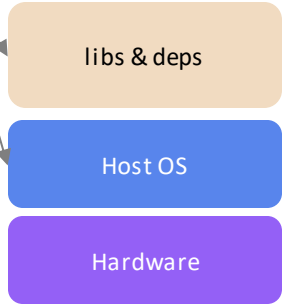
Conda Virtual Environments Introduction

Virtual Environments

What are virtual environments?

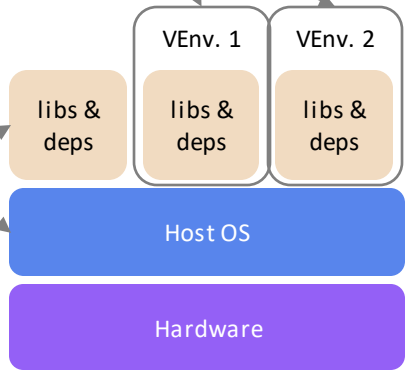
Basically: isolated instances of software (the software is stored in dedicated folders)

Normally, software is installed directly on the operating system



Virtual environments allow you to install software that will only be used while within the environment.

While outside the virtual environment, normal system software is used



Why you should virtual environments

Best practice

For every project, create a corresponding virtual environment

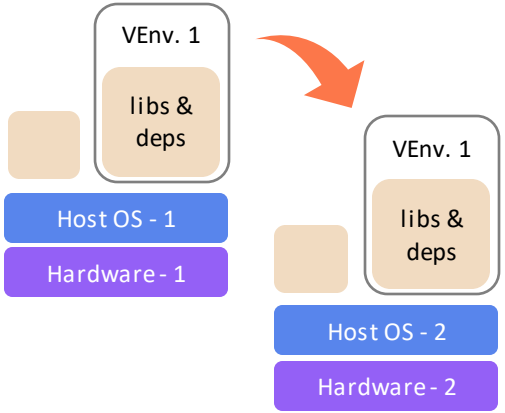
Manage package dependencies:

Often different projects will require different version of the same software.

Using virtual environments allows you the install the version you need for your project.

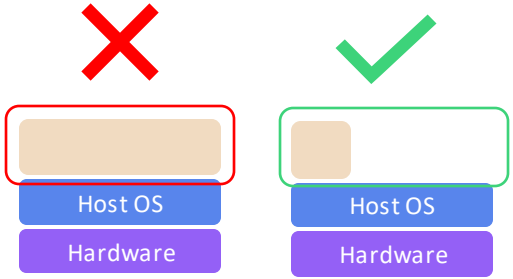
Easier reproducibility:.

By keeping all your software for a specific project within a virtual environment, it is easier to keep track of required software and specific versions. This will make it easier to remake the environment.



Keep OS software lean & clean:.

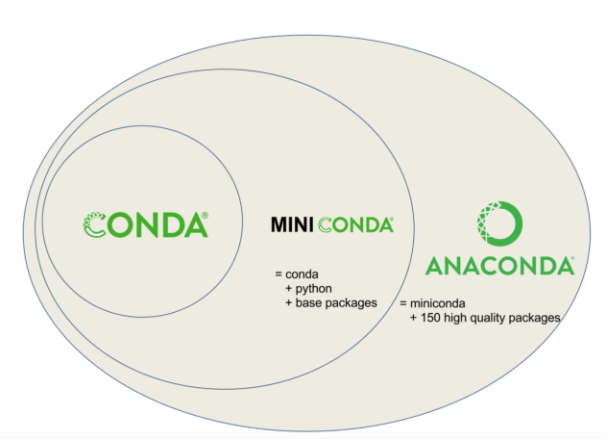
Installing different versions of the same software on your OS can get messy and have unexpected consequences and installing everything directly on your OS can cause the system to get bloated.



Conda & Mamba

What is conda/mamba?

conda is one of the most popular python package & environment management tools for data scientist.



Mamba is a reimplementation of the conda package manager in C++ (meaning it's much faster).
However, it *cannot* to do EVERYTHING conda can.

General rule:

- Use **'mamba' for installing** software (this includes creating new environments)
- Use **'conda' for everything** else

Why you should use conda/mamba

- Can install python packages
- Can install non-python packages
- Can use pip
- Can create virtual environments
- **Checks dependencies**

Setting up conda & mamba

Installing conda & mamba

1. **Install conda** by follow installation instructions here:
https://conda.io/projects/conda/en/latest/user-guide/install/index.html

You will be told asked to download a file and run it

Hints

- I recommend (and use) 'miniconda' over 'anaconda' because it is more lightweight. Anaconda is a much larger file with many pre-installed packages that you may not need.
- If you are using Windows, I recommend installing WSL (Windows Subsystem for Linux) with Ubuntu and installing conda (Linux_ on WSL Ubuntu. Instructions for setting up WSL can be found here: https://ubuntu.com/wsl

2. After installing conda, **install mamba** by running:

```
conda install mamba -n base -c conda-forge
```

More information on mamba can be found here:
https://github.com/mamba-org/mamba

Setup Hints

In general, you should **never install anything in** your **'base' environment** (installing mamba is an exception).

By default, after installing conda, the 'base' environment will always be active when you open a terminal window.

```
File  Edit  View  Search  Terminal  H
(base) jcosme@jane:~$
```

You can change this setting by running the following:

```
conda config --set auto_activate_base false
```

```
File  Edit  View  Search  Terminal  H
jcosme@jane:~$
```

Conda installs from 'channels.'

If you are going to use GPUs, you might also want to add the NVIDIA channel and the Rapids.ai channel.

```
conda config --add channels nvidia
```

```
conda config --add channels rapidsai
```

The most popular channel is 'conda-forge.' It is a good idea to always add this channel to conda.

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

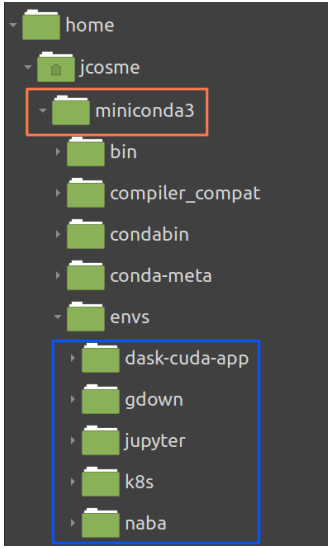
Note: the '--add' flag will, in addition to adding the channel, make it the *highest priority* channel. Since we want the highest priority channel to be 'conda-forge' we should add it last.

Overview

To use:

1. **Create** an environment

Conda will store your environments in the 'envs' folder, within the conda software folder (called 'miniconda3' in my case).



2. **Activate** the environment

In order to install packages or use an environment, it must be activated first

3. **Install** your packages

Install any software or packages you want within the activated environment

4. **Use** the environment

Run your scripts or start your applications from within the activated environment

The 3 main commands:

1. To create an environment, use:
mamba **create**

```
mamba create -n {your-env-name}
```

2. To activate an environment, use:
conda **activate**

```
conda activate {your-env-name}
```

3. To install packages in an active environment, use:
mamba **install**

```
mamba install {package-name}
```

To install packages in a specific, non-active environment, add the **'-n' flag**, followed by the environment name

```
mamba install -n {an-env-name} {package-name}
```

You can **install packages** when creating a new environment by specifying the package name(s) afterwards

```
mamba create -n {your-env-name} {package-name-1} {package-name-2}
```

You can install a **specific version** by adding the '=' sign, followed by the version (no spaces)

```
mamba create -n {your-env-name} {package-name}={version}
```

```
mamba install {package-name}={version}
```

You can install from a **specific channel** by using the '-c' flag, followed by the channel

```
mamba create -n {your-env-name} -c {a-channel} {package-name}
```

```
mamba install -c {a-channel} {package-name}
```

Here are a couple of examples:

```
mamba create -n my-proj -c conda-forge python=3.8 pandas
```

```
mamba install -c conda-forge python=3.8 pandas
```

Other important commands:

To deactivate an environment, use:
conda **deactivate**

```
conda deactivate
```

To view all environments, use:
conda **env list**

```
conda env list
```

To delete an environment, use:
conda **env remove**

```
conda env remove -n {your-env-name}
```

To remove packages, use:
mamba **uninstall**

```
mamba uninstall {package-name}
```

To export an ***active*** environment**, use:
mamba **env export**

```
mamba env export > {env-export-file-name}
```

To re-create an exported environment**, use:
mamba **env create**

```
mamba env create -f {env-export-file-name}
```

**More information on 'exporting' to follow

Exporting & recreating environments

What is exporting conda environments and why do it?

Exporting an environment creates a file (yaml) that **lists** all the **packages and version** that are **installed in** that **environment**. It will also save the channels, and any environment variables saved to the environment.

This **file can be saved to a repository** of your project code for others to be able to easily create an environment that will successfully run your code.

Can also be **used by you to recreate the environment**, in the case where you have not touched the code for years and need to once again work on it.

Can also be **used in docker image creation** whenever you need to containerize your code.

Allows you to **remove** the conda **environment** (and free up space**) once you are finished** working on the project.

stui.yaml

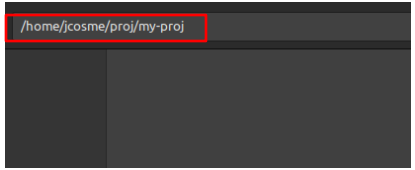
```yaml
name: stui
channels:
  - conda-forge
dependencies:
  - _libgcc_mutex=0.1=conda_forge
  - _openmp_mutex=4.5=2_gnu
  - bzip2=1.0.8=h7f98852_4
  - ca-certificates=2022.6.15=ha878542_0
  - ld_impl_linux-64=2.36.1=hea4e1c9_2
  - libffi=3.4.2=h7f98852_5
  - libgcc-ng=12.1.0=h8d9b700_16
  - libgomp=12.1.0=h8d9b700_16
  - libnsl=2.0.0=h7f98852_0
  - libsqlite=3.39.2=h753d276_1
  - libuuid=2.32.1=h7f98852_1000
  - libzlib=1.2.12=h166bdaf_2
  - ncurses=6.3=h27087fc_1
  - openssl=3.0.5=h166bdaf_1
  - pip=22.2.2=pyhd8ed1ab_0
  - psutil=5.9.1=py310h5764c6d_0
  - python=3.10.6=ha86cf86_0_cpython
  - python_abi=3.10=2_cp310
  - readline=8.1.2=h0f457ee_0
  - s-tui=1.1.3=py310hff52083_1
  - setuptools=65.3.0=py310hff52083_0
  - tk=8.6.12=h27826a3_0
  - tzdata=2022c=h191b570_0
  - urwid=2.1.2=py310h5764c6d_6
  - wheel=0.37.1=pyhd8ed1ab_0
  - xz=5.2.6=h166bdaf_0
prefix: /home/jcosme/miniconda3/envs/stui
```

Export conda environments

When exporting conda environments, the yaml file will be created in the current directory.

In our example, we start in the following directory: /home/jcosme/proj/my-proj

```
jcosme@jane:~/proj/my-proj$ pwd
/home/jcosme/proj/my-proj
```



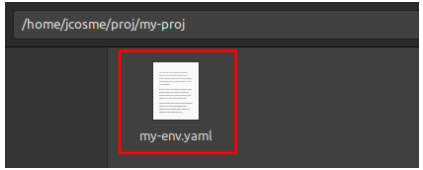
1. Activate the conda environment

```
conda activate {your-env-name}
```

2. Run the export command (use mamba)

```
mamba env export > {env-export-file-name}
```

```
jcosme@jane:~/proj/my-proj$ conda activate my-env  1
(my-env) jcosme@jane:~/proj/my-proj$ mamba env export > my-env.yaml  2
(my-env) jcosme@jane:~/proj/my-proj$
```

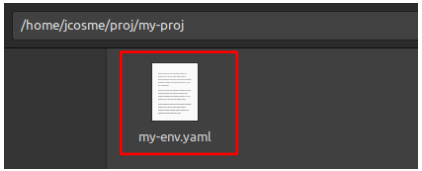


3. (optional) deactivate and delete environment

Recreate conda environments

1. Start in the directory with the yaml file

```
jcosme@jane:~/proj/my-proj$ pwd
/home/jcosme/proj/my-proj
```



2. Make sure the environment name doesn't already exist

```
jcosme@jane:~$ conda env list
# conda environments:
#
base                  *  /home/jcosme/miniconda3

jcosme@jane:~$
```

3. Run the following command (use mamba)

```
mamba env create -n {your-env-name} -f {env-export-file-name}
```

The key is the '-f' flag, which tells conda to create an environment from a file.

The '-n' flag is optional in this case, as the yaml file already contains a name. If you want to use the environment name from the file, omit the '-n' flag:

```
mamba env create -f {env-export-file-name}
```

In our example, I could use the following to create the 'my-env' environment

```
mamba env create -f my-env.yaml
```

Miscellaneous

Using pip

- The preferred way of installing is using mamba/conda.
- Sometimes mamba/conda don't have the package version you need.
- In these cases, you can use pip

However, <u>if you must you pip</u>:

1. Use mamba/conda to install everything that you can
2. <u>Use pip at the end</u>

Skip installation confirmation

By default, when installing mamba/conda packages, the user will be prompted to confirm the installs

```
Confirm changes: [Y/n]
```

You can skip this confirmation by adding the '-y' flag at the **end** of an install command

```
mamba install {package-name} -y
```

```
mamba install -c {a-channel} {package-name}={version} -y
```

Error recreating environment from yaml file

Sometime, when recreating an environment from a yaml file, you might run into an error that looks like this:

```
Encountered problems while solving:
  - nothing provides requested ptxcompiler ==0.2.0 py38h98f4b32_0
```

In these situations, open the yaml file, locate the line with the package…

```
p2-etl.yaml
289     - psutil=5.9.2=py38h0a891b7_0
290     - pthread-stubs=0.4=h36c2ea0_1001
291     - ptxcompiler=0.2.0=py38h98f4b32_0
292     - ptyprocess=0.7.0=pyhd3deb0d_0
293     - pure_eval=0.2.2=pyhd8ed1ab_0
```

…and comment out anything beyond the name of the package.

```
p2-etl.yaml
289     - psutil=5.9.2=py38h0a891b7_0
290     - pthread-stubs=0.4=h36c2ea0_1001
291     - ptxcompiler#=0.2.0=py38h98f4b32_0
292     - ptyprocess=0.7.0=pyhd3deb0d_0
293     - pure_eval=0.2.2=pyhd8ed1ab_0
```

This will allow mamba/conda to automatically find a suitable version of the package.

Conda environment variables

It is possible to **set** environment **variables in** an **activated environment**, using the format:

```
conda env config vars set {env-var-name}={env-var-value}
```

For example:

```
conda env config vars set TF_FORCE_GPU_ALLOW_GROWTH=true
```

To **view variables** in an activated environment, run:

```
conda env config vars list
```

To **remove** environment **variables** in an activated environment, use this format:

```
conda env config vars unset {env-var-name}
```

For example:

```
conda env config vars unset TF_FORCE_GPU_ALLOW_GROWTH
```

Set environment variables *will* be included in yaml file during exports, and *will* be set when recreating environments from yaml files.

Thank you!

info@run.ai | www.run.ai