Intro to Conda: Environment management and design

Zed Chen 2025/07/10 Workshop 1

# Acknowledgement

Substantial credit to Al Ivens at University of Edinburgh

The code templates are on /mnt/shared/projects/rbge/zedchen/workshop/Conda\_Env\_01

# Packages

When we login to Crop Diversity and start a Python session ("bubble"), can we do everything we want? No, our initial bubble just provides some basic functionality, the rest of all the cool things we might want are not immediately available, we need to get them "into our bubble".  
  
We accessed modules using **import** commands such as  
import os, shutil, subprocess  
which enabled us to do many file manipulation tasks that we couldn’t do when we had just started the Python3 session.  
  
Modules, packages, libraries...?

* **Module**: a collection of functions and variables, e.g. in a script
* **Package**: a collection of modules with an **init**.py file (can be empty), e.g. a directory with scripts
* **Library**: a collection of packages with related functionality

The phrases *library* and *package* are often used interchangeably...

Modules (function) < packages (script) < library (collection of scripts/directory)

# Dependencies

When we install **pandas (for manipulating dataframe in python)** or similar, many packages do not just do everything on their own. Instead, they **depend** on other packages for their functionality.  
  
For example, the [scipy package](https://docs.scipy.org/doc/scipy/) is used for numerical routines, but also makes use of other packages, such as **numpy** and **matplotlib** and so on. So we say that **numpy** and **matplotlib** are **dependencies** of **scipy**: it won't work without them...!  
  
Many packages are continually being developed, so we can end up with loads of different **versions** of packages, and sometimes, functionalities have been added or removed...oops! other packages, which are dependent on this one, suddenly don't work anymore....  
  
Using the **scipy** example again, we know it depends on **numpy** and **matplotlib**, but also that it depends on numpy version >= 1.6 and matplotlib version >= 1.1.  
**numpy** version 1.5 in this case would not be sufficient.

# **Potential challenges with packages**

There comes a time when one version of a package or the programming language is not enough anymore. An older tool depends on an older version of the programming language (e.g. Python 3.6), but many of the newer ones depend on a newer version (e.g. Python 3.10).

A screen shot of a computer

AI-generated content may be incorrect.  
  
We could have another computer or virtual machine (VM) to run the other version of the programming language, but this is not very efficient/sensible, since we may want to use the tools together in a workflow later on. Tadaa! **Environments** are one solution to the problem.  
  
We can install packages in isolated environments: these act as different "bubbles" that we can choose to use whenever we need them!

# **Environment**

Imagine each environment as a greenhouse you setup for various plants (packages) you want to grow.

It allows you to put provide the desirable space for compatible packages, and most importantly, when the tools you want to use are not compatible, each can operate in a separate environment all found on the same server!

It is like a big greenhouse with various rooms for different habitat requirements.

So don’t install everything into the (base) environment on Crop Diversity. Sooner or later an incompatibility will occur. Plus you probably don’t want to upset the most basic functionalities.



# **Why do we need Environment management**

An environment management system attempts to solve compatibility issues, such as:

* an application we need for a research project requires different versions of our base programming language or different versions of various third-party packages from the versions that we are currently using.
* an application we developed as part of a previous research project that worked fine on our system six months ago now no longer works.
* codes that we have written for a joint research project works on our machine but not on our collaborators’ machines.
* an application that we are developing on our local machine doesn’t provide the same results when run on our remote cluster (well we probably almost always work on a remote cluster:)

# **What can we achieve using Environment**

An environment management system enables us to set up a new, project-specific, software environment containing specific Python versions (for example), as well as the versions of additional packages and required dependencies that are all mutually compatible.

* environment management systems help resolve dependency issues by allowing we to use different versions of a package for different projects.
* make our projects self-contained and reproducible by capturing all package dependencies in a single requirements file.
* allow us to install packages on a host on which we do not have admin privileges.

Conda is not the only way; Python for example has many more ways of working with environments (e.g. [virtualenv](https://virtualenv.pypa.io/en/latest/), [pipenv](https://pipenv.pypa.io/en/latest/), [venv](https://docs.python.org/3/library/venv.html), [pyenv](https://github.com/pyenv/pyenv)...)

But we got Conda on Crop Diversity and it’s free to install, so…

# **Conda**

From the [official Conda documentation](https://conda.io/projects/conda/en/latest/index.html).

*Conda is an open-source package management system and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. Conda easily creates, saves, loads, and switches between environments on our local computer. It was created for Python programs but it can package and distribute software for any language.  
Conda as a package manager helps us find and install packages. If we need a package that requires a different version of Python, we do not need to switch to a different environment manager because conda is also an environment manager. With just a few commands, we can set up a totally separate environment to run that different version of Python, while continuing to run our usual version of Python in our normal environment.*

This is what we will do today.

# **Crop Diversity**

Conda **DOES** has a default environment called **base** that includes a Python installation and some core system libraries and dependencies of Conda

* Once log in to Crop Diversity, you automatically **activate** the **<base>** environment by default.
* This environment is also automatically **activated** when you start an interactive job using **srun** (conda deactivate to go back to the previous env)

However, we should always installing packages for projects into an env other than <base>: it's just good practice, and minimises the risk of compromising our **base** installation.

And don’t install everything into just one env!

The point of setting up environment is that we don’t need to cram all the packages into one greenhouse.

Don’t know where to put these info…

BTW we have some variables already setup:

echo $HOME

echo $SCRATCH

echo $USER

# **The Conda Basics**

## What is a Conda environment

A [Conda environment](https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html) is a directory that contains details of which packages, and what version numbers, are required; different environments, different packages. If you change one environment, our other environments are not affected. we can easily activate or deactivate environments, which is how we switch between them (see below!).

## starting with conda

#presumably you are on Crop Diversity now

# What version of conda is running?

conda --version

# How do we get help?

conda --help

What do you see?

#What environments are available?

conda env list

#where is the base env?

## Make an environment

Say we want to create a new environment for a Python project (or anything else), we just need the **conda create** command.  
  
It is a good idea to give our environment a meaningful name...  
  
conda create --name python3-env python

#what version of python is installed?

#What packages are installed?

The command above will create a new Conda environment called "python3-env" and install, by default, the most recent version of Python. If we had wanted a specific version of Python, we could have issued a different command, e.g.  
**conda create --name python3.8.6-env python=3.8.6**

Where we can, it is a good idea to specify the version number of each package that we install into an environment; search to see what versions are available using the **conda search**, for example:  
  
conda search scikit-learn

#Goodness how many versions are there?

conda search scikit-learn|wc -l

we can specify a lot of packages and their versions at once:

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

## How are packages installed?

A diagram of a process flow

AI-generated content may be incorrect.

## Modify an Env

It’s okay if we don’t install all the packages when creating the env, which is more often the case. Hence, we can add more packages are the project unfolds itself.

We can rename a conda environment with the **conda rename** command. conda rename supports renaming our current environment, or any of our existing environments.  
  
conda rename -n **basic-scipy-env** **scipy-env**

**We can keep installing new packages into our env:**

conda install -c bioconda **samtools** -y --name **scipy-env**

-y: this is very useful when recording your commands into a code, because it will run automatically without stopping to ask your permission to proceed

#task: try to install igv without -y

**We can keep installing new packages into our env after we activate the env:**

conda activate $env\_name

conda install -c bioconda bamtools -y

**# What packages do we have in the basic-scipy-env environment?**

**conda list --name scipy-env**

**# what happens if you just do conda list without specifying an env?**

**# what happens if you try to install a package that’s already present in the env?**

## Freezing installed packages

To prevent existing packages from being updated when using the **conda install** command, we can use the **--freeze-installed** option. This may force Conda to install older versions of the requested packages in order to maintain compatibility with previously installed packages. Using the **--freeze-installed** option does not prevent additional dependency packages from being installed.

## Remove a package from an environment

To remove a package from an environment we can run the **uninstall** command.  
For example to remove that **numba** package from the basic-scipy-env environment we can do:

conda uninstall samtools --name scipy-env

## Where to store the environments

Naturally, we might want to keep our environments and packages in our home space on Crop Diversity, but as our **${HOME}** isn't very large, it tends to fill up and then **bad things happen**.

A screenshot of a computer

AI-generated content may be incorrect.  
  
**There are two (three?) main alternatives:**

1. use scratch space, as it is huge....BUT files etc. get deleted sooner or later: bad option
2. use projects: some limits but should be enough (where all the packages will be stored too)
3. use GitHub: don’t ever store any of the packages there (i.e. don’t upload the entire env directory)! But it is a good way to make environments sharable (discussed later in the ‘SAVE THE ENV’ section)

## Where do the environments live?

conda config --show envs\_dirs

**# What is in my envs folder?**

ls -1 ${THE DIRECTORY PATH}

**# How much "stuff" is in my envs folder?**

# Use the **du** command to find out (takes a while...)

du -hc --max-depth 1 ${THE DIRECTORY PATH}

#are they in places that we want them to be?

## Config the env path (optional)

I hope by now we have some idea for where to keep the env directories: somewhere with enough space and not wiped every 6 months:

We can keep all environments in one place:

# Tell **conda** where we want our environment details to be kept:

conda config --add envs\_dirs /home/${USER}/projects/rbge/$YOUR\_DIRECTORY/env/  
  
# Tell **conda** where we want our packages to be kept:

conda config --add pkgs\_dirs /home/${USER}/projects/rbge/$YOUR\_DIRECTORY/pkg

# Check what settings we have now:  
# saved in a file called **.condarc**

**#what’s in .condarc?**

cat ${HOME}/.condarc

## Deleting entire environments

Occasionally, we need to delete an entire environment, luckily it is easy.

conda create --name to\_be\_deleted

conda remove --name to\_be\_deleted –all

#a different method?

or of course you can rm -rf the directory once you know where it is, but if you do so, can you still see the env in conda env list?

## the configuration file—— .condarc

yes there’s actually a dot there, not a typo. The dot keeps the file ‘hidden’ from ls, but you can force it to appear with ls -al

A user’s conda settings are stored 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**.

**.condarc** configuration file follows a simple YAML syntax (will see this format later)  
  
The **.condarc file** is not included by default, but it is automatically created in our home directory the first time we run the **conda config** command.

## Creating or modifying .condarc

To create or modify a **.condarc file**, enter the **conda config** command and use the modifier options **--add**, **--set**, **--append** , **--prepend** or **--remove** followed by the configuration key and a value .  
  
# Add **conda-forge** to the top of the list of **channels/remove it**

conda config --add channels conda-forge

conda config --show channels

#what if you try to add it again?

conda config --remove channels conda-forge

You can also do this manually, but I am not talking about it this time

## A fun configuration to do

ls ~ -al

#what are the hidden files starting with . ?

Find the .bashrc file:

nano ~/ .bashrc

#What do you see here?

Add this line to the end of the file

BETTER CHANGE $USER TO YOUR ACTUAL USER NAME

alias jobs='squeue|grep $USER'

now you can type ‘jobs’ to view all the jobs your have submitted on slurm/crop diversity

# **Don’t forget Tea break after an hour of work**

# **A Toby Hack: try to run graphic tools on servers**

#task 1

Make a new environment on Crop Diversity

Install aliview

#can you use aliview there?

#task 2

Start a session on Toby

Try to install aliview. Can you do that?

#However, we can install packages into an environment

#Can you create an environment on Toby?

#Can you get conda?

#Check out the new guide and apply your conda knowledge:

<https://github.com/Zedthedrifter/Workshops-and-Guides/blob/main/Installing%20Packages%20Freely%20on%20Toby.pdf>

how to get file from CropDiversity to Toby?

#Task

Try to scp gene\_1.fasta to your current directory and visualize it with aliview. Path:

/mnt/shared/projects/rbge/zedchen/workshop/Conda\_Env\_01/gene\_1.fasta

A template for scp from Crop Diversity in Toby:

* + scp -r $USER@gruffalo.cropdiversity.ac.uk:$FULL\_PATH\_TO\_DIRECTORY\_OR\_FILE $PATH\_TO\_DESTINATION\_ON\_TOBY
  + you can type . for the $PATH\_TO\_DESTINATION\_ON\_TOBY bit, which means ‘here’

it will ask for your passphrase for the Crop Diversity Login Key

# **Environment Hacks**

## An alternative way to record the environment setup

We’ll see the more standard method of saving environment later, but you can also simply record the set up of an environment as a script and run the script to recreate the env: template\_single.sh

This is not a job that requires array, so you can either submit it as sbatch, or run it directly despite the sbatch headlines:

./ template\_single.sh

Might need to chmod 700 template\_single.sh first to make it executable, but that’s a minor issue.

## Organizing your script: Order and Compartmentalization

* format: function name\_of\_the\_function { content of the function }
* Group installation of related packages into one function
* Install the installation tools first: e.g. pip
* Install everything depending on that afterward
* Write a function for each package that takes special treatment, e.g. function install\_easy353
* Wrap up with a ‘main’ function to control the steps of executing each sub function

## Running a script in an interactive session on Crop Diversity

#what’s the memory limit of when you are at ‘home’ on Crop Diversity?

[Frequently Asked Questions — Crop Diversity HPC Help documentation](https://help.cropdiversity.ac.uk/faq.html)

Note that gruffalo has a 6 GB memory limit for your shell to avoid processes using up the node’s memory.

That’s not much isn’t it

Thankfully we got srun:

srun --partition=short --cpus-per-task=8 --mem=16G --pty bash

#can you request more memory?

## A Python package isn’t available on any Conda channel! What should I do?

If a Python package that we need isn’t available on any Conda channel, then we can use the default Python package manager [**Pip**](https://pip.pypa.io/en/stable/) to install this package from [**PyPI**](https://pypi.org/).  
  
If we find we need to install a Python package that is only available via Pip, then we first install pip into our Conda environment and THEN use that pip to install the desired package.  
  
Many of the common pitfalls of using Conda and Pip together can be avoided by adopting the following practices.

* always explicitly install pip in every Python-based Conda environment.
* always be sure our desired environment is active before installing anything using pip.

Check out template.sh for how to do this

## What if pip cannot help either?

A lot of the packages are available on GitHub and often include commands from installing directly without an environment via git clone:

<https://github.com/plant720/Easy353>

An example: the function install\_tree in template\_single.sh

#what could be a good place to keep the source scripts of the newly installed package (okay you’d better consider that before installing it)?

#will the program know where to find the easy353 scripts?

Configuration

#Should I config locally or globally?

## Tasks

Create a new environment, modify the code to install at least

* Biopython (pip),
* easy353 (GitHub),
* and iqtree & figtree & mafft (default method)
* are there other packages you want to try installing for your research?
* If you haven’t activate an env, what will happen if you did not specify env name in conda install?
* Keep this env. We will need it for the last section

# **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.  
  
The environment file can then be shared with others to reproduce the analysis environment containing software with the same version numbers.



## Creating an environment file

Conda uses [YAML](https://en.wikipedia.org/wiki/YAML) ("YAML Ain’t Markup Language", or "Yet Another Markup Language" or ...) for writing its environment files.  
  
[YAML](https://docs.ansible.com/YAMLSyntax.html) is a human-readable language that is commonly used for configuration files and that that uses Python-style indentation to indicate nesting.  
  
Creating a project’s Conda environment from a single environment file is a Conda "best practice": we have a file to share with collaborators AND have a file that can be version controlled: an ideal research project administration model!

Within an env:

conda env export > environment.yml

If you use conda env export, it will export all of those packages. However, if you use conda env export --from-history, it will only export those you specifically chose:

conda env export --from-history > environment.yml

How many things are in our environment?

How many things are listed in the yml file?

## Restore an environment

Now let’s try to remove the environment and restore it

* conda env remove --name $ENV\_name
* conda env create --name pansyn --file environment.yaml
* Let’s have a look at environment.yaml: what are listed and what are missing?
* Can you use the packages you installed before?
  + samtools?
  + Pip?
  + Things installed with pip?
  + Easy353 ?
    - Be careful with the ‘hacking’ way of installing env
    - So you probably understand why we need to keep a copy of the script that generates the env in the first place…
    - To reconcile a hack, another hack—thankfully not a thousand—is needed

# How do we organize the work directories?

Use my own work directory as an example:

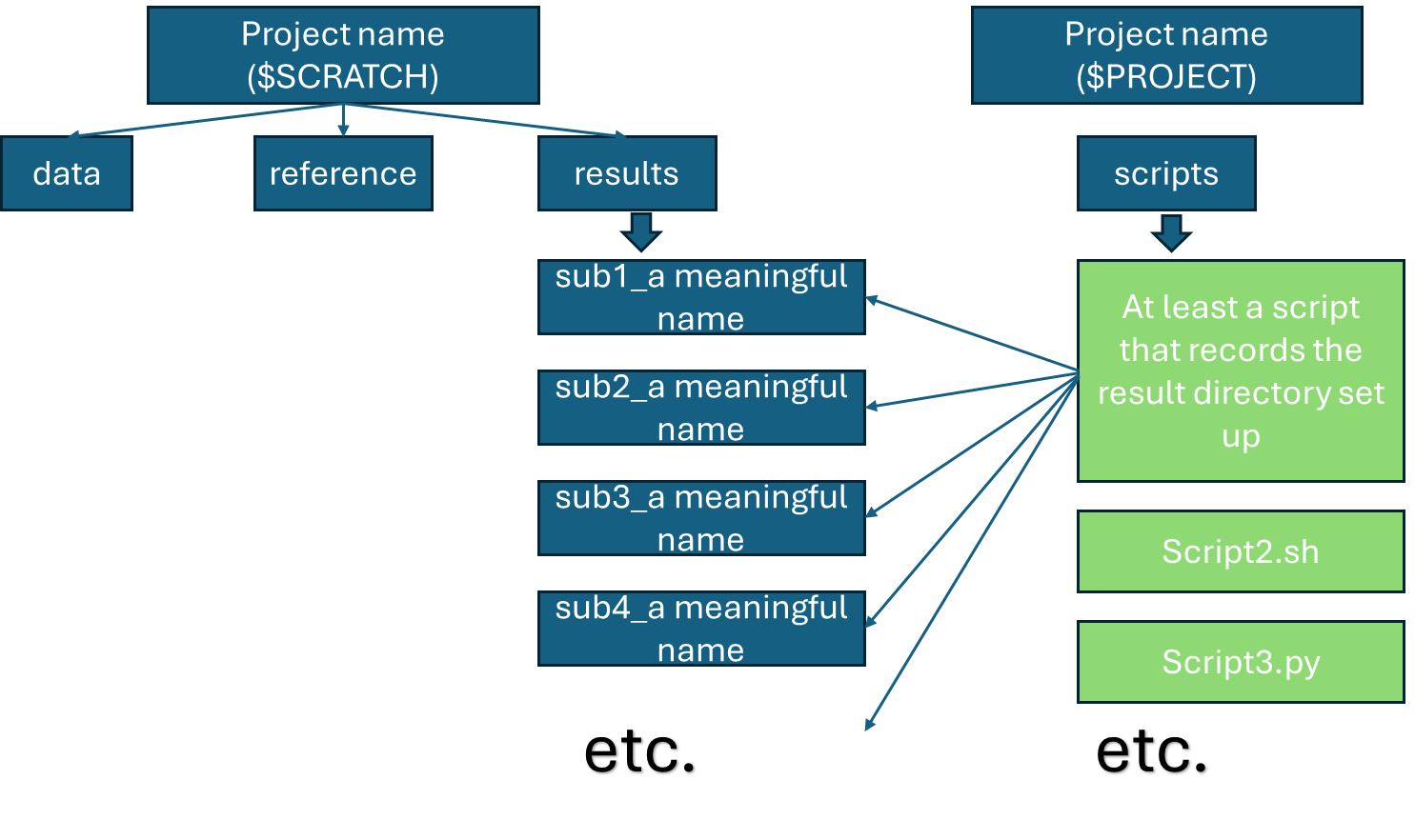
Do we actually need to be in the same directory as the results?

And please don’t put results from different steps in the same directory…

And separate results and data…

How can we easily display the ‘layers’ of results that reflect the process of our analysis?

A simple but organized layout improves reusability of your SOP and codes:



## Where do we keep the scripts?

#can they stay safe forever on scratch?

#actually, you can keep all your scripts in projects, run them, and keep the intermediate results in scratch

One way or the other, better to keep them in one directory… they got not reason to be mixed with results anyway

#and it’s probably time to use backup the scripts onto GitHub…