Conda Cluster Instructions

1. Download the latest version of *anaconda/miniconda*. When working on a remote cluster, I recommend *miniconda*. It is a barebones version that does not come with a lot of the GUI and other things that cannot be used remotely. The latest versions can be found here: https://docs.conda.io/en/latest/miniconda.html

Right click and copy the link for the linux shell (sh) script and then use wget to download:

```
wget <link_to_miniconda>
```

A real version of this looks like this:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
```

2. Run the downloaded shell script and follow the instructions.

```
bash <miniconda_script.sh>
```

A real version of this looks like this:

```
bash Miniconda3-latest-MacOSX-x86_64.sh
```

Once miniconda is installed, you don't need to reinstall it.

3. Different software programs are curated and available through different channels. Without specifying or adding any channels, conda will only look in the "defaults" channel.

If we type:

```
conda search bowtie2
```

We will not find anything. This is because bowtie2 is found on the "bioconda" channel. We can search this channel specifically using the **-c bioconda** option:

```
conda search -c bioconda bowtie2
```

However, we a lot of bioinformatics tools are available primarily through bioconda. Rather than always include the **-c bioconda** option, we can add it to our list of channels, so that conda always searches in bioconda.

We can see the list of current channels:

```
conda config --show-sources
```

If you have just installed miniconda, then you should see this:

```
channels:
- defaults
restore_free_channel: True
report_errors: False
```

To add channels, we ca do the following:

```
conda config --add channels <your_channel>
```

There are many possible channels, the two additional ones I find most useful are *bioconda* and *conda-forge*

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

The order of the channels matter. The last channel added is placed at the top of the list and given priority in installation.

```
conda config --show-sources
```

```
channels:
- bioconda
- conda-forge
- defaults
restore_free_channel: True
report_errors: False
```

Now we can search conda again for *bowtie2*

```
conda search bowtie2
```

It should now show up without specifying the channel.

4. When you login, your "base" Conda environment will typically load automatically. I highly recommend that you do not install subsequent software into your base environment. This is because as you install more programs, the various software version requirements can start to conflict with each other, which will then result in chaos with each new software installed and each attempt to update. This also makes version control between projects very difficult. **Instead, you should create a seperate conda environment (env) for each software or project**. An environment is kind of a standalone space on the computer/server where you can install and use completely different versions of various programs and avoid conflict.

Some people like to do this on a software by software basis...so if you are installing bowtie2, this will have its own conda environment. I do this sometimes, especially for software that has really specific requirements, e.g. a program that requires python2, rather than python3. Otherwise, I often create an environment for the particular project. The other advantage of Conda environments, is that you can export the environment to a "yml" file, which other people can use to install the same Conda environment, thus ensuring consistent use of software.

To create a conda environment:

```
conda create -n <name_your_environment>
```

Here we will create an environment for installing the program *methylpy*

```
conda create -n methylpy
```

5. Now we need to "activate" the conda environment to install programs or use those programs.

```
conda activate <environment_name>
```

A real version of this looks like this:

```
conda activate methylpy
```

6. We can now install software. Our goal here is to install methylpy and all its required software (see: https://github.com/yupenghe/methylpy or https://pypi.org/project/methylpy/).

First we will search for the methylpy package on conda:

```
conda search methylpy
```

At the time of this writing, this returns:

```
Loading channels: done
# Name Version Build Channel
methylpy 1.4.1 py27h41a55b7_0 bioconda
methylpy 1.4.1 py36h41a55b7_0 bioconda
methylpy 1.4.1 py37h41a55b7_0 bioconda
methylpy 1.4.2 py27h41a55b7_0 bioconda
methylpy 1.4.2 py36h41a55b7_0 bioconda
methylpy 1.4.2 py37h41a55b7_0 bioconda
methylpy 1.4.3 py27h41a55b7_0 bioconda
methylpy 1.4.3 py36h41a55b7_0 bioconda
methylpy 1.4.3 py37h41a55b7_0 bioconda
```

We can install methylpy by typing **(dont do this just yet!)**:

```
conda install methylpy
```

or we can install a specific version by typing **(dont do this just yet!)**:

```
conda install methylpy=1.4.3
```

or specific build **(dont do this just yet!)**:

```
conda install methylpy=py27h41a55b7_0
```

or from a specific channel **(dont do this just yet!)**:

```
conda install -c bioconda methylpy
```

However, the latest version of methylpy is 1.4.6, while only 1.4.3 is available on conda. Fortunately, version 1.4.6 is available through *pip* ...which can also work with conda. However first we need to install *python* and *pip*.

```
conda install python
```

This should install *pip* as well. You can double check this by typing:

```
which pip
```

Which should include the path to your environment:

```
~/miniconda3/envs/methylpy/bin/pip
```

Now we can use pip to install methylpy within our current environment:

```
pip install methylpy
```

We also need to install some other prerequisites, such as *samtools*, *bowtie2*, *cutadapt*, *picard*, and *wigToBigWig*

```
conda install samtools
```

```
conda install bowtie2
```

```
conda install cutadapt
```

```
conda install picard
```

```
conda install wigToBigWig
```

7. You can now run *methylpy*

```
methylpy --help
```

You will always need to activate the conda environment in order to use methylpy. So each time you log in and want to use it, you need to type:

```
conda activate methylpy
```

However, methylpy takes a long time to run. On the cluster, we will need to submit jobs to the SLURM manager in order to run it. However, submitted scripts do not recognize the *conda activate* command. There is a work around however.

LINUX systems use environmental variables to know where to look for things. Everytime you type in a command on the cluster, it uses these environmental variables to point it to where the actual program may be located and run that program. There can be multiple types of environmental variables, the main one however is *PATH*. Also relevant to us is the variable *LD_LIBRARY_PATH*.

You can see what these look like by typing:

```
echo $PATH
```

```
echo $LD_LIBRARY_PATH
```

You may see something like this:

```
echo $PATH
/mnt/home/niederhu/miniconda3/envs/methylpy/bin:/mnt/home/niederhu/miniconda3/condabin:/opt/software/powertools/bin:/opt/software/MATLAB/2018a:/opt/software/MATLAB/2018a/bin:/opt/software/Java/1.8.0_152:/opt/software
foss-2018a/bin:/opt/software/SQLite/3.21.0-GCCcore-6.4.0/bin:/opt/software/Tcl/8.6.8-GCCcore-6.4.0/bin:/opt/software/libreadline/7.0-GCCcore-6.4.0/bin:/opt/software/ncurses/6.0-GCCcore-
6.4.0/bin:/opt/software/CMake/3.11.1-GCCcore-6.4.0/bin:/opt/software/bzip2/1.0.6-GCCcore-6.4.0/bin:/opt/software/FFTW/3.3.7-gompi-2018a/bin:/opt/software/OpenBLAS/0.2.20-GCC-6.4.0-
2.28/bin:/opt/software/imkl/2018.1.163-gompi-2018a/mkl/bin:/opt/software/imkl/2018.1.163-gompi-2018a/bin:/opt/software/OpenMPI/2.1.2-GCC-6.4.0-2.28/bin:/opt/software/binutils/2.28-GCCcore-
6.4.0/bin:/opt/software/GCCcore/6.4.0/bin:/usr/lib64/qt-
3.3/bin:/opt/software/core/lua/lua/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/local/hpcc/bin:/usr/lpp/mmfs/bin:/opt/ibutils/bin:/opt/puppetlabs/bin:/opt/dell/srvadmin/bin
```

These variables are hierarchical and separated by colons. When you type a command like *methylpy*, the system looks at the first path in that list for the program, in this case */mnt/home/niederhu/miniconda3/envs/methylpy/bin*. If it doesn't find it there, it proceeds to the next in the list, and so on.

When you type in the command *conda activate*, what it is primarily doing is setting various environmental variables to that environment's path. It sets these at the start, so it looks first within your conda environment.

We can therefore, achieve the same result for our SLURM submission script by setting these variables in the script itself. So in the same script you submit to run methylpy, add these lines before your invocation of methylpy:

```
export PATH="<path_to_your_env>:$PATH"
```

This is using the *export* command to set the variable, which does so globally. We include the full path to the *bin* directory in your conda evnironment, followed by a colon, followed by the variable *$PATH*, which ensures that we keep the original values of the *PATH* variable.

So where are these located?

Conda installs all software within itself...in this instance, that is the directory "miniconda3". For me, this located in my home directory "/mnt/home/niederhu". Within the "miniconda3" directory, new environments are stored in the "envs" directory. Each of the environments you create using *conda env create -n* exist as a directory within this directory. So the "methylpy" environment is located in "/mnt/home/niederhu/miniconda3/envs/methylpy". Executable scripts and programs are stored in the "bin" directory. Libraries, which many programs need and reference are stored in the "lib" directory.

So in my shell script that I plan to submit to SLURM, I put the line:

```
export PATH="/mnt/home/niederhu/miniconda3/envs/methylpy/bin:$PATH"
```

There are also various libraries often required by these programs. We set the *LD_LIBRARY_PATH* to point to these. Use the same path as for your "bin" directory, but replace "bin" with "lib":

```
export LD_LIBRARY_PATH="/mnt/home/niederhu/miniconda3/envs/methylpy/lib:LD_LIBRARY_PATH"
```

You should now be able to submit a script like this:

```
#!/bin/bash --login
#SBATCH --time=168:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=20
#SBATCH --mem=100GB
#SBATCH --job-name methylpy
#SBATCH --output=%x-%j.SLURMout
cd $PBS_O_WORKDIR
export PATH="/mnt/home/niederhu/miniconda3/envs/methylpy/bin:$PATH"
export LD_LIBRARY_PATH="/mnt/home/niederhu/miniconda3/envs/methylpy/lib:LD_LIBRARY_PATH"
methylpy <commands>
```

8. Some notes about the shell scripts & SLURM.

The hashtagged lines are necessary for submitting a script to the SLURM job manager. I will explain some of these below. There are others, which can be found in the HPCC documentation:
https://wiki.hpcc.msu.edu/display/ITH/High+Performance+Computing+at+iCER

a. `#!/bin/bash --login` Every job must begin with this b. `#SBATCH --time=168:00:00` This line requests the amount of time. The format here is hours:minutes:seconds. 168 is the max hours you can request. c. `#SBATCH --nodes=1` The number of nodes on the cluster. With few exceptions, this will almost always be **1** . Most software we use are not designed for working with distributed memory (more than 1 node) and so only require 1 node. d. `#SBATCH --ntasks-per-node=1` Tasks (the job you are submitting) per node. This is almost always **1** . e. `#SBATCH --cpus-per-task=20` Many programs are capable of multi-threading, which greatly speeds up the task. This normally has to be specified in the program when you run it (see that program's instructions). So if you are going to run bowtie2 with 20 threads (bowtie2 -p 20), then you need to request 20 CPUs. Keep the number reasonable. There are only so many CPUs per node (look at ICER for descriptions of hardware). f. `#SBATCH --mem=100GB` How much memory (RAM) your program will need. This is done using GB (gigabyte) or MB (megabyte) and so on. Here I am requesting 100GB. Again, there are limitations depending on the node. Make sure you request enough, otherwise it will kill the job. But if you request too much, it will require a long time before your job starts. g. `#SBATCH --job-name flye` You can give a name to your job, so you know which one it is. h. `#SBATCH --output=%x-%j.SLURMout` Anything that the job would normally print to the screen (normal messages, error messages, etc) will be printed to this file. i. `cd $PBS_O_WORKDIR` This will tell the script to redirect to whatever directory you submitted the job from.

To submit the job:

```
sbatch myjob.sh
```

You can then check on the status of the job by typing:

```
qstat
```