.
- Conda is tightly integrated into popular solutions for reproducible data analysis such as
    - Snakemake
    - Nextflow
    - Galaxy,
    - bcbio-nextgen

Key Points

- Conda is a platform agnostic, open source package and environment management system.

- Using a package and environment management tool facilitates portability and reproducibility of data science workflows.

- Conda solves both the package and environment management problems and targets multiple programming languages.
  - Other open source tools solve either one or the other, or target only a particular programming language.

Working with Environments

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments

Working with Environments: Questions

- What is a conda environment?
- How do I create (delete) an environment?
- How do I activate (deactivate) an environment?
- Where should I create my environments?
- How do I find out what packages have been installed in an environment?
- How do I find out what environments that exist on my machine?
- How do I delete an environment that I no longer need?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: What is a Conda environment ?

A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.

For example, you may be working on a research project that requires `NumPy 1.18` and its dependencies, while another environment associated with an finished project has `NumPy 1.12` (perhaps because version 1.12 was the most current version of NumPy at the time the project finished).

If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them.

Working with Environments: Creating environments

To create a new environment for Python development using conda you can use the conda create command.

```
$ conda create
```

Working with Environments: Creating environments

To create a new environment for Python development using conda you can use the conda create command.

Add an environment name using --name option

```
$ conda create --name python3-env
```

It is a good idea to give your environment a meaningful name in order to help yourself remember the purpose of the environment. While naming things can be difficult, `$PROJECT-NAME-env` is a good convention to follow. Sometimes also the specific version of a package why you had to create a new environment is a good name

Working with Environments: Creating environments

To create a new environment for Python development using conda you can use the conda create command.

Add a package name

```
$ conda create --name python3-env python
```

Working with Environments: Specify version

The previous command above will create a new Conda environment called "python3-env" and install the most recent version of Python.

If you wish, you can specify a particular version of packages for conda to install when creating the environment.

```
$ conda create --name python36-env python=3.6
```

Working with Environments: Multiple Packages

You can create a Conda environment and install multiple packages by listing the packages that you wish to install.

```
$ conda create --name basic-scipy-env
```

Working with Environments: Multiple Packages

You can create a Conda environment and install multiple packages by listing the packages that you wish to install.

```
$ conda create --name basic-scipy-env ipython=7.13
```

Working with Environments: Multiple Packages

You can create a Conda environment and install multiple packages by listing the packages that you wish to install.

```
$ conda create --name basic-scipy-env ipython=7.13 matplotlib=3.1
```

Working with Environments: Multiple Packages

You can create a Conda environment and install multiple packages by listing the packages that you wish to install.

```
$ conda create --name basic-scipy-env ipython=7.13 matplotlib=3.1 numpy=1.18
```

Working with Environments: Multiple Packages

You can create a Conda environment and install multiple packages by listing the packages that you wish to install.

```
$ conda create --name basic-scipy-env ipython=7.13 matplotlib=3.1 numpy=1.18 scipy=1.4
```

Working with Environments: Dependencies

When conda installs a package into an environment it also installs any required dependencies.

For example, even though Python is not listed as a packaged to install into the **`basic-scipy-env`** environment above, conda will still install Python into the environment because it is a required dependency of at least one of the listed packages.

Exercise: Creating a new environment

Create a new environment called `machine-learning-env` with Python and the most current versions of

- IPython
- Matplotlib
- Pandas
- Numba
- Scikit-Learn

Hint to install latest compatible version do not specify a version number

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Activating Envs

Activating environments is essential to making the software in environments work

Activation of an environment does two things.

Working with Environments: Activating Envs

Activating environments is essential to making the software in environments work

Activation of an environment does two things.

1. Adds entries to $PATH for the environment.

Working with Environments: Activating Envs

Activating environments is essential to making the software in environments work

Activation of an environment does two things.

1. Adds entries to $PATH for the environment.
2. Runs any activation scripts that the environment may contain.

Working with Environments: Activating Envs

Activating environments is essential to making the software in environments work

Activation of an environment does two things.

1. Adds entries to $PATH for the environment.
2. Runs any activation scripts that the environment may contain.

Step 2 is particularly important as activation scripts are how packages can set arbitrary environment variables that may be necessary for their operation.

Working with Environments: Activating Envs

You activate the basic-scipy-env environment by name using the activate command.

```
$ conda activate basic-scipy-env
```

Working with Environments: Activating Envs

You can see that an environment has been activated because the shell prompt will now include the name of the active environment.

```
(basic-scipy-env) $
```

Working with Environments: Deactivating Envs

To deactivate the currently active environment use the Conda deactivate command as follows.

```
(basic-scipy-env) $ conda deactivate
```

Working with Environments: Deactivating Envs

You can see that an environment has been deactivated because the shell prompt will no longer include the name of the previously active environment.

```
(base) $
```

Working with Environments: Deactivating Envs

To return to the base Conda environment, it’s better to call `conda activate` with no environment specified, rather than to use deactivate.

If you run conda deactivate from your base environment, you may lose the ability to run conda commands at all

```
$ conda activate basic-scipy-env
$ (basic-scipy-env) $ conda activate
```

Exercise: Activate an existing environment by name

1. Activate the `machine-learning-env` environment created in the previous challenge by name.

2. Deactivate the `machine-learning-env` environment that you activated in the previous challenge.

Working with Environments: Installing into Existing Env

You can install a package into an existing environment using the **`conda install`** command.

Working with Environments: Installing into Existing Env

You can install a package into an existing environment using the **`conda install`** command.

This command accepts a list of package specifications (i.e., **`numpy=1.18`**) and installs a set of packages consistent with those specifications and compatible with the underlying environment.

```
(my_active-env) $ conda install numpy=1.18
```

Working with Environments: Installing into Existing Env

You can install a package into an existing environment using the **`conda install`** command.

This command accepts a list of package specifications (i.e., **`numpy=1.18`**) and installs a set of packages consistent with those specifications and compatible with the underlying environment.

If full compatibility cannot be assured, an error is reported and the environment is not changed.

Working with Environments: Installing into Existing Env

By default the conda install command will install packages into the current, active environment.

The following would activate the **`basic-scipy-env`** we created above and install

```
$ conda activate basic-scipy-env
(basic-scipy-env)$ conda install numba
```

Working with Environments: Installing into Existing Env Ver Num

As was the case when listing packages to install when using the `conda create` command, if version numbers are not explicitly provided, Conda will attempt to install the newest versions of any requested packages.

Working with Environments: Installing into Existing Env Ver Num

As was the case when listing packages to install when using the `conda create` command, if version numbers are not explicitly provided, Conda will attempt to install the newest versions of any requested packages.

To accomplish this, Conda may need to update some packages that are already installed or install additional packages.

Working with Environments: Installing into Existing Env Ver Num

As was the case when listing packages to install when using the `conda create` command, if version numbers are not explicitly provided, Conda will attempt to install the newest versions of any requested packages.

To accomplish this, Conda may need to update some packages that are already installed or install additional packages.

It is always a good idea to explicitly provide version numbers when installing packages with the `conda install` command.

Working with Environments: Installing into Existing Env Ver Num

The following would install a particular version of **`scikit-learn`**, into the current, active environment.

```
(basic-scipy-env)$ conda install scikit-learn=0.22
```

Working with Environments: Remove a package from an environment

To remove a package from an environment you can run the command.

```
$ conda uninstall PKGNAME --name ENVNAME
```

Working with Environments: Remove a package from an environment

To remove the scikit-learn package from the basic-scipy-env environment run

```
$ (basic-scipy-env) conda uninstall scikit-learn -n basic-scipy-env
```

Exercise: Installing a package into a specific environment

Dask provides advanced parallelism for data science workflows enabling performance at scale for the core Python data science tools such as Numpy Pandas, and Scikit-Learn.

Run

```
$ conda install --help
```

And figure mout how to Install **`dask`** into the **`machine-learning-env`** that you created in the previous challenge.

Working with Environments: Where do Conda environments live?

Environments created with conda, by default, live in the **`envs/`** folder of your miniconda3 (or anaconda3) directory the absolute path to which will look something the following:

```
/home/training/miniconda3/envs
```

```
C:\Users\$USERNAME\Anaconda3
```

Working with Environments: Where do Conda environments live?

You can see the location of your conda environments by running the command.

```
$ conda config --show envs_dirs
```

Working with Environments: Where do Conda environments live?

Running **`ls`** (linux) / **`dir`** (Windows) on your anaconda envs/ directory will list out the directories containing the existing Conda environments.

```
$ ls /home/training.miniconda3/envs
```

Working with Environments: How do I specify a location for a Conda environment?

You can control where a Conda environment lives by providing a path to a target directory when creating the environment.

The following command will create a new environment in a sub-directory of the current working directory called **`env`**

```
$ pwd
$ conda create --prefix ./env
```

Working with Environments: How do I specify a location for a Conda environment?

You can control where a Conda environment lives by providing a path to a target directory when creating the environment.

For example the following command will create a new environment in a sub-directory of the current working directory called **`env`**

```
$ pwd
$ conda create --prefix ./env ipython=7.13 matplotlib=3.1 pandas=1.0 python=3.6
```

Working with Environments: How do I specify a location for a Conda environment?

You can control where a Conda environment lives by providing a path to a target directory when creating the environment.

For example the following command will create a new environment in a sub-directory of the current working directory called **`env`**

```
$ pwd
$ conda create --prefix ./env ipython=7.13 matplotlib=3.1 pandas=1.0 python=3.6
```

In unix the dot-slash, `./`, is a relative path a file or directory in the current directory.

Working with Environments: How do I specify a location for a Conda environment?

It is often a good idea to specify a path to a sub-directory of your project directory when creating an environment. Why?

Working with Environments: How do I specify a location for a Conda environment?

It is often a good idea to specify a path to a sub-directory of your project directory when creating an environment. Why?

Makes it easy to tell if your project utilizes an isolated environment by including the environment as a sub-directory.

Working with Environments: How do I specify a location for a Conda environment?

It is often a good idea to specify a path to a sub-directory of your project directory when creating an environment. Why?

Makes it easy to tell if your project utilizes an isolated environment by including the environment as a sub-directory.

Makes your project more self-contained as everything including the required software is contained in a single project directory.

Working with Environments: How do I specify a location for a Conda environment?

It is often a good idea to specify a path to a sub-directory of your project directory when creating an environment. Why?

Makes it easy to tell if your project utilizes an isolated environment by including the environment as a sub-directory.

Makes your project more self-contained as everything including the required software is contained in a single project directory.

An additional benefit of creating your project's environment inside a sub-directory is that you can then use the same name for all your environments; if you keep all of your environments in your `~/miniconda3/env/` folder, you'll have to give each of them a different name.

Exercise: Creating a new environment as a sub-directory within a project directory environment

First create a project directory called `project-dir` using the following command.

```
$ mkdir project-dir
$ cd project-dir
```

Next, create a new environment inside the newly created `project-dir` in a sub-directory called `env` an install

- Python 3.6,
- Matplotlib 3.1
- TensorFlow 2.0

Working with Environments:

Placing Conda environments outside of the default `~/miniconda3/envs/` folder comes with a couple of minor drawbacks.

1. `conda` can no longer find your environment with the `--name` flag; you'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.

Working with Environments:

Placing Conda environments outside of the default `~/miniconda3/envs/` folder comes with a couple of minor drawbacks.

1. `conda` can no longer find your environment with the `--name` flag; you'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.

2. Second, an annoying side-effect of specifying an install path when creating your Conda environments is that your command prompt is now prefixed with the active environment's absolute path rather than the environment's name. After activating an environment using its prefix your prompt will look similar to the following.

```
(/absolute/path/to/env) $
```

Working with Environments:

Placing Conda environments outside of the default `~/miniconda3/envs/` folder comes with a couple of minor drawbacks.

1. `conda` can no longer find your environment with the `--name` flag; you'll generally need to pass the `--prefix` flag along with the environment's full path to find the environment.

2. Second, an annoying side-effect of specifying an install path when creating your Conda environments is that your command prompt is now prefixed with the active environment's absolute path rather than the environment's name. After activating an environment using its prefix your prompt will look similar to the following.

```
(/absolute/path/to/env) $
(/Users/USER_NAME/research/data-science/PROJECT_NAME/env) $
```

this can quickly get out of hand.

Working with Environments:

If (you find this long prefix to your shell prompt annoying, then there is a quick fix: modify the env_prompt setting in your .condarc file, which you can do with the following command.

```
$ conda config --set env_prompt '({name})'
```

Working with Environments:

If (you find this long prefix to your shell prompt annoying, then there is a quick fix: modify the env_prompt setting in your .condarc file, which you can do with the following command.

```
$ conda config --set env_prompt '({name})'
```

Working with Environments:

This will either edit your `~/.condarc file` if you already have one or create a `~/.condarc` file if you do not.

Now your command prompt will display the active environment’s generic name.

```
$ cd project-directory
```

```
$ conda activate ./env
(env) project-directory $
```

Exercise: Activate an existing environment by path

1. Activate the environment created in a previous challenge. `project-dir.` using the path to the environment directory.

2. Create an environments for R project

First create a project directory called `r-project-dir` using the following command.

```
cd ~/

mkdir r-project-dir

cd r-project-dir
```

Create a new environment inside the newly created `r-project-dir` in a sub-directory called `env` and install

- `r-base`
- `r-tidyverse`

Working with Environments: Listing existing environments

Now that you have created a number of Conda environments on your local machine you have probably forgotten the names of all of the environments and exactly where they live.

This a conda command lists all of your existing environments together with their locations.

```
$ conda env list
```

Working with Environments: Listing the contents of environments

You will probably forget exactly what has been installed in a particular Conda environment.

To list the contents of the basic-scipy-env that you created above, run the following command.

```
$ conda list --name basic-scipy-env
```

Working with Environments: Listing the contents of environments

If you created your Conda environment using the **`--prefix`** option to install packages into a particular directory, then you will need to use that prefix in order for conda to locate the environment on your machine.

```
$ conda list --prefix /path/to/conda-env
```

Exercise: Listing the contents of a particular environment.

List the packages installed in the `machine-learning-env` environment that you created in a previous challenge.

Working with Environments: Deleting entire environments

Occasionally, you will want to delete an entire environment. The command to delete an environment is the following.

```
$ conda remove --name python36-env --all --dry-run

$ conda remove --name python36-env --all
```

--all	Remove all packages, i.e., the entire environment.
--dry-run	Only display what would have been done.

Working with Environments: Deleting entire environments

If you wish to delete and environment that you created with a --prefix option, then you will need to provide the prefix again when removing the environment.

```
$ conda remove --prefix /path/to/conda-env/ --all
```

Exercise: Delete an entire environment

Delete the entire **`machine-learning-env`** environment.

Hint try `--dry-run` first

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.
- You activate (deactivate) an environment using the `conda activate` (`conda deactivate`) commands.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.
- You activate (deactivate) an environment using the `conda activate` (`conda deactivate`) commands.
- You install packages into environments using `conda install`;

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.
- You activate (deactivate) an environment using the `conda activate` (`conda deactivate`) commands.
- You install packages into environments using `conda install`;
- You should install each environment as a sub-directory inside its corresponding project directory

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.
- You activate (deactivate) an environment using the `conda activate` (`conda deactivate`) commands.
- You install packages into environments using `conda install`;
- You should install each environment as a sub-directory inside its corresponding project directory
- Use the `conda env list` command to list existing environments and their respective locations.

Working with Environments: Key Points

- A Conda environment is a directory that contains a specific collection of Conda packages that you have installed.
- You create (remove) a new environment using the `conda create` (`conda remove`) commands.
- You activate (deactivate) an environment using the `conda activate` (`conda deactivate`) commands.
- You install packages into environments using `conda install`;
- You should install each environment as a sub-directory inside its corresponding project directory
- Use the `conda env list` command to list existing environments and their respective locations.
- Use the `conda list` command to list all of the packages installed in an environment.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/02-working-with-environments/index.html

Using Packages and Channels

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels

Using Packages and Channels: Questions

- What are Conda packages?
- What are Conda channels?
- Why should I be explicit about which channels my research project uses?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html

Using Packages and Channels: What are Packages?

A conda package is a compressed archive file (`.tar.bz2`) that contains:

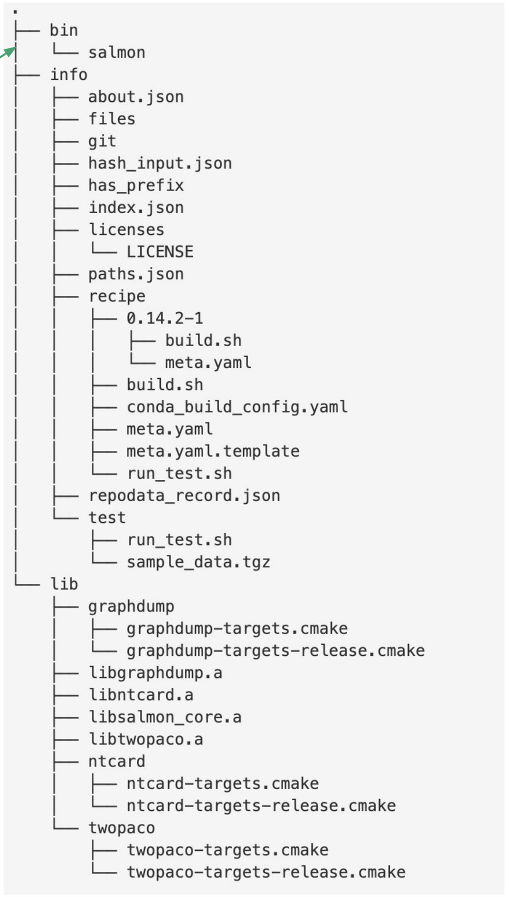
- system-level libraries
- Python or other modules
- executable programs and other components
- metadata under the `info/` directory
- a collection of files that are installed directly into an `install` prefix.

Conda keeps track of the dependencies between packages and platforms; the conda package format is identical across platforms and operating systems.

Using Packages and Channels: Package Structure?

All conda packages have a specific sub-directory structure inside the tarball file.

There is a `bin` directory that contains any binaries for the package;



```
.
├── bin
│   └── salmon
├── info
│   ├── about.json
│   ├── files
│   ├── git
│   ├── hash_input.json
│   ├── has_prefix
│   ├── index.json
│   ├── licenses
│   │   └── LICENSE
│   ├── paths.json
│   ├── recipe
│   │   ├── 0.14.2-1
│   │   │   ├── build.sh
│   │   │   └── meta.yaml
│   │   ├── build.sh
│   │   ├── conda_build_config.yaml
│   │   ├── meta.yaml
│   │   ├── meta.yaml.template
│   │   └── run_test.sh
│   ├── repodata_record.json
│   └── test
│       ├── run_test.sh
│       └── sample_data.tgz
└── lib
    ├── graphdump
    │   ├── graphdump-targets.cmake
    │   └── graphdump-targets-release.cmake
    ├── libgraphdump.a
    ├── libntcard.a
    ├── libsalmon_core.a
    ├── libtwopaco.a
    ├── ntcard
    │   ├── ntcard-targets.cmake
    │   └── ntcard-targets-release.cmake
    └── twopaco
        ├── twopaco-targets.cmake
        └── twopaco-targets-release.cmake
```

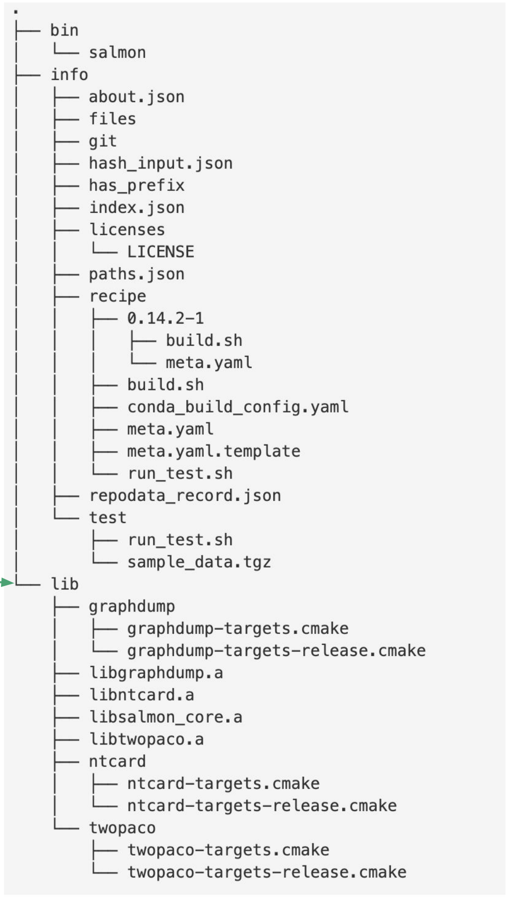
Using Packages and Channels: Package Structure?

All conda packages have a specific sub-directory structure inside the tarball file.

There is a `bin` directory that contains any binaries for the package;

a `lib` directory containing the relevant library files (i.e., the `.py` files);

and an `info` directory containing package metadata.



```
.
├── bin
│   └── salmon
├── info
│   ├── about.json
│   ├── files
│   ├── git
│   ├── hash_input.json
│   ├── has_prefix
│   ├── index.json
│   ├── licenses
│   │   └── LICENSE
│   ├── paths.json
│   ├── recipe
│   │   ├── 0.14.2-1
│   │   │   ├── build.sh
│   │   │   └── meta.yaml
│   │   ├── build.sh
│   │   ├── conda_build_config.yaml
│   │   ├── meta.yaml
│   │   ├── meta.yaml.template
│   │   └── run_test.sh
│   ├── repodata_record.json
│   └── test
│       ├── run_test.sh
│       └── sample_data.tgz
└── lib
    ├── graphdump
    │   ├── graphdump-targets.cmake
    │   └── graphdump-targets-release.cmake
    ├── libgraphdump.a
    ├── libntcard.a
    ├── libsalmon_core.a
    ├── libtwopaco.a
    ├── ntcard
    │   ├── ntcard-targets.cmake
    │   └── ntcard-targets-release.cmake
    └── twopaco
        ├── twopaco-targets.cmake
        └── twopaco-targets-release.cmake
```

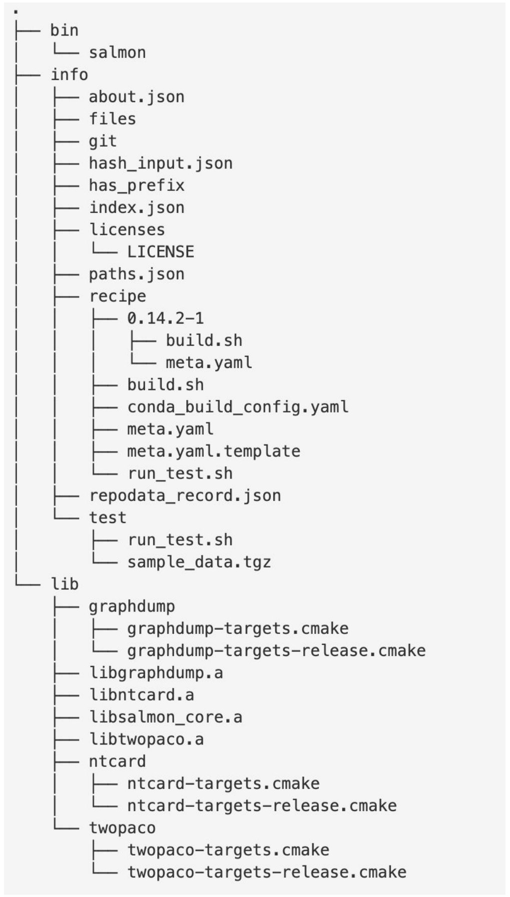
Using Packages and Channels: Package Structure?

All conda packages have a specific sub-directory structure inside the tarball file.

There is a `bin` directory that contains any binaries for the package;

a `lib` directory containing the relevant library files (i.e., the `.py` files);

and an `info` directory containing package metadata.



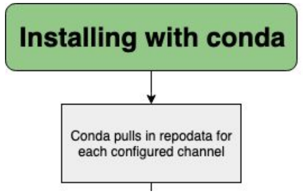
```
.
├── bin
│   └── salmon
├── info
│   ├── about.json
│   ├── files
│   ├── git
│   ├── hash_input.json
│   ├── has_prefix
│   ├── index.json
│   ├── licenses
│   │   └── LICENSE
│   ├── paths.json
│   ├── recipe
│   │   ├── 0.14.2-1
│   │   │   ├── build.sh
│   │   │   └── meta.yaml
│   │   ├── build.sh
│   │   ├── conda_build_config.yaml
│   │   ├── meta.yaml
│   │   ├── meta.yaml.template
│   │   └── run_test.sh
│   ├── repodata_record.json
│   └── test
│       ├── run_test.sh
│       └── sample_data.tgz
└── lib
    ├── graphdump
    │   ├── graphdump-targets.cmake
    │   └── graphdump-targets-release.cmake
    ├── libgraphdump.a
    ├── libntcard.a
    ├── libsalmon_core.a
    ├── libtwopaco.a
    ├── ntcard
    │   ├── ntcard-targets.cmake
    │   └── ntcard-targets-release.cmake
    └── twopaco
        ├── twopaco-targets.cmake
        └── twopaco-targets-release.cmake
```

Using Packages and Channels: What actually happens when I install packages?

Installing with conda

```
$ conda install super-fun-package=0
```

Using Packages and Channels: What actually happens when I install packages?



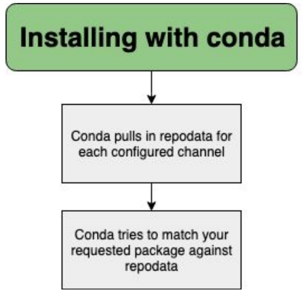
Conda pulls in repodata for each configured channel

```
- https://repo.anaconda.com/pkgs/main/linux-64
- https://repo.anaconda.com/pkgs/main/noarch
- https://repo.anaconda.com/pkgs/r/linux-64
- https://repo.anaconda.com/pkgs/r/noarch
```

https://repo.anaconda.com/pkgs/main/linux-64/repodata.json

```
$ conda install super-fun-package=0
```

Using Packages and Channels: What actually happens when I install packages?

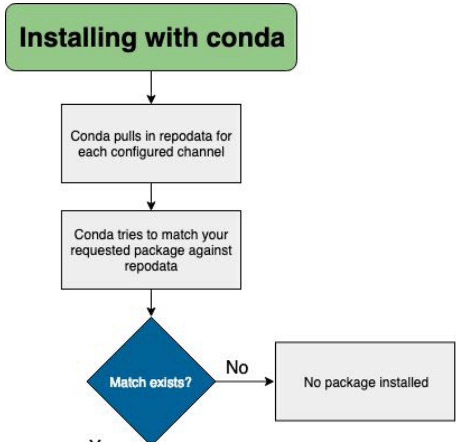

Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata

```
- https://repo.anaconda.com/pkgs/main/linux-64
- https://repo.anaconda.com/pkgs/main/noarch
- https://repo.anaconda.com/pkgs/r/linux-64
- https://repo.anaconda.com/pkgs/r/noarch
```

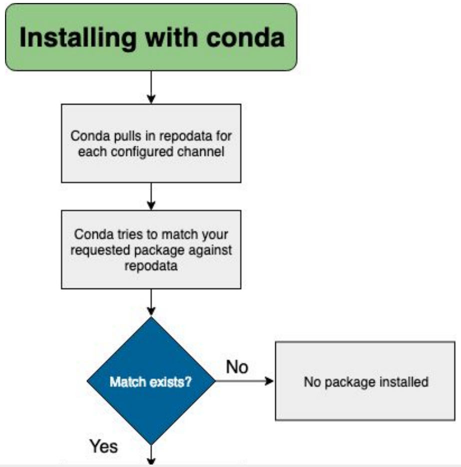
repodata.json

Conda tries to match your requested package against repodata

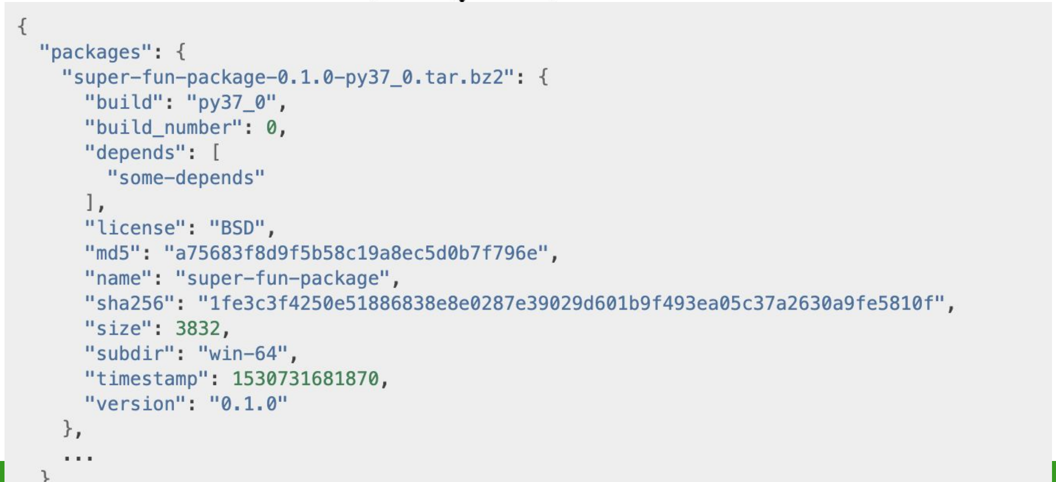
Using Packages and Channels: What actually happens when I install packages?


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed

Using Packages and Channels: What actually happens when I install packages?

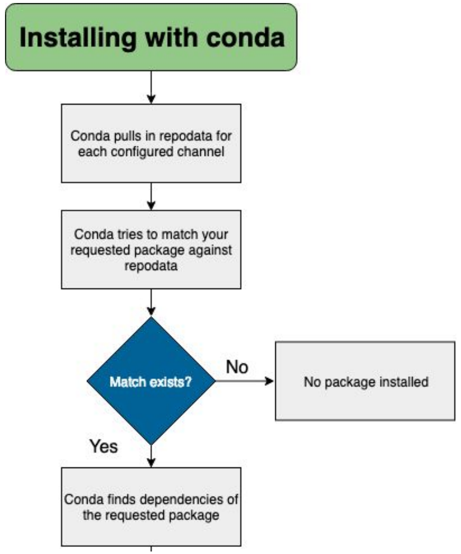

Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes

repodata.json



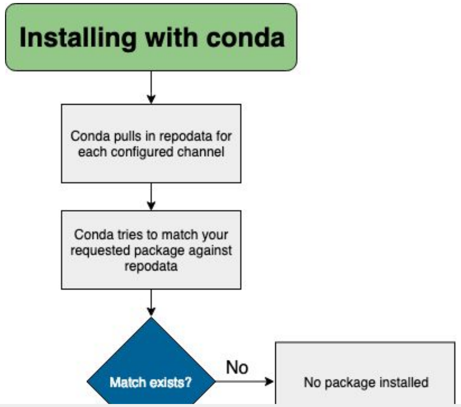
```
{
  "packages": {
    "super-fun-package-0.1.0-py37_0.tar.bz2": {
      "build": "py37_0",
      "build_number": 0,
      "depends": [
        "some-depends"
      ],
      "license": "BSD",
      "md5": "a75683f8d9f5b58c19a8ec5d0b7f796e",
      "name": "super-fun-package",
      "sha256": "1fe3c3f4250e51886838e8e0287e39029d601b9f493ea05c37a2630a9fe5810f",
      "size": 3832,
      "subdir": "win-64",
      "timestamp": 1530731681870,
      "version": "0.1.0"
    },
    ...
```

Using Packages and Channels: What actually happens when I install packages?


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package

Conda find dependencies of requested package

Using Packages and Channels: What actually happens when I install packages?

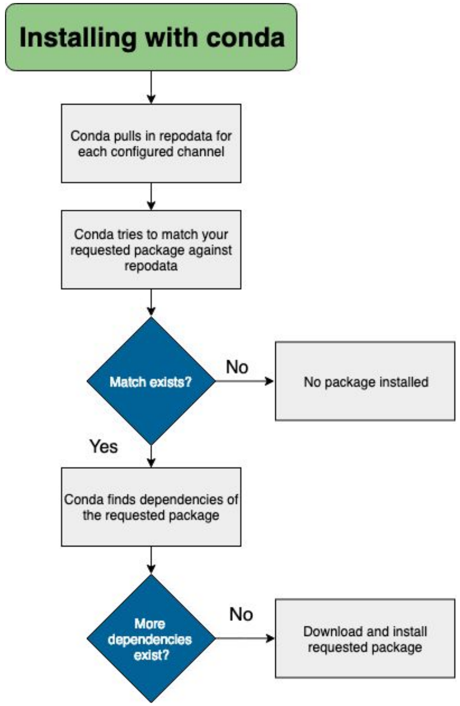

Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed

repodata.json

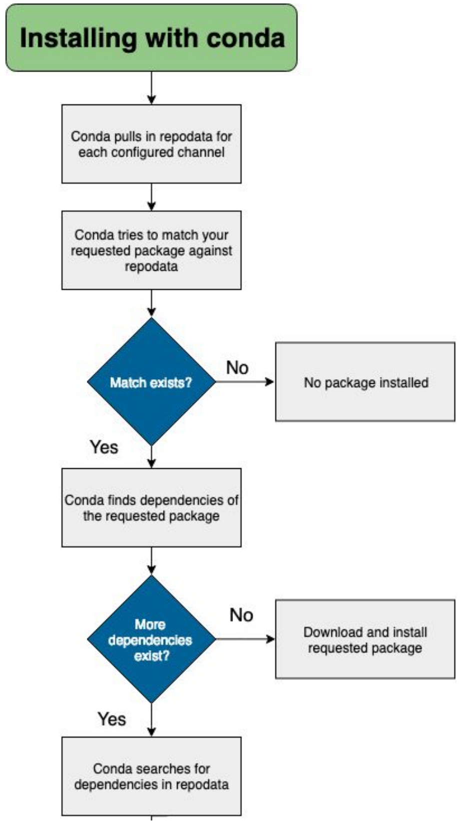
```
{
  "packages": {
    "super-fun-package-0.1.0-py37_0.tar.bz2": {
      "build": "py37_0",
      "build_number": 0,
      "depends": [
        "some-depends"
      ],
      "license": "BSD",
      "md5": "a75683f8d9f5b58c19a8ec5d0b7f796e",
      "name": "super-fun-package",
      "sha256": "1fe3c3f4250e51886838e8e0287e39029d601b9f493ea05c37a2630a9fe5810f",
      "size": 3832,
      "subdir": "win-64",
      "timestamp": 1530731681870,
      "version": "0.1.0"
    },
    ...
  }
```

dependencies

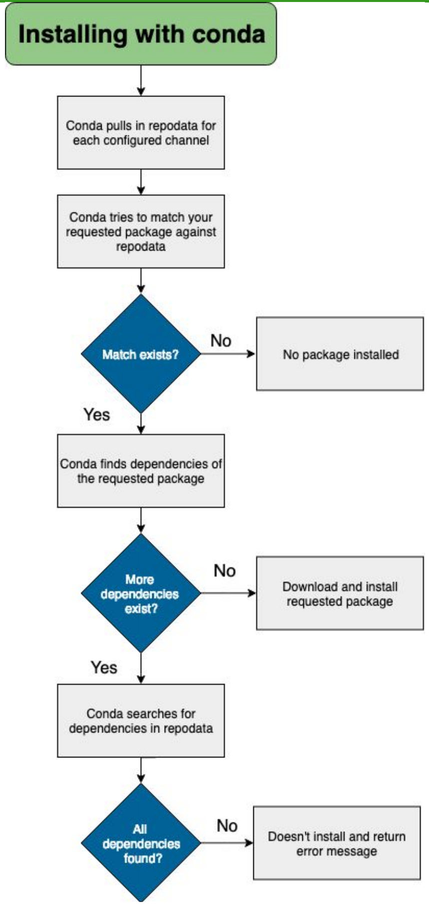
Using Packages and Channels: What actually happens when I install packages?


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package

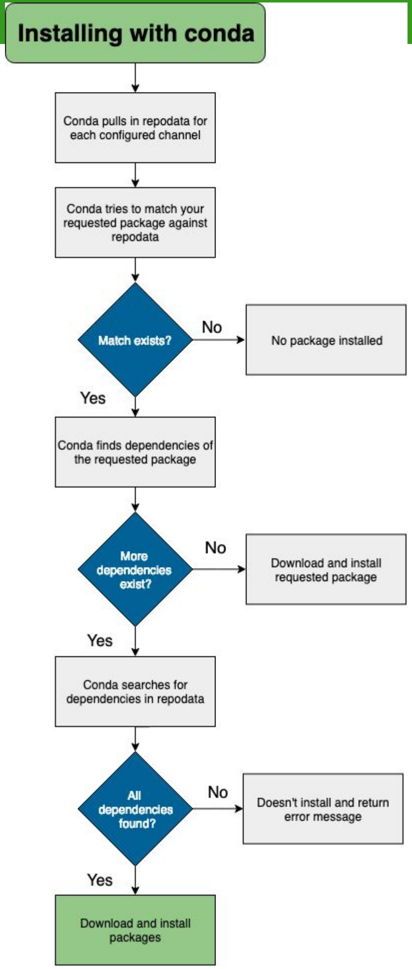
Using Packages and Channels: What actually happens when I install packages?


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata

Using Packages and Channels: What actually happens when I install packages?


Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata
All dependencies found?
No
Doesn't install and return error message

Using Packages and Channels: What actually happens when I install packages?

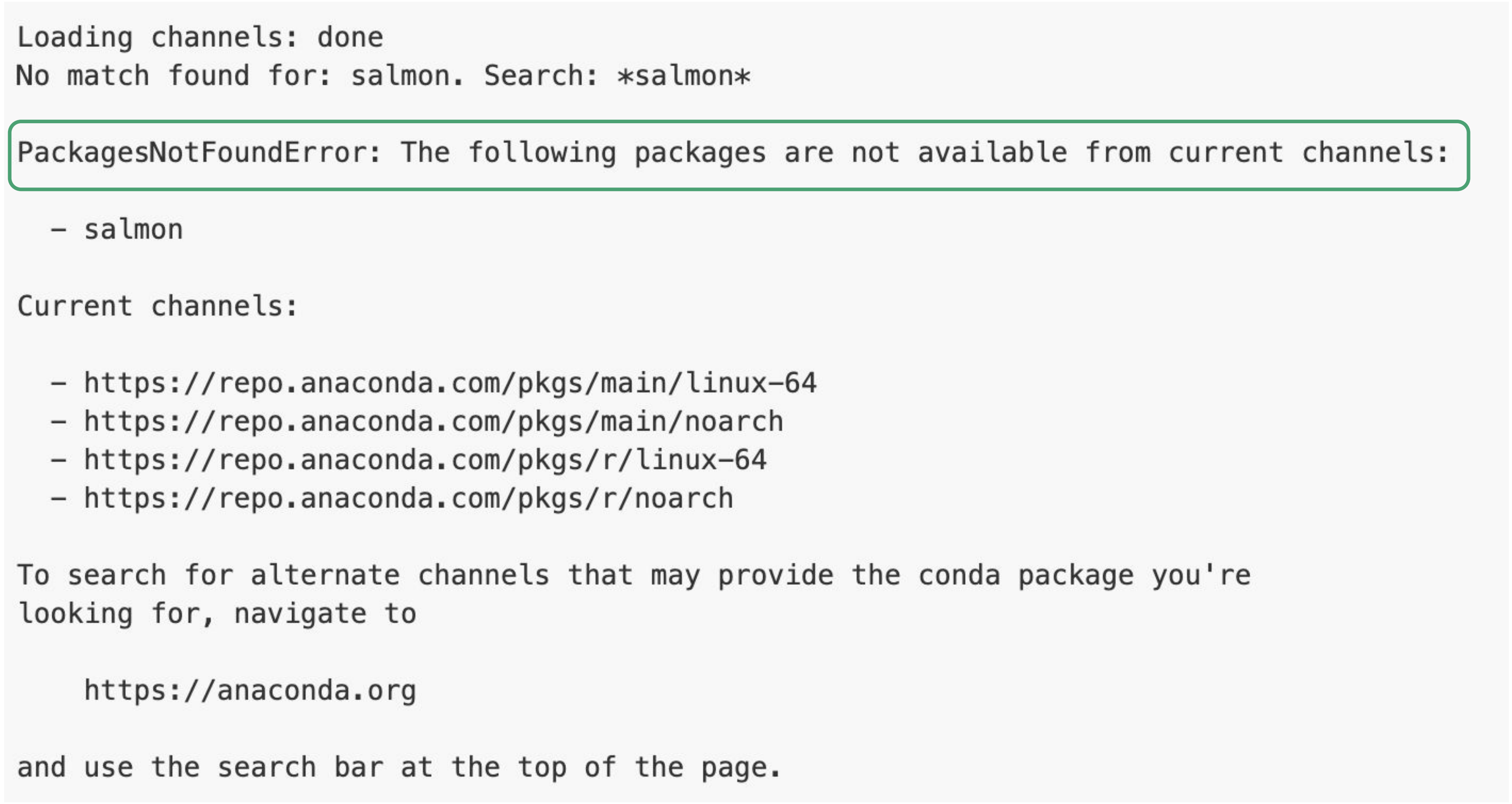

Installing with conda
Conda pulls in repodata for each configured channel
Conda tries to match your requested package against repodata
Match exists?
No
No package installed
Yes
Conda finds dependencies of the requested package
More dependencies exist?
No
Download and install requested package
Yes
Conda searches for dependencies in repodata
All dependencies found?
No
Doesn't install and return error message
Yes
Download and install packages

Using Packages and Channels: Installing salmon package

Let's create a new environment `basic-rnaseq-env` and install a transcript quantification package, salmon , for an RNA-Seq analysis project.

```
$ conda create --name basic-rnaseq-env salmon
```

Using Packages and Channels: Installing salmon package



```
Loading channels: done
No match found for: salmon. Search: *salmon*

PackagesNotFoundError: The following packages are not available from current channels:

  - salmon

Current channels:

  - https://repo.anaconda.com/pkgs/main/linux-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/linux-64
  - https://repo.anaconda.com/pkgs/r/noarch

To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org

and use the search bar at the top of the page.
```

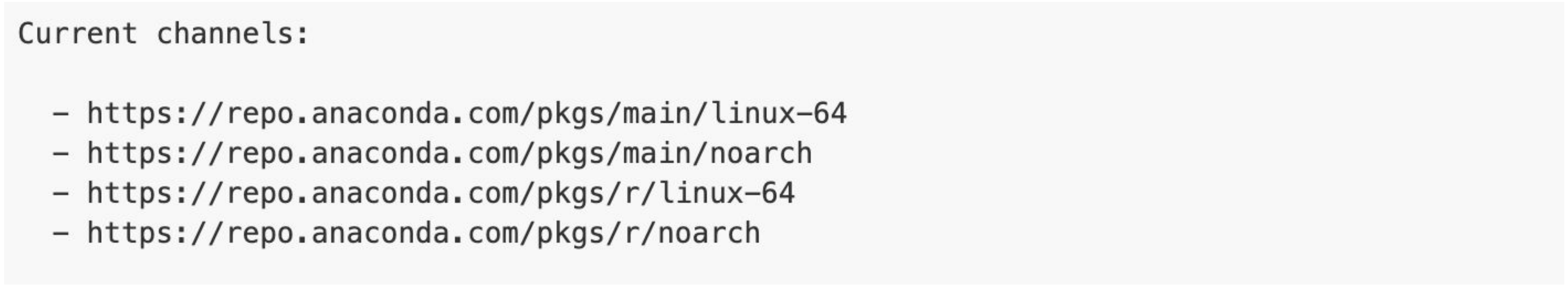
Using Packages and Channels: What are Conda channels?

When you `install` or `search` for a package in conda it searches for it in remote repositories called channels.

Using Packages and Channels: What are Conda channels?

When you `install` or `search` for a package in conda it searches for it in remote repositories called channels.

These remote channel are URLs to directories containing conda packages.



```
Current channels:

  - https://repo.anaconda.com/pkgs/main/linux-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/linux-64
  - https://repo.anaconda.com/pkgs/r/noarch
```

Using Packages and Channels: What are Conda channels?

When you `install` or `search` for a package in conda it searches for it in remote repositories called channels.

These remote channel are URLs to directories containing conda packages.

```
Current channels:

  - https://repo.anaconda.com/pkgs/main/linux-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/linux-64
  - https://repo.anaconda.com/pkgs/r/noarch
```

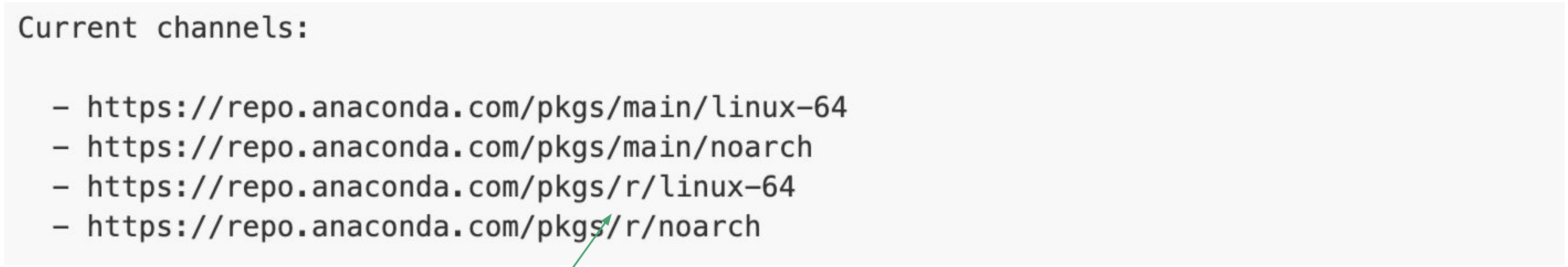
By default the `conda search` command searches a set of channels defined here. Anaconda Cloud channels.

- `main`: The majority of all new Anaconda, Inc. package builds are hosted here. Included in conda's defaults channel as the top priority channel.

Using Packages and Channels: What are Conda channels?

When you `install` or `search` for a package in conda it searches for it in remote repositories called channels.

These remote channel are URLs to directories containing conda packages.



```
Current channels:

  - https://repo.anaconda.com/pkgs/main/linux-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/linux-64
  - https://repo.anaconda.com/pkgs/r/noarch
```

By default the `conda search` command searches a set of channels defined here. Anaconda Cloud channels.

- `main`: The majority of all new Anaconda, Inc. package builds are hosted here. Included in conda's defaults channel as the top priority channel.
- `r`: Microsoft R Open conda packages and Anaconda, Inc.'s R conda packages. This channel is included in conda's defaults channel. When creating new environments, MRO is now chosen as the default R implementation.

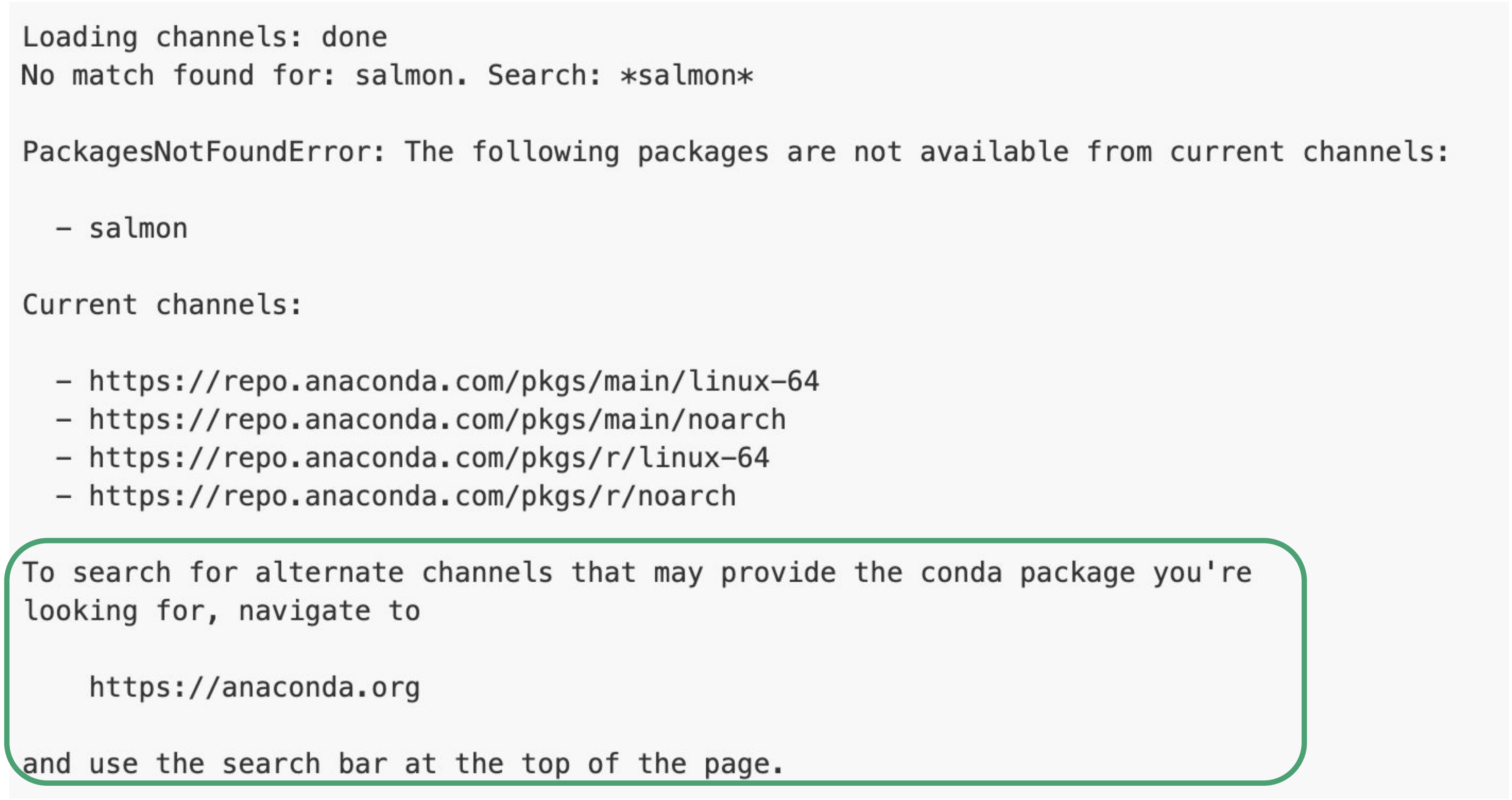
Using Packages and Channels: What are Conda channels?

Collectively, the Anaconda managed channels are referred to as the `defaults` channel because, unless otherwise specified, packages installed using `conda` will be downloaded from these channels.

Using Packages and Channels: My package isn't available on the `defaults`

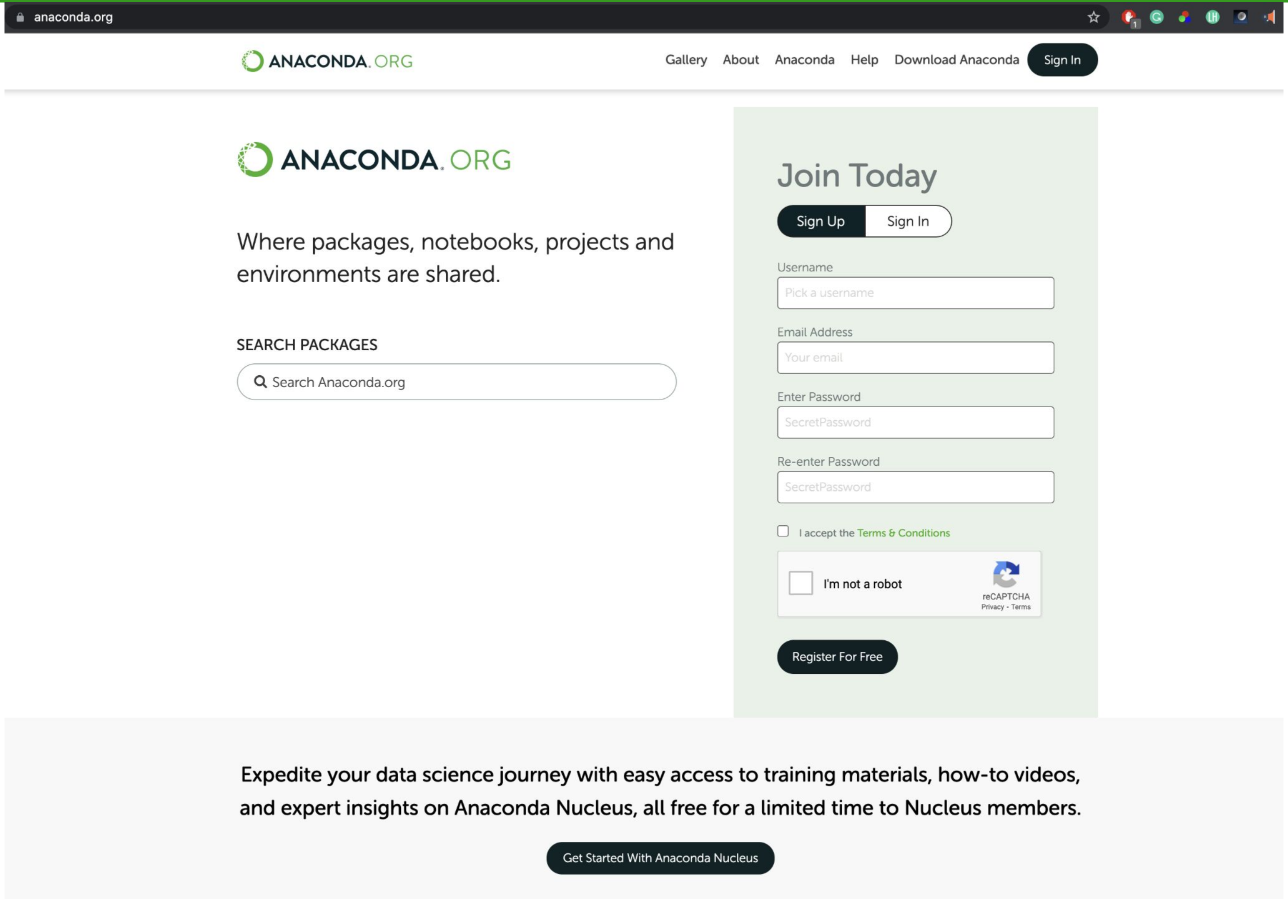
What should I do ?

Using Packages and Channels: My package isn't available on the `defaults`



```
Loading channels: done
No match found for: salmon. Search: *salmon*

PackagesNotFoundError: The following packages are not available from current channels:

  - salmon

Current channels:

  - https://repo.anaconda.com/pkgs/main/linux-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/linux-64
  - https://repo.anaconda.com/pkgs/r/noarch

To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org

and use the search bar at the top of the page.
```

Using Packages and Channels: Anaconda.org


anaconda.org

ANACONDA.ORG

Gallery About Anaconda Help Download Anaconda Sign In

ANACONDA.ORG

Where packages, notebooks, projects and environments are shared.

SEARCH PACKAGES

Search Anaconda.org

Join Today

Sign Up | Sign In

Username: Pick a username

Email Address: Your email

Enter Password: SecretPassword

Re-enter Password: SecretPassword

I accept the Terms & Conditions

I'm not a robot

reCAPTCHA

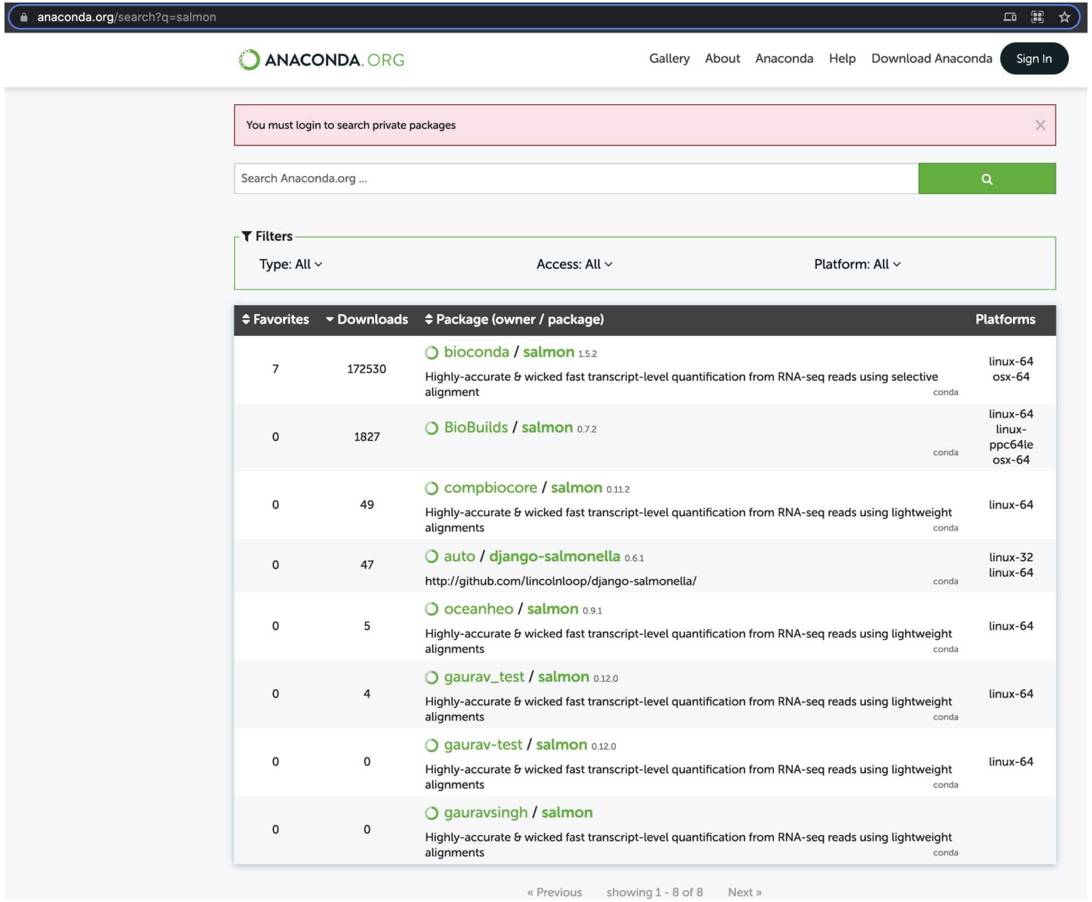
Privacy - Terms

Register For Free

Expedite your data science journey with easy access to training materials, how-to videos, and expert insights on Anaconda Nucleus, all free for a limited time to Nucleus members.

Get Started With Anaconda Nucleus

Using Packages and Channels: Anaconda.org



anaconda.org/search?q=salmon

ANACONDA.ORG — Gallery | About | Anaconda | Help | Download Anaconda | Sign In

You must login to search private packages

Search Anaconda.org ...

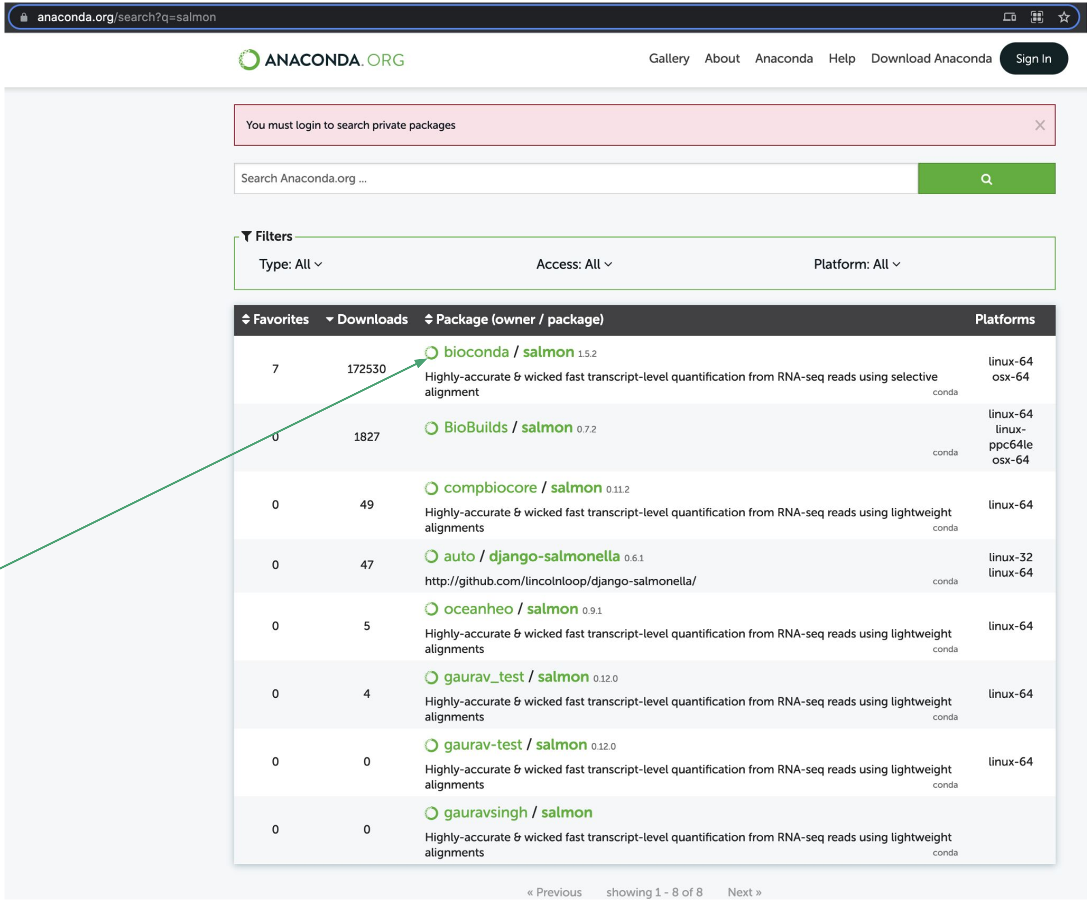
Filters

Type: All — Access: All — Platform: All

Favorites	Downloads	Package (owner / package)	Platforms
7	172530	bioconda / salmon 1.5.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using selective alignment conda	linux-64 osx-64
0	1827	BioBuilds / salmon 0.7.2 conda	linux-64 linux-ppc64le osx-64
0	49	compbiocore / salmon 0.11.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	47	auto / django-salmonella 0.6.1 http://github.com/lincolnloop/django-salmonella/ conda	linux-32 linux-64
0	5	oceanheo / salmon 0.9.1 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	4	gaurav_test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gaurav-test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gauravsingh / salmon Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	

« Previous   showing 1 - 8 of 8   Next »

Using Packages and Channels: Anaconda.org



anaconda.org/search?q=salmon

ANACONDA.ORG   Gallery   About   Anaconda   Help   Download Anaconda   Sign In

You must login to search private packages

Search Anaconda.org ...

Filters

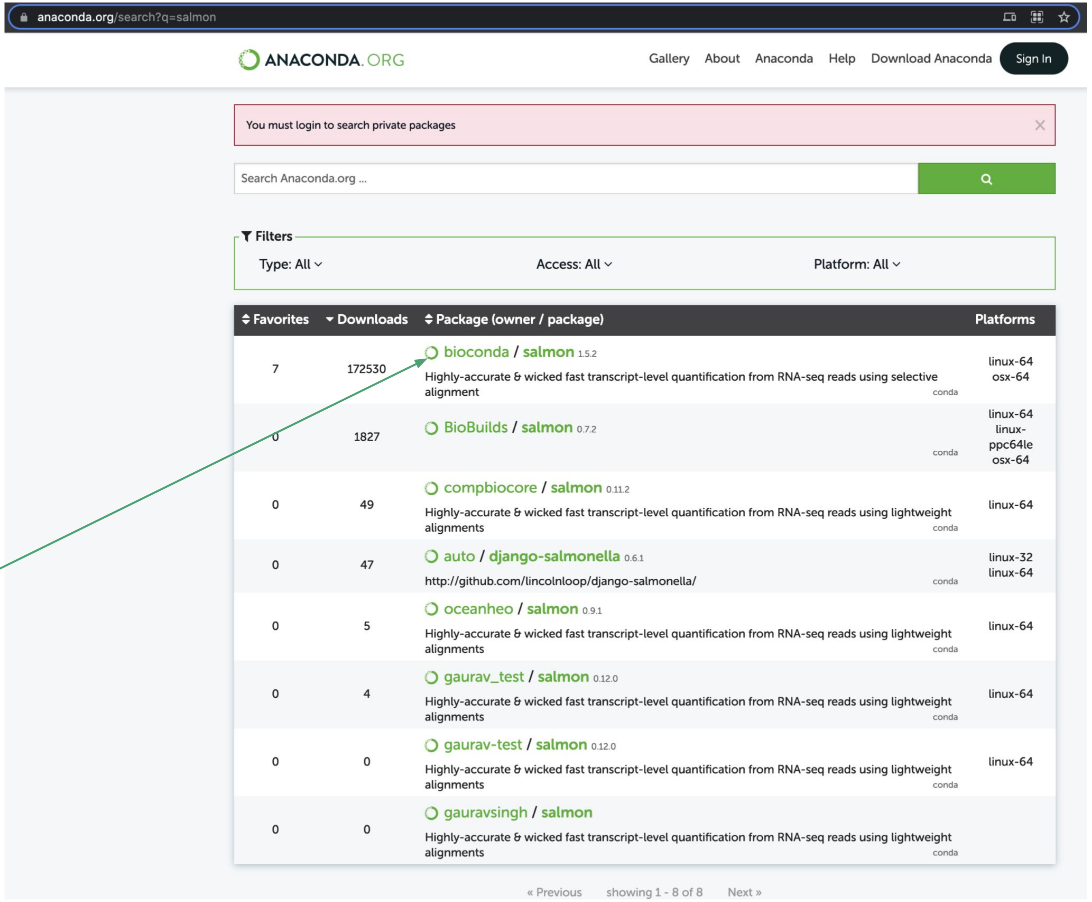
Type: All   Access: All   Platform: All

Favorites	Downloads	Package (owner / package)	Platforms
7	172530	bioconda / salmon 1.5.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using selective alignment conda	linux-64 osx-64
0	1827	BioBuilds / salmon 0.7.2 conda	linux-64 linux-ppc64le osx-64
0	49	compbiocore / salmon 0.11.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	47	auto / django-salmonella 0.6.1 http://github.com/lincolnloop/django-salmonella/ conda	linux-32 linux-64
0	5	oceanheo / salmon 0.9.1 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	4	gaurav_test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gaurav-test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gauravsingh / salmon Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	

« Previous   showing 1 - 8 of 8   Next »

channels

Using Packages and Channels: Anaconda.org



anaconda.org/search?q=salmon

ANACONDA.ORG   Gallery About Anaconda Help Download Anaconda Sign In

You must login to search private packages

Search Anaconda.org ...

Filters

Type: All   Access: All   Platform: All

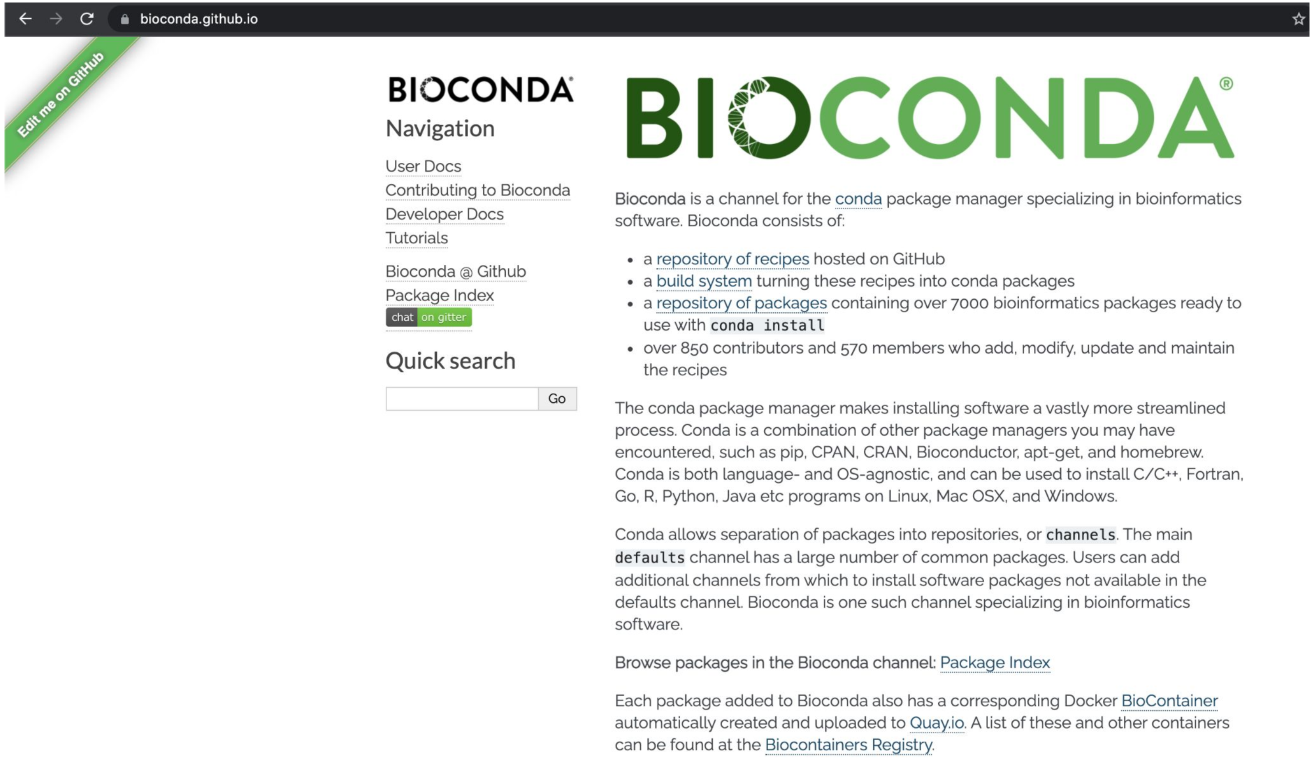
Favorites	Downloads	Package (owner / package)	Platforms
7	172530	bioconda / salmon 1.5.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using selective alignment conda	linux-64 osx-64
0	1827	BioBuilds / salmon 0.7.2 conda	linux-64 linux-ppc64le osx-64
0	49	compbiocore / salmon 0.11.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	47	auto / django-salmonella 0.6.1 http://github.com/lincolnloop/django-salmonella/ conda	linux-32 linux-64
0	5	oceanheo / salmon 0.9.1 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	4	gaurav_test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gaurav-test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gauravsingh / salmon Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	

« Previous   showing 1 - 8 of 8   Next »

channels

Using Packages and Channels: bioconda

Bioconda is a channel, maintained by the Bioconda project, specialising in bioinformatics software. Bioconda contains 1000's of bioinformatics packages ready to use with conda install.



Navigation

- User Docs
- Contributing to Bioconda
- Developer Docs
- Tutorials
- Bioconda @ Github
- Package Index

Quick search

Bioconda is a channel for the conda package manager specializing in bioinformatics software. Bioconda consists of:

- a repository of recipes hosted on GitHub
- a build system turning these recipes into conda packages
- a repository of packages containing over 7000 bioinformatics packages ready to use with `conda install`
- over 850 contributors and 570 members who add, modify, update and maintain the recipes

The conda package manager makes installing software a vastly more streamlined process. Conda is a combination of other package managers you may have encountered, such as pip, CPAN, CRAN, Bioconductor, apt-get, and homebrew. Conda is both language- and OS-agnostic, and can be used to install C/C++, Fortran, Go, R, Python, Java etc programs on Linux, Mac OSX, and Windows.

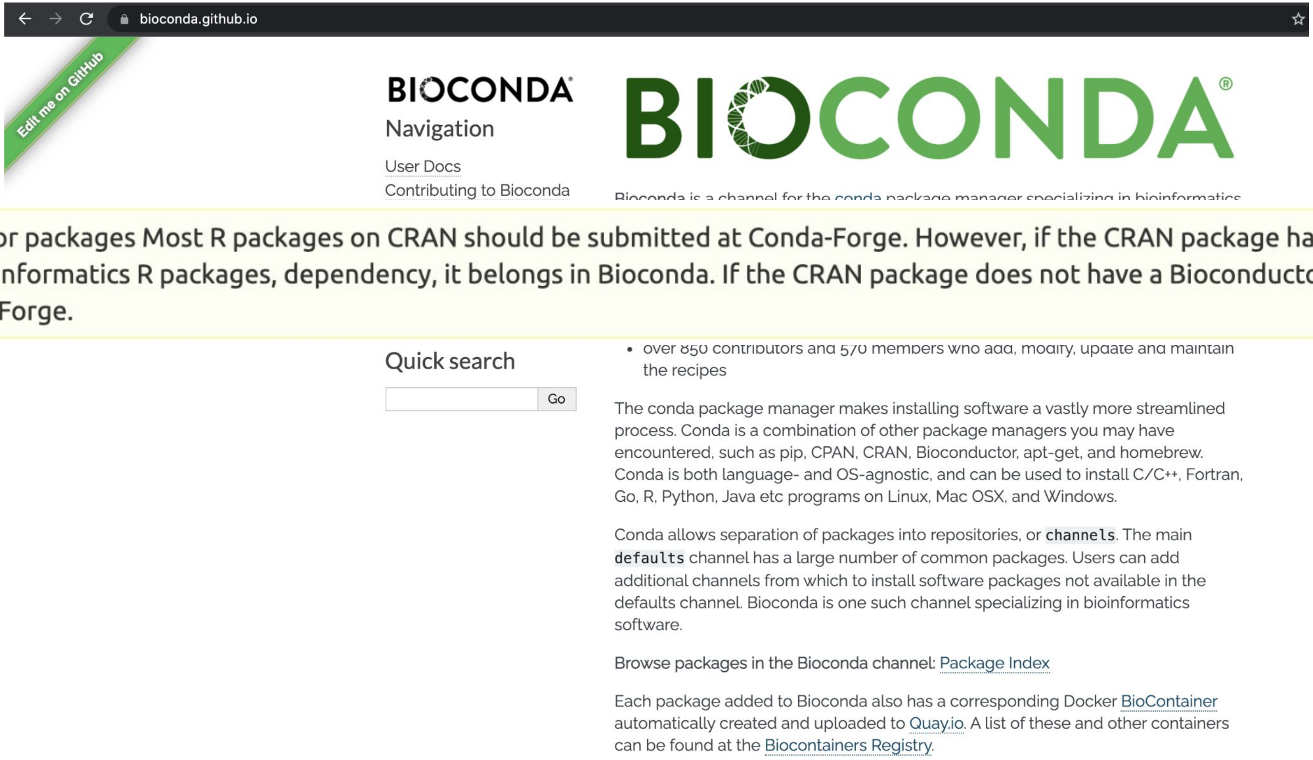
Conda allows separation of packages into repositories, or `channels`. The main `defaults` channel has a large number of common packages. Users can add additional channels from which to install software packages not available in the defaults channel. Bioconda is one such channel specializing in bioinformatics software.

Browse packages in the Bioconda channel: Package Index

Each package added to Bioconda also has a corresponding Docker BioContainer automatically created and uploaded to Quay.io. A list of these and other containers can be found at the Biocontainers Registry.

Using Packages and Channels: bioconda

Bioconda is a channel, maintained by the Bioconda project, specialising in bioinformatics software. Bioconda contains 1000’s of bioinformatics packages ready to use with conda install.



Bioconda is a channel for the conda package manager specializing in bioinformatics

R and Bioconductor packages Most R packages on CRAN should be submitted at Conda-Forge. However, if the CRAN package has a Bioconductor, a repository for bioinformatics R packages, dependency, it belongs in Bioconda. If the CRAN package does not have a Bioconductor package dependency, it belongs in Conda-Forge.

- over 850 contributors and 570 members who add, modify, update and maintain the recipes

The conda package manager makes installing software a vastly more streamlined process. Conda is a combination of other package managers you may have encountered, such as pip, CPAN, CRAN, Bioconductor, apt-get, and homebrew. Conda is both language- and OS-agnostic, and can be used to install C/C++, Fortran, Go, R, Python, Java etc programs on Linux, Mac OSX, and Windows.

Conda allows separation of packages into repositories, or `channels`. The main `defaults` channel has a large number of common packages. Users can add additional channels from which to install software packages not available in the defaults channel. Bioconda is one such channel specializing in bioinformatics software.

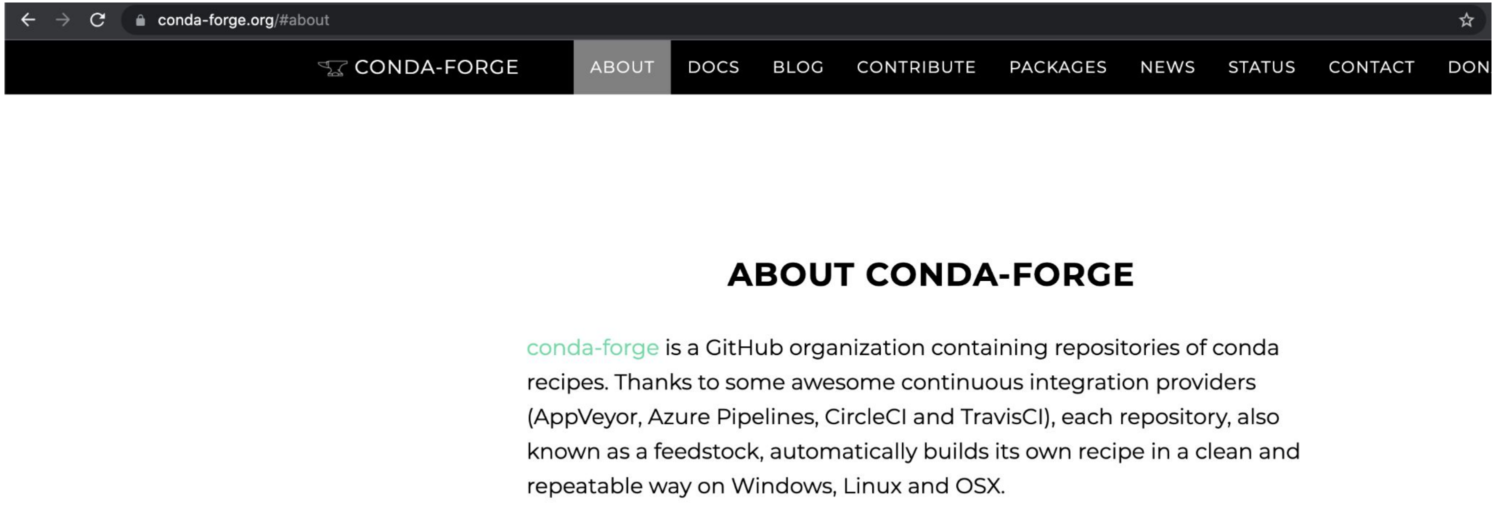
Browse packages in the Bioconda channel: Package Index

Each package added to Bioconda also has a corresponding Docker BioContainer automatically created and uploaded to Quay.io. A list of these and other containers can be found at the Biocontainers Registry.

Using Packages and Channels: conda-forge

In addition to the `default` channels that are managed by Anaconda Inc., there is another channel called `Conda-Forge` that also has a special status.

The Conda-Forge project "is a *community* led collection of recipes, build infrastructure and distributions for the conda package manager."



ABOUT CONDA-FORGE

conda-forge is a GitHub organization containing repositories of conda recipes. Thanks to some awesome continuous integration providers (AppVeyor, Azure Pipelines, CircleCI and TravisCI), each repository, also known as a feedstock, automatically builds its own recipe in a clean and repeatable way on Windows, Linux and OSX.

Using Packages and Channels: conda-forge

There are a number of reasons that you may wish to use the `conda-forge` channel instead of the `defaults` channel maintained by Anaconda:

Using Packages and Channels: conda-forge

There are a number of reasons that you may wish to use the `conda-forge` channel instead of the `defaults` channel maintained by Anaconda:

1. Packages on `conda-forge` may be more up-to-date than those on the `defaults` channel.

Using Packages and Channels: conda-forge

There are a number of reasons that you may wish to use the `conda-forge` channel instead of the `defaults` channel maintained by Anaconda:

1. Packages on `conda-forge` may be more up-to-date than those on the `defaults` channel.
2. There are packages on the `conda-forge` channel that aren’t available from `defaults`.

Using Packages and Channels: conda-forge

There are a number of reasons that you may wish to use the `conda-forge` channel instead of the `defaults` channel maintained by Anaconda:

1. Packages on `conda-forge` may be more up-to-date than those on the `defaults` channel.
2. There are packages on the `conda-forge` channel that aren't available from `defaults`.
3. You may wish to use a dependency such as `openblas` (from `conda-forge`) instead of `mkl`(from `defaults`).

Using Packages and Channels: Search Channel

If you know the channel your package is likely to be located on, you can can use the `conda search` command with the `--channel` option and the name of the channel.

```
$ conda search --channel bioconda salmon
```

Using Packages and Channels: Install from Channel

If you know the channel your package is available from you can install a package from a specific channel into the currently activate environment by passing the `--channel` or `-c` option to the `conda install` command as follows.

```
$ conda create --name basic-rnaseq-env

$ conda activate basic-rnaseq-env

$ conda install --channel bioconda salmon
```

Using Packages and Channels: Install from Channel

You can also install a package from a specific channel into a named environment (using `--name` or `-n`)

Or into an environment installed at a particular prefix (using `--prefix` or `-p`).

```
$ conda install salmon --channel bioconda --name basic-rnaseq-env
```

```
$ conda install salmon --channel bioconda --prefix ./env
```

Where would this command would install `salmon` package ?
What version of salmon is installed?

Using Packages and Channels: Install from Channel

You may have noticed that we didn't manage to install the latest version for salmon, why?

The `bioconda` channel contains bioinformatics packages (salmon, STAR, samtools, DESeq2, etc.), however the channel `conda-forge` has most of the dependencies (numpy, scipy, zlib, CRAN packages, etc.) needed.

We need to specify multiple channels to install the latest version.

Using Packages and Channels: Multiple Channels

To specify multiple channels for installing packages by passing the `--channel` argument multiple times.

```
$ conda install salmon=1.5 --channel conda-forge --channel bioconda --name basic-rnaseq-env
```

Using Packages and Channels: Multiple Channels

To specify multiple channels for installing packages by passing the `--channel` argument multiple times.

This also works when install multiple packages.

```
$ conda install salmon=1.5 --channel conda-forge --channel bioconda --name basic-rnaseq-env
```

```
$ conda install fastqc=0.11 multiqc=1.10 --channel conda-forge --channel bioconda --name
basic-rnaseq-env
```

Exercise: Specifying channels when installing packages

Create a new directory called `rnaseq-project` and then create an environment in a sub-directory called `env/` with the the packages

- `salmon=1.5`
- `fastqc=0.11`
- `multiqc=1.11`

Hint: For the lazy typers: the `--channel` argument can also be shortened to `-c`

Exercise: Alternative syntax for installing packages from specific channels

There exists an alternative syntax for installing conda packages from specific channels that more explicitly links the channel being used to install a particular package.

```
channel::<package_name>
```

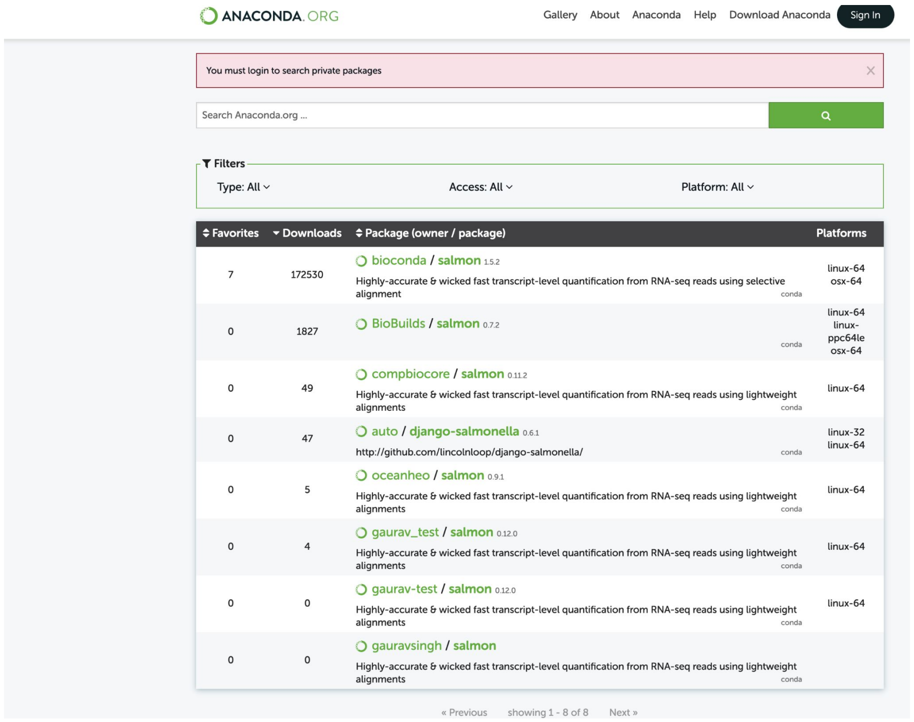
```
$ conda install biconda::multiqc  --prefix ./env
```

Install the latest version of the workflow manager **`nextflow`** using this alternative syntax

Using Packages and Channels: Channel Priority

Different channels can have the same package, so conda must decide which channel to install the package from.

Conda channels have a priority hierarchy.



You must login to search private packages

Search Anaconda.org ...

Filters

Type: All

Access: All

Platform: All

Favorites	Downloads	Package (owner / package)	Platforms
7	172530	bioconda / salmon 1.5.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using selective alignment conda	linux-64 osx-64
0	1827	BioBuilds / salmon 0.7.2 conda	linux-64 linux-ppc64le osx-64
0	49	compbiocore / salmon 0.11.2 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	47	auto / django-salmonella 0.6.1 http://github.com/lincolnloop/django-salmonella/ conda	linux-32 linux-64
0	5	oceanheo / salmon 0.9.1 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	4	gaurav_test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gaurav-test / salmon 0.12.0 Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	linux-64
0	0	gauravsingh / salmon Highly-accurate & wicked fast transcript-level quantification from RNA-seq reads using lightweight alignments conda	

« Previous showing 1 - 8 of 8 Next »

Using Packages and Channels: Channel Priority

Different channels can have the same package, so conda must decide which channel to install the package from. Conda channels have a priority hierarchy.

By default, conda prefers packages from a higher priority channel over any version from a lower priority channel.

Using Packages and Channels: Channel Priority

Channel priority listed on the command line decreases from left to right.

So if you were to install base R `r-base` using the command below.

```
$ conda create --name rproject-env --channel defaults --channel conda-forge
r-base
```

The first channel `defaults` would have a higher priority than the second `conda-forge`.

This is true even, if the version number of the package is higher in the second channel.

Using Packages and Channels: Channel Priority

Different channels can have the same package, so conda must decide which channel to install the package from. Conda channels have a priority hierarchy.

By default, conda prefers packages from a higher priority channel over any version from a lower priority channel.

Higher Priority ←

```
$ conda install salmon=1.5 --channel conda-forge --channel bioconda --name
basic-rnaseq-env
```

The bioconda team suggests that the `conda-forge` channel be a higher priority than the `bioconda` channel.

Using Packages and Channels: Channel Priority

Different channels can have the same package, so conda must decide which channel to install the package from. Conda channels have a priority hierarchy.

By default, conda prefers packages from a higher priority channel over any version from a lower priority channel.

Conda collects all of the packages with the same name across all listed channels and processes them as follows:

Using Packages and Channels: Channel Priority

1. Sorts packages from highest to lowest channel priority.

Using Packages and Channels: Channel Priority

1. Sorts packages from highest to lowest channel priority.

2. Sorts tied packages—packages with the same channel priority—from highest to lowest version number.
   a. For example, if `channelA` contains `NumPy 1.12.0` and `1.13.1`, NumPy 1.13.1 will be sorted higher.

Using Packages and Channels: Channel Priority

1. Sorts packages from highest to lowest channel priority.

2. Sorts tied packages—packages with the same channel priority—from highest to lowest version number. For example, if channelA contains NumPy 1.12.0 and 1.13.1, NumPy 1.13.1 will be sorted higher.

3. Sorts still-tied packages, packages with the same channel priority and same version, from highest to lowest build number.
   a. For example, if channelA contains both NumPy `1.12.0 build 1` and `build 2`,
   b. *`build 2`* is sorted first.
   c. Any packages in channelB would be sorted below those in channelA.

Using Packages and Channels: Channel Priority

1. Sorts packages from highest to lowest channel priority.

2. Sorts tied packages—packages with the same channel priority—from highest to lowest version number. For example, if channelA contains NumPy 1.12.0 and 1.13.1, NumPy 1.13.1 will be sorted higher.

3. Sorts still-tied packages, packages with the same channel priority and same version, from highest to lowest build number. For example, if channelA contains both NumPy 1.12.0 build 1 and build 2, build 2 is sorted first. Any packages in channelB would be sorted below those in channelA.

4. Installs the first package on the sorted list that satisfies the installation specifications.

Using Packages and Channels: Pip

If a Python package that you need isn’t available on any Conda channel, then you can use the default Python package manager Pip to install this package from PyPI.

Using Packages and Channels: Pip

If a Python package that you need isn’t available on any Conda channel, then you can use the default Python package manager Pip to install this package from PyPI.

However, there are a few potential issues that you should be aware of when using Pip to install Python packages when using Conda.

Using Packages and Channels: Pip

Pip is sometimes installed by default on operating systems where it is used to manage any Python packages needed by your OS.

You do not want to use this `pip` to install Python packages when using Conda environments.

Need to deactivate first

```
(base) $ conda deactivate
$ which python
/usr/bin/python
$ which pip # sometimes installed as pip3
 /usr/bin/pip
```

Using Packages and Channels: Pip

Second, Pip is also included in the Miniconda installer where it is used to install and manage OS specific Python packages required to setup your base Conda environment.

You do not want to use this `pip` to install Python packages when using Conda environments.

```
$ conda activate
(base) $ which python
~/miniconda3/bin/python
$ which pip
~/miniconda3/bin/pip
```

Using Packages and Channels: Pip

If you find yourself needing to install a Python package that is only available via Pip, then you should first install `pip` into your Conda environment and then use that `pip` to install the desired package.

```
python -m pip install <package_name>
```

Exercise: Installing packages into Conda environments using pip

pandas pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation too

Activate the conda environment `python3-env` and use `pip` to install the python package `pandas`.

Using Packages and Channels: Key Points

- A package is a tarball containing system-level libraries, Python or other modules, executable programs and other components, and associated metadata.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html

Using Packages and Channels: Key Points

- A package is a tarball containing system-level libraries, Python or other modules, executable programs and other components, and associated metadata.
- A Conda channel is a URL to a directory containing a Conda package(s).

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html

Using Packages and Channels: Key Points

- A package is a tarball containing system-level libraries, Python or other modules, executable programs and other components, and associated metadata.
- A Conda channel is a URL to a directory containing a Conda package(s).
- You can specific a conda channel using the option `--channel` or add it to your `.condarc`

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html

Using Packages and Channels: Key Points

- A package is a tarball containing system-level libraries, Python or other modules, executable programs and other components, and associated metadata.
- A Conda channel is a URL to a directory containing a Conda package(s).
- You can specific a conda channel using the option `--channel` or add it to your `.condarc`
- If a python package isn't available on a conda channel you can install it into your environment using the python package installer `pip`.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/03-using-packages-and-channels/index.html

Sharing Environments

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Questions

- Why should I share my Conda environment with others?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Questions

- Why should I share my Conda environment with others?
- How do I share my Conda environment with others?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Questions

- Why should I share my Conda environment with others?
- How do I share my Conda environment with others?
- How do I create an environment file that can be read by Windows, Mac OS, or Linux.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Questions

- Why should I share my Conda environment with others?
- How do I share my Conda environment with others?
- How do I create an environment file that can be read by Windows, Mac OS, or Linux.
- How do I specifying the package version in a Conda environment file.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Reproducible Research

Conda environments are useful when making bioinformatics projects reproducible.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Reproducible Research

Conda environments are useful when making bioinformatics projects reproducible.

Full reproducibility requires the ability to recreate the system that was originally used to generate the results.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Reproducible Research

Conda environments are useful when making bioinformatics projects reproducible.

Full reproducibility requires the ability to recreate the system that was originally used to generate the results.

This can, to a large extent, be accomplished by using a Conda environment file to make an environment with specific versions of the packages that are needed in the project.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Reproducible Research

Conda environments are useful when making bioinformatics projects reproducible.

Full reproducibility requires the ability to recreate the system that was originally used to generate the results.

This can, to a large extent, be accomplished by using a Conda environment file to make an environment with specific versions of the packages that are needed in the project.

This environment file can then be shared with others users to reproduce your analysis environment containing software with the same version number.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Creating an environment file

Conda uses YAML (“YAML Ain’t Markup Language”) for writing its environment files.

YAML is a human-readable language that is commonly used for configuration files and that that uses Python-style indentation to indicate nesting.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Creating an environment file

Conda uses YAML (“YAML Ain’t Markup Language”) for writing its environment files.

YAML is a human-readable language that is commonly used for configuration files and that that uses Python-style indentation to indicate nesting.

```
name: rnaseq-env
```

a key-value pair
Key name
Value `rnaseq-env`

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Creating an environment file

Conda uses YAML (“YAML Ain’t Markup Language”) for writing its environment files.

YAML is a human-readable language that is commonly used for configuration files and that that uses Python-style indentation to indicate nesting.

```
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
```

The second item is an array channels with 2 elements, each denoted by an opening dash **-**.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Creating an environment file

Conda uses YAML ("YAML Ain't Markup Language") for writing its environment files.

YAML is a human-readable language that is commonly used for configuration files and that that uses Python-style indentation to indicate nesting.

```yaml
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
```

I indented the elements in channels with two spaces. Indentation is how YAML denotes nesting.

The number of spaces can vary from file to file, but tabs are not allowed

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Creating an environment file

Creating your project’s Conda environment from a single environment file is a Conda “best practice”.

Not only do you have a file to share with collaborators but you also have a file that can be placed under version control (Git) to further enhancing the reproducibility of your research project and workflow.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: An environment file

```yaml
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

`environment.yml`

Sharing Environments: An environment file

The first line specifies a default name `rnaseq-env` for the environment

```
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

`environment.yml`

Sharing Environments: An environment file

The environment would be installed inside the conda environment directory e.g. `~/miniconda3/envs/` directory, unless we specified a different path using `conda create` command line option `--prefix` or `-p`.

```yaml
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

Sharing Environments: An environment file

The second line specifies a list of channels, listed in priority order, that packages may need to be installed from.

```
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

`environment.yml`

Sharing Environments: An environment file

The dependencies lists the most current and mutually compatible versions of the listed packages (including all required dependencies) to download.

```
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

`environment.yml`

Sharing Environments: An environment file

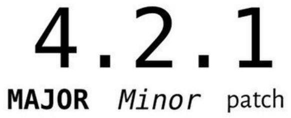
Explicit versions numbers for all packages should be preferred a better environment file would be the following.

```
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

`environment.yml`

Sharing Environments: An environment file

Note that we are only specifying the major and minor version numbers and not the patch or build numbers.



```yaml
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

Defining the version number by fixing only the major and minor version numbers while allowing the patch version number to vary allows us to use our environment file to update our environment to get any bug fixes whilst still maintaining significant consistency of our Conda environment across updates.

`environment.yml`

Sharing Environments: An environment file

```
$ cd ~/
```

```
$ mkdir rnaseq-project-2
```

```
$ cd rnaseq-project-2
```

Sharing Environments: An environment file

Once your project folder is created, create an `environment.yml` file using your favourite editor for instance `nano`.

```yaml
name: rnaseq-env
channels:
  - conda-forge
  - bioconda
dependencies:
  - salmon=1.5
  - fastqc=0.11
  - multiqc=1.11
```

Sharing Environments: Create a new conda environment:

Create a new conda environment: `conda env create`

```
$ conda env create --prefix ./env --file environment.yml
```

```
$ conda activate ./env
```

Sharing Environments: Create a new conda environment:

We can automatically generate the contents of an environment file using the `conda env export` command.

```
$ conda env export --name basic-rnaseq-env
```

Sharing Environments: Export a conda environment:

To export this list into an `environment.yml` file, you can use `--file` option to directly save the resulting YAML environment into a file.

```
$ conda env export --name basic-rnaseq-env --file environment.yml
```

Make sure you do not have any other `environment.yml` file from before in the same directory when running the above command.

Sharing Environments: Export a conda environment: history

This exported environment file will however not *consistently* produce environments that are reproducible across Mac OS, Windows, and Linux. The reason is, that it may include operating system specific low-level packages which cannot be used by other operating systems.

If you need an environment file that can produce environments that are reproducible across Mac OS, Windows, and Linux, then you are better off just including those packages into the environment file that your have specifically installed using the `--from-history` option.

```
$ conda env export --name basic-rnaseq-env --from-history --file environment.yml
```

Exercise: Create a new environment from a YAML file.

Create a new project directory `rnaseq-project-3` and then create a new `environment.yml` file inside your project directory with the following contents.

```yaml
name: rnaseq-project3-env

channels:

  - conda-forge

  - bioconda

dependencies:

  - salmon=1.5

  - fastqc=0.11

  - multiqc=1.11
```

- Now use this file to create a new Conda environment.
- Where is this new environment created?

.

Sharing Environments: Updating an environment:

You are unlikely to know ahead of time which packages (and version numbers!) you will need to use for your research project.

Sharing Environments: Updating an environment:

You are unlikely to know ahead of time which packages (and version numbers!) you will need to use for your research project. For example it may be the case that

- one of your core dependencies just released a new version (dependency version number update).

Sharing Environments: Updating an environment:

You are unlikely to know ahead of time which packages (and version numbers!) you will need to use for your research project. For example it may be the case that

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).

Sharing Environments: Updating an environment:

You are unlikely to know ahead of time which packages (and version numbers!) you will need to use for your research project. For example it may be the case that

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).
- you have found a better visualization package and no longer need to old visualization package (add new dependency and remove old dependency).

Sharing Environments: Updating an environment:

You are unlikely to know ahead of time which packages (and version numbers!) you will need to use for your research project. For example it may be the case that

- one of your core dependencies just released a new version (dependency version number update).
- you need an additional package for data analysis (add a new dependency).
- you have found a better visualization package and no longer need to old visualization package (add new dependency and remove old dependency).

Sharing Environments: Updating an environment

If any of these occurs during the course of your research project, all you need to do is update the contents of your `environment.yml` file accordingly and then run the following command.

```
$ cd ~/

$ cd rnaseq-project-2

$ conda env update --prefix ./env --file environment.yml
```

Note this doesn't remove packages that you remove from `environment.yml`

Exercise: Update environment from environment.yml

1. Update the environment file `environment.yml` from the previous exercise, **rnaseq-project-3**,
   a. by adding the package kallisto=0.46 and
   b. removing the salmon package.

2. Then rebuild the environment `conda env update`

Is the salmon package still present?

Sharing Environments: Rebuilding a Conda environment

When working with environment.yml files it is often just as easy to rebuild the Conda environment from scratch whenever you need to add or remove dependencies.

To rebuild a Conda environment from scratch you can pass the `--force` option to the `conda env create` command.

This will remove any existing environment directory before rebuilding it using the provided environment file.

```
$ cd ~/rnaseq-project-2
$ conda env create --prefix ./env --file environment.yml --force
```

Sharing Environments: Restoring an environment:

Conda keeps a history of all the changes made to your environment, so you can easily “roll back” to a previous version.

To list the history of each change to the current environment:

```
$ conda activate basic-rnaseq-env
```

```
$ conda list --revisions
```

Sharing Environments: Restoring an environment:

To restore environment to a previous revision:

```
$ conda install --revision=REVNUM or conda install --rev REVNUM.
```

```
$ conda install --revision=1
```

Exercise: List revisions.

1. Activate the environment inside the `rnaseq-project-3` and list the revisions
2. Install revision 0

Sharing Environments: Key Points

- Sharing Conda environments with other researchers facilitates the reproducibility of your research.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Sharing Environments: Key Points

- Sharing Conda environments with other researchers facilitates the reproducibility of your research.
- Conda environment files ,environment.yml, describes your project’s software environment.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/04-sharing-environments/index.html

Configuring Conda

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration

Configuring Conda: Questions

- How can I configure conda ?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration

Configuring Conda: Questions

- How can I configure conda ?
- How can I see conda’s configuration values?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration

Configuring Conda: Questions

- How can I configure conda ?
- How can I see conda’s configuration values?
- How can I modify conda’s configuration settings?

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration

Configuration

Conda has a number of configuration setting which control how it performs.

To display and control these setting we can use the conda config subcommand.

To display all configuration settings run the config --show subcommand :

```
$ conda config --show
```

Configuration

As you can see conda supports a large number of configuration options.

To show a single setting add the setting name after the `conda config --show` command.

```
$ conda config --show channels
```

Configuration

By default conda only searches the `defaults` channel, this is why we had to include `conda-forge` and `bioconda` channels via the command line option `--channel` in the previous episode.

To get more information about an individual conda setting and its’ possible values run `conda config --describe <option>`.

```
$ conda config --describe channels
```

Configuration: .condarc

A user’s conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

Configuration: .condarc

A user’s conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

- Where conda looks for packages `channels`.

Configuration: .condarc

A user’s conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

- Where conda looks for packages `channels`.
- Where conda lists known environments `envs_dirs`.

Configuration: .condarc

A user’s conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

- Where conda looks for packages `channels`.
- Where conda lists known environments `envs_dirs`.
- Whether to update the Bash prompt with the currently activated environment name `env_prompt`.

Configuration: .condarc

A user’s conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

- Where conda looks for packages `channels`.
- Where conda lists known environments `envs_dirs`.
- Whether to update the Bash prompt with the currently activated environment name `env_prompt`.
- What default packages or features to include in new environments `create_default_packages`.

Configuration: .condarc

A user's conda setting are store in the runtime configuration configuration file, `.condarc`. This file allows users to configure various aspects of conda including:

- Where conda looks for packages `channels`.
- Where conda lists known environments `envs_dirs`.
- Whether to update the Bash prompt with the currently activated environment name `env_prompt`.
- What default packages or features to include in new environments `create_default_packages`.

Configuration: .condarc

Like the environment file the `.condarc` configuration file follows a simple YAML syntax

The `.condarc file` is not included by default, but it is automatically created in your home directory the first time you run the `conda config` command.

Configuration: Creating or modify .condarc

To create or modify a `.condarc file`, enter the `conda config` command and use the modifier options

- `--add`,
- `--set`,
- `--append` ,
- `--prepend`
- `--remove`

followed by the configuration key and a value .

```
conda config <modifier> <KEY> <VALUE>
```

Configuration: config --add

Adding conda-forge

```
$ conda config --add channels conda-forge
```

Configuration: config --add

Adding conda-forge

```
$ conda config --add channels conda-forge
```

This would add the `conda-forge` channel to the top of the channel list.

```
$ conda config --show channels
```

Configuration: config --append

We can use the `conda config` modifier `--append` to add `conda-forge` to the end of the channel list,

```
$ conda config --append channels conda-forge
```

Giving it the lowest priority.

```
$ conda config --show channels
```

Configuration: config --prepend

To move a channel to the highest priority use the `conda config` `--prepend` modifier.

```
$ conda config --prepend channels conda-forge
```

Giving it the highest priority.

```
$ conda config --show channels
```

Note: It is generally best to have `conda-forge` as the highest priority channel as this will usually have the most up-to-date packages.

Exercise: Adding the channels bioconda and conda-forge to .condarc.

Add the bioconda and conda-forge channels to your **`.condarc file`**.

Give conda-forge the highest priority.

Configuration: Setting configuration settings --describe

If our configuration setting has a single boolean or string value we can use `conda config --set` to set it.

For example, In a previous episode we set the command line prompt setting for conda using `env_prompt`.

```
$ conda config --describe env_prompt
```

The `env_prompt` setting takes a value of either `'{prefix}'`, `'{name}'`, and `'{default_env}'`.

Configuration:: Setting configuration settings --set

To set the `env_prompt` to the default value `'({default_env})'` we can run:.

```
$ conda config --set env_prompt '({default_env})'
```

Note:: You need to deactivate then reactivate the environment for the changes in env prompt to take effect.

Configuration: Setting configuration settings --set

To set the `env_prompt` to the default value `'({default_env})'` we can run:.

```
$ conda config --set env_prompt '({default_env})'
```

To change it back to the just the environment name, we can run.

```
$ conda config --set env_prompt '({name})'
```

Exercise: Set conda channel_priority

1. Use the conda `config --describe` to investigate the setting **channel_priority**.
2. Set the **channel_priority** so that packages in lower priority channels are not considered if a package with the same name appears in a higher priority channel.
3. Why would you want to do change this setting?

Configuration: Editing the .condarc file manually

You can also use a text editor such as nano to directly edit the `.condarc`.

To show the location and contents of your `.condarc` file you can use

```
$ conda config --show-sources
```

Note: If the .condarc file is in the root environment, it will override any in the home directory

Exercise: Locate and view the `.condarc`

- Locate your .condarc file.
- Using your favourite text editor look at the `.condarc`.

Configuration: Help

As with all conda commands you can use the `--help` option to get help.

For example, for a complete list of `conda config` commands run

```
$ conda config --help
```

https://conda.io/projects/conda/en/latest/commands/config.html

Configuring Conda: Key Points

- The `.condarc` is an optional configuration file that stores custom conda setting.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration/index.html

Configuring Conda: Key Points

- The `.condarc` is an optional configuration file that stores custom conda setting.

- You can use the conda config subcommand to add, set or remove configuration setting in the `.condarc` file.

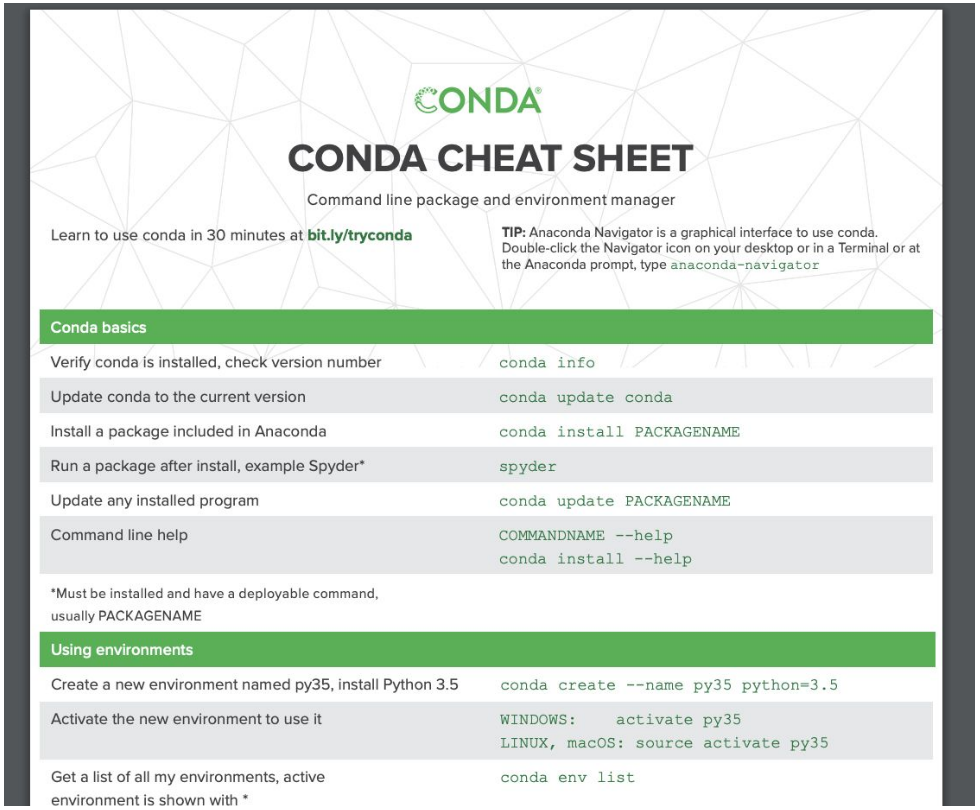
https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration/index.html

Configuring Conda: Key Points

- The `.condarc` is an optional configuration file that stores custom conda setting.
- You can use the conda config subcommand to add, set or remove configuration setting in the `.condarc` file.
- You can also edit the contents of the .condarc file directly using a text editor.

https://edcarp.github.io/introduction-to-conda-for-data-scientists/05-configuration/index.html

Conda: Cheat Sheet



CONDA

CONDA CHEAT SHEET

Command line package and environment manager

Learn to use conda in 30 minutes at **bit.ly/tryconda**

TIP: Anaconda Navigator is a graphical interface to use conda. Double-click the Navigator icon on your desktop or in a Terminal or at the Anaconda prompt, type `anaconda-navigator`

Conda basics	
Verify conda is installed, check version number	`conda info`
Update conda to the current version	`conda update conda`
Install a package included in Anaconda	`conda install PACKAGENAME`
Run a package after install, example Spyder*	`spyder`
Update any installed program	`conda update PACKAGENAME`

Command line help	`COMMANDNAME --help`
	`conda install --help`

*Must be installed and have a deployable command, usually PACKAGENAME

Using environments	
Create a new environment named py35, install Python 3.5	`conda create --name py35 python=3.5`

Activate the new environment to use it	`WINDOWS:     activate py35`
	`LINUX, macOS: source activate py35`

Get a list of all my environments, active environment is shown with *	`conda env list`

https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf

Please fill out the survey