Conda Code Clinic

What is Conda

- Also known as **Anaconda**, originally (2009) for **data science** usage with a **Python/R focus**.
- Now more towards general **scientific computing** usage with packages for wider range of programming languages
- **Package Manager** – provides repository of software, libraries, multi-versions and dependency.
- **Environment Manager** – (allows different packages and versions per environment)
- **Windows GUI and Command Line**
- **Open Source** – BSD license (no licencing or IP worries)
- **Popular**, **easy** learning curve, well **supported**, lots of documentation

Why use Conda ?

- **Configuration Management** – Organise and maintain software, their dependencies and environments.
- **Reproducibility** – Verifiable, collaborative, transparent, reusable - The Turing way.
- **Open Research** – Help make all aspects of our research accessible and meet our funding commitments toward open research.
- A **codified, canonical** source of your software configuration.
- **Security** - Does not need admin rights. Individual user installation and configuration.
- **Universal** – No external (or system) dependencies, runs on HPC and desktop (varying levels of cross platform support (Linux/Windows/Mac), (Intel/IBM/ARM)).
- **ARC3/ARC4** – Supported on our HPC systems.

Conda is NOT..

- **NOT Version control** (use Git, Github, Gitlab)
- **NOT Workflow management** (use snakemake, nextflow)
- **NOT Containers** (use Docker, Singularity)
- **NOT a repository platform** (Use Binder, Zenodo)
- **NOT Automatic** – it is a tool to help you manage and maintain your code projects, it won't do that for you.

Conda Alternatives

- **The alternatives:**
- Often have **external dependencies**
- Often need **admin rights** to install
- Often do not work easily on **HPC**
- Often **less user friendly** (not every researcher is a computer expert).

- **Examples**
- **Python** – pip, venv, poetry
- **R** – CRAN, renv
- Other **languages** – cargo, Maven, npm, RubyGems, Yarn
- **Containers** – Docker/Singularity
- **Virtual Machines** - Vagrant

Getting Conda

- Download instructions are provided from the main **anaconda.org** website
- There are 2 main versions of the Conda installer
  - The full windows installer, available from the main Anaconda website includes the **Navigator GUI**.
  - **Miniconda** is available from the conda docs website. This is our preferred option, its a far smaller download, provides just enough packages to manage your packages and environments. Does not include a GUI which can’t be used on HPC, easier to record your packages and enviroments.

Packages

- Packages can contain any combination of **software, libraries and their metadata** (such as version, build, dependencies, architectures will run on)
- conda keeps track of the **packages and their dependencies**
- conda can **search, install, remove, build, update** packages.
- **Meta packages** exist to install a collection of packages, they specify dependencies without containing any software or libraries.

Channels

- **Channels** are the **different locations** that conda downloads packages from
- **Common channels** are:
  - conda (default)
  - bioconda - https://bioconda.github.io/
  - conda-forge – https://conda-forge.org/
- conda allows you to **manage channels** (add, remove, set default)

CHANNELS AND PACKAGES

Tip: Package dependencies and platform specifics are automatically resolved when using conda.

install packages from specified channel	`conda install -c CHANNELNAME PKG1 PKG2`
list installed packages	`conda list`
uninstall package	`conda uninstall PKGNAME`
update all packages	`conda update --all`
install specific version of package	`conda install PKGNAME=3.1.4`
install a package from specific channel	`conda install CHANNELNAME::PKGNAME`
install package with AND logic	`conda install "PKGNAME>2.5,<3.2"`
install package with OR logic	`conda install "PKGNAME [version='2.5|3.2']"`
list installed packages with source info	`conda list --show-channel-urls`
view channel sources	`conda config --show-sources`
add channel	`conda config --add channels CHANNELNAME`
set default channel for pkg fetching (targets first channel in channel sources)	`conda config --set channel_priority strict`

Environments

- Conda allows you to create **isolated environments** containing configuration, packages and their dependencies.
- These are individual environments, allow you to install **different versions** (eg Python) in each environment, one per project is a good start.
- Avoids excessive amounts of packages being installed – easier to manage and can slow package management operations down.
- Easier to track down issues related to specific packages installed.

WORKING WITH CONDA ENVIRONMENTS

Tip: List environments at the beginning of your session. Environments with an asterisk are active.	
list all environments and locations	`conda env list`
update all packages in environment	`conda update --all --name ENVNAME`
install packages in environment	`conda install --name ENVNAME PKG1 PKG2`
remove package from environment	`conda uninstall PKGNAME --name ENVNAME`

ENVIRONMENT MANAGEMENT

Tip: Specifying the environment name confines conda commands to that environment.	
list packages + source channels	`conda list -n ENVNAME --show-channel-urls`
uninstall package from specific channel	`conda remove -n ENVNAME -c CHANNELNAME PKGNAME`
create environment with Python version	`conda create -n ENVNAME python=3.10`
clone environment	`conda create --clone ENVNAME -n NEWENV`
list revisions made to environment	`conda list -n ENVNAME --revisions`
restore environment to a revision	`conda install -n ENVNAME --revision NUMBER`
delete environment by name	`conda remove -n ENVNAME --all`

Environment.yml

- Possibly **the most useful and important** file in conda.
- Defines the environment name, channels and packages (and their versions) to install into an environment.
- Best to **create manually** (rather than generate from current environment.
- Can also be used to specify any **pip dependencies**.
- Is stored separately from the environment (treat as part of your code).

Example environment.yml

```yaml
name: arcdocs-jb
channels:
  - defaults
dependencies:
  - python=3.7.6
  - jinja2=3.0.3
  - pip=20
  - pip:
    - jupyter-book==0.9.1
```

Some Best Practices

- **Use miniconda** if possible (far smaller base)
- **Do not add any packages to the base environment** (use a specific environment)
- **One Environment for each project** is a good start (for clean separation and performance)
- Only use the **minimum packages** you need for each project (in each environment), remove unneeded packages.
- **Manually write your environment.yml**, never use an export (exports can by restrictive making updates to packages and adding packages hard).
- **Specify versions of the top level libraries** you need where required (maintain consistancy, could affect your results).
- **Don’t specify all the dependencies** (harder to make changes)
- Use **version control to store your environment file** alongside your code.

Further Reading and References

- **Anaconda open source** - https://www.anaconda.com/products/distribution
- **Miniconda** - https://docs.conda.io/en/latest/miniconda.html
- **Conda Docs** - https://docs.conda.io/en/latest/
- **Conda Cheat Sheet** - https://docs.conda.io/projects/conda/en/latest/user-guide/cheatsheet.html
- **The Turing Way** - https://the-turing-way.netlify.app/
- **Open Research** - https://library.leeds.ac.uk/info/1406/researcher_support/199/open_research
- **Research Computing website** - https://arc.leeds.ac.uk/
- **Research Computing Documentation** - https://arcdocs.leeds.ac.uk/