# A brief introduction to the use of *conda* on the Trantor cluster

*Conda* is both an environment management system (similarly to "*virtualenv*") and a package management system (similarly to "*pip*"). While it focuses on Python, it can also be used to install libraries for other programming languages as well as software binaries.

**Once created, *conda* environments can be used both on head and compute nodes. It is important to note, however, that environment creation and manipulation is only possible on the head nodes**.

## Initializing your shell for conda

Before using *conda*, it is necessary to properly configure your shell with the command:

```
/cluster/shared/software/miniconda3/bin/conda init
```

For changes to take effect, you have to logout and login again. After logging in, you'll see the prefix `(base)` at the left of the shell prompt (e.g. `(base) [user@hostname ~]$` ). The name (or the path) inside the parentheses shows the currently active environment.

## Preconfigured environments

At the time of writing, the following preconfigured environments are available on the cluster:

### "base"

**Path**: /cluster/shared/software/miniconda3/
**Description**: Base python 3 environment.

### "callisto"

**Path**: /cluster/shared/software/conda_envs/callisto/
**Description**:
A Python environment for scientific computing. Includes the following packages:

- Python 3.8
- IPython
- Numpy
- Pandas
- Scipy
- Scikit-learn
- Seaborn
- Matplotlib

“cosmo3.8”

**Path**: /cluster/shared/software/conda_envs/cosmo3.8/
**Description**: Python 3.8 environment containing specific packages for the needs of the Cosmology group.

## Activating and deactivating a conda environment

You can activate an environment by means of the following command:

```
conda activate <environment name or path>
```

This command requires as argument the short name or the full path of the environment. Please note that the short name can only be used for “*base*” or for your custom environments (discussed in the following). In the case of preconfigured environments (such as “*callisto*”), you have to provide their full path.

Examples:

```
conda activate base
conda activate my_custom_env
conda activate /cluster/shared/software/conda_envs/callisto
```

After that, you can use the `conda list` command to get a detailed list of all the packages installed in the currently active environment.

Finally, the environment can be deactivated with `conda deactivate`.

## Activating a conda environment in PBS Jobs

This section explains how to set up the job script in order to use *conda* environments in your (non-interactive) PBS jobs.

> **Note**
>
> If you are not familiar with PBS, be sure to read the following introductory guide to PBS available on the HPC Center’s website:
> https://hpccenter.sns.it/page/wiki/pages/Submitting_Inspecting_and_Cancelling_PBS_Jobs.html

To activate a *conda* environment within a PBS job, executing `conda activate` as part of the job script is (usually) not enough!
In the case of ***bash*** scripts, this is due to the fact that `conda init` inserts the required configuration within your personal *.bashrc* file. However, regular jobs run in *non-interactive shells*, which don't process *.bashrc* at startup. As a consequence, before activating the desired environment, you need to set up your shell for *conda* by means of the following command:

```
eval "$(/cluster/shared/software/miniconda3/bin/conda shell.bash hook)"
```

Here is an example:

```
#!/bin/bash
#PBS -l select=1:ncpus=2
#PBS -q q07helicon
#PBS -N my_job
eval "$(/cluster/shared/software/miniconda3/bin/conda shell.bash hook)"
conda activate /cluster/shared/software/conda_envs/callisto
python3 my_python_script.py
...
```

## User-defined environments

Although the preconfigured environments can't be modified by regular users, you can create custom virtual environments according to your specific needs. Such environments will be stored in your home directory and will be available both on head and compute nodes.

The following subsections describe the steps required for creating, modifying and removing a custom virtual environment with *conda*.

### [Optional] Setting conda-forge as primary channel

It is recommended to set "*conda-forge*" as the primary channel for searching and downloading packages, since it provides considerably more updated software with respect to the default channels. To this end, follow these steps:

1. Change the working directory to your home:

   ```
   cd ~
   ```

2. Disable any previously activated environment (if any):

   ```
   conda deactivate
   ```

   You may need to run this command multiple times, until the prefix at the left of the shell prompt disappears.

3. Add conda-forge to the top of the channels list:

   ```
   conda config --add channels conda-forge
   conda config --set channel_priority strict
   ```

**Notes**

- This procedure needs to be executed only once. There is no need to repeat it every time you create or modify an environment.
- Although it is possible to rely on conda-forge only for selected packages (e.g. by using the `--channel` option of the the `conda install` command), this may lead to incompatibilities between the packages downloaded from conda-forge and the

ones retrieved from the default channels.

- Carefully read the "tips and tricks" section of the conda-forge's documentation: https://conda-forge.org/docs/user/tipsandtricks.html
- Setting *conda-forge* as the primary channel is **mandatory** for creating *R* environments that can be used within *JupyterLab*!

## Creating a custom Python-based virtual environment

1. Change the working directory to your home:

   ```
   cd ~
   ```

2. Create the new *conda* environment:

   ```
   conda create -n env_name python=x.y package_2 package_3 … package_N
   ```

   Replace `env_name` with the name of the new environment, and `x.y` with the desired Python version. List at the end of the command any additional packages you want to install (e.g. *numpy*, *matplotlib* etc.).

3. Finally, remember to activate your environment before using it:

   ```
   conda activate env_name
   ```

**Tip**

The following Web page allows to search among the available packages, provided by both official and third-party channels: https://anaconda.org/anaconda/repo

## Creating a custom R-based virtual environment

1. Change the working directory to your home:

   ```
   cd ~
   ```

2. Create the new *conda* environment:

   ```
   conda create -n env_name r-base r-essentials package_2 … package_N
   ```

   Replace `env_name` with the name of the new environment. List at the end of the command any additional packages you want to install.

3. Finally, remember to activate your environment before using it:

   ```
   conda activate env_name
   ```

**Tip**

In the case of *R* libraries, package names usually start with the “r-” prefix.

## Modifying a custom virtual environment

1. Change the working directory to your home:

   ```
   cd ~
   ```

2. Activate the environment you want to modify (if not already active):

   ```
   conda activate env_name
   ```

3. Install or remove packages according to your needs :

   ```
   conda install package_1 package_2 … package_N
   ```

   or

   ```
   conda remove package_1 package_2 … package_N
   ```

**Note**

In order to avoid dependency conflicts, you should install as many packages as possible in one go, preferably when creating the environment. To this end, you may define the required packages in a *yml* file and pass it as an input of the `conda create` command: https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html#creating-an-environment-from-an-environment-yml-file

## Removing a custom virtual environment

1. Change the working directory to your home:

   ```
   cd ~
   ```

2. Disable any previously activated environment (if any):

   ```
   conda deactivate
   ```

   You may need to run this command multiple times, until the prefix at the left of the shell prompt disappears.

3. Delete the environment:

   ```
   conda remove --name env_name --all
   ```

4. Print the list of the local *conda* environments to confirm the removal:

   ```
   conda env list
   ```

### Using pip in a conda environment

Issues may arise when mixing *pip* and *conda* to manage packages, since each of them may be not aware of the changes made by the other. If you really need *pip* in a *conda* environment, you should use *pip* only after installing as many packages as possible with *conda*. After that, you should avoid using *conda* to apply further changes to the environment. For further details, please read the following section of the *conda* user's guide:
https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html#using-pip-in-an-environment

## Further readings

### Conda documentation

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

C*onda* ***channels*** and their management:
https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/channels.html
https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-channels.html

Additional recommendations on using ***pip*** in a *conda* environment:
https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#using-pip-in-an-environment

Cheat-sheet summarizing the main *conda* commands:
https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html

### Conda-forge documentation

https://conda-forge.org/docs/index.html

In particular, **carefully read  the "tips and tricks" section**:
https://conda-forge.org/docs/user/tipsandtricks.html