Working with Anaconda/Miniconda

What is Anaconda?

Anaconda is a distribution platform and toolkit for running independent software environments, especially for python but it can be used for almost any software component. So if you need a specific version of python, or a specific version of some of its modules, this may be the place to go. Anaconda is a full environment, Miniconda is the minimal set to get started. Both will function the same once installed and configured

See Anaconda website for more information and downloads

Installing Anaconda

Method 1: Use the provided version (easiest, recommended method)

All our desktops and compute nodes currently have conda pre-installed, so you can skip the download step, and with this version, the `init` step below can also be skipped. So if you choose this easiest solution. proceed to creating your environments (or tweaking the setup through config commands)

Method 2: Download and install it yourself (NOT recommended)

You can install anaconda or Miniconda (see anaconda website). This downloads a shell script you can run (either chmod 755 to make it executable, or run 'bash Miniconda3-latest-Linux-x86_64.sh' or whatever the name of your download is. Choose the location where you want to install the distribution (in a directory you own, but if you plan to install a lot of software, you should probably not install it in your home directory).

Method 3: Use EasyBuild installed versions of Miniconda

We have pre-installed versions of Miniconda available through EasyBuild. Run

```
module load Miniconda3
```

to load it (or check first using `module avail Miniconda` what versions are available. You will see that there may also be Miniconda environments with some software already installed.

Setup

Now that you have the `conda` package manager available, you can use it to complete the setup. One bit of setup that `conda` insists on, is to add some initialization code to the login environment, so

conda will always be active (`conda init bash` or `conda init tcsh`). However, see “pitfalls” section below.

Note that with the system default version of conda, this init step is not necessary!

Other bits of setup can be done through the `conda config` commands, see `conda config –help` for details. It is also possible to edit the `$HOME/.condarc` directly if you know what settings you want (eg by looking them up online).

Creating an environment

One conda install can manage several environments, which are independent (except when set to inherit from another environment). The original install creates an environment `base`, but it is best to create separate environments before using and modifying anything. To create an environment called TEST, you run:

```
conda create --name TEST
```

You can add package names and versions to add some packages immediately to the newly created environment, e.g.

```
conda create --name TEST python==3.6
```

For more information, see the Anaconda website and output of `conda create –help`

Once created, you activate the environment using `conda activate NAME`

Installing packages

Install a package using `conda install packagename`, optionally with a version as in `conda install python==3.7` to make sure that exact version gets installed, if available.

If unsure what packages are available, use `conda search packagename` to search for any matching names (wildcards allowed, should be enclosed in quotes). To list installed packages, use `conda list`.

Pitfalls

Interference with system python environments

As you can see above, the default behaviour when installing conda or running `conda init` is, to add some code to your .bashrc or .tcshrc to activate conda on every shell and in every window you open. That may sound like a nice feature, until you realize, that a custom version of python with a custom set of packages may wreak havoc on any environment that relies on the system default version. And there is a lot of software that happens to be written in python, or linked with it, including big parts of

the Gnome and Cinnamon desktop applications. So, if you happen to set up a non-standard version of python as default, it might not be possible for you to log in in these desktop environments any more. Or even worse, if you need to install non-standard versions of gcc or its libraries, you might not be able to run a big part of the system software any more.

Solutions to these pitfalls

First of all, no workarounds are needed if you use the system-provided version of `conda` (method 1 in the description above), everything has been taken care of in this case. So once again, if you can use the system version of conda, please do. It makes life so much easier on you (and on the computer group).

—

When using a miniconda or anaconda from an environment module, or one you downloaded and installed yourself, here are some possible workarounds to the `conda init` issues:

For the `bash` shell, the simplest workaround is not activating conda in your `.bashrc` but activating it through

```
source activate TEST
```

when TEST is the name of your conda environment. This works, whereas the usual `conda activate` TEST will fail and mention it needs the initialization code in `.bashrc`

Another solution is, to use a different shell for everything that requires `conda`. So if your default login shell is `tcsh`, you could type `bash` in any session where you want to do some work with `conda`, and do `conda init bash` to add the conda init stuff to your `.bashrc` without influencing anything that runs in `tcsh`, including your desktop login session.

One remaining problem: the x2go remote login facility always uses `bash` to run its initial login. So if you use `x2go`, this method might not work. Of course you can reverse the role of the shells, set your default shell to `bash` and leave that with the default setup so all logins including `x2go` will work fine, and start `tcsh` when you need `conda`, and run `conda init tcsh` to set it up.

Disk usage

Conda environments can be big. No wonder, since they can contain a full python install, libraries, compilers and many other tools. And as with most programs, conda defaults to store all of this in your $HOME since that is the one place that is known to exist on any UNIX system.

So, you will want to change this location to something with more space, e.g. a local `/data1` or `/data2` disk. This can be done in `.condarc` by settings `envs_dirs` and `pkgs_dirs` to a chosen location. If you don't have a `.condarc` yet, create one with a text editor and add something like:

```
auto_activate_base: false
envs_dirs:
  - /data2/yourname/conda/envs
pkgs_dirs:
```

```
  - /data2/yourname/conda/pkgs
```

Additional tip: having the environments locally on your workstation has advantages and disadvantages. One advantage is, that with the same setup, you can have a different instance of an environment on another machine, which is convenient if that other machine is not running the same Linux version as your desktop (eg compute nodes and vdesk). Disadvantage: if you occasionally use another desktop, you will not see your environments. In that case, use `/net/computername/data2` in stead of `/data2`. in the configuration.

If you are making this change when you already have some conda environments (in $HOME/.conda), you can move them to the new location. And just to be sure, in case there are already hard-coded path names in the environment, make a symbolic link. So, using the example .condarc shown above:

```
mv .conda /data2/yourname/conda
ln -s /data2/yourname/conda ~/.conda
```