Introduction to Python on HPC2N's systems

Birgitte Brydsø

HPC2N, Umeå University

9. September 2022

UMEÅ UNIVERSITET

Introduction to Python on HPC2N's systems

- HPC2N's compute cluster is called Kebnekaise. This presentation will look at how to use this cluster
- HPC2N has both Python 2.7.x and Python 3.x installed. We will be using Python 3.x in this course
- For this course, the recommended version of Python to use on Kebnekaise is 3.9.5
- HPC2N has several Python packages already installed and available for the users - check first before installing yourself!
- HPC2N do NOT recommend (and do not support) using Anaconda/Conda on our systems. You can read more about this here: https://www.hpc2n.umu.se/documentation/guides/anaconda

Most programs are accessed by first loading them as a 'module'

Modules are:

- used to set up your environment (paths to executables, libraries, etc.) for using a particular (set of) software package(s)
- a tool to help users manage their Unix/Linux shell environment, allowing groups of related environment-variable settings to be made or removed dynamically
- allows having multiple versions of a program or package available by just loading the proper module
- installed in a hierarchial layout. This means that some modules are only available after loading a specific compiler and/or MPI version.

The Module System

Useful commands

- See which modules exists:
  `module spider` or `ml spider`
- Modules depending only on what is currently loaded:
  `module avail` or `ml av`
- See which modules are currently loaded:
  `module list` or `ml`
- Example: loading Python 3.9.5 and its prerequisite: ml GCCcore/10.3.0 Python/3.9.5
  `module load GCCcore/10.3.0 Python/3.9.5` or `ml GCCcore/10.3.0 Python/3.9.5`
- Example: Unload the above module and its prerequisite:
  `module unload GCCcore/10.3.0 Python/3.9.5` or `ml -GCCcore/10.3.0 -Python/3.9.5`
- More information about a module:
  `module show <module>` or `ml show <module>`
- Unload all modules except the 'sticky' modules:
  `module purge` or `ml purge`

The Module System

Example

```
b-an01 [~]$ ml spider Python

----------------------------------------------------------------------------
  Python:
----------------------------------------------------------------------------
    Description:
      Python is a programming language that lets you work more quickly and
      integrate your systems more effectively.

     Versions:
        Python/2.7.15
        Python/2.7.16
        Python/2.7.18-bare
        Python/2.7.18
        Python/3.7.2
        Python/3.7.4
        Python/3.8.2
        Python/3.8.6
        Python/3.9.5-bare
        Python/3.9.5
        Python/3.9.6-bare
        Python/3.9.6
     Other possible modules matches:
        Biopython  Boost.Python  GitPython  IPython  flatbuffers-python  ...
...
```

The Module System

Example

```
b-an01 [~]$ ml spider Python/3.9.5

----------------------------------------------------------------------------
  Python: Python/3.9.5
----------------------------------------------------------------------------
    Description:
      Python is a programming language that lets you work more quickly and integrate
      your systems more effectively.

    You will need to load all module(s) on any one of the lines below before the "Python/3.9.5"
    module is available to load.

      GCCcore/10.3.0

    This module provides the following extensions:

       alabaster/0.7.12 (E), appdirs/1.4.4 (E), asn1crypto/1.4.0 (E), atomicwrites/1.4.0 (E)
       attrs/21.2.0 (E), Babel/2.9.1 (E), bcrypt/3.2.0 (E), bitstring/3.1.7 (E),
       blist/1.3.6 (E), CacheControl/0.12.6 (E), cachy/0.3.0 (E), certifi/2020.12.5 (E),
       cffi/1.14.5 (E), chardet/4.0.0 (E), cleo/0.8.1 (E), click/7.1.2 (E), clikit/0.6.2 (E),
       colorama/0.4.4 (E), crashtest/0.3.1 (E), cryptography/3.4.7 (E), Cython/0.29.23 (E),
       decorator/5.0.7 (E), distlib/0.3.1 (E), docopt/0.6.2 (E), docutils/0.17.1 (E),
       ecdsa/0.16.1 (E), filelock/3.0.12 (E), flit-core/3.2.0 (E), flit/3.2.0 (E),
       fsspec/2021.4.0 (E), future/0.18.2 (E), html5lib/1.1 (E), idna/2.10 (E),
       imagesize/1.2.0 (E), importlib_metadata/4.0.1 (E), iniconfig/1.1.1 (E),
...
```

The Module System - Example

```
b-an01 [~]$ ml

Currently Loaded Modules:
  1) snicenvironment (S)   2) systemdefault (S)

  Where:
   S:  Module is Sticky, requires --force to unload or purge

b-an01 [~]$ ml GCC/10.3.0 Python/3.9.5
b-an01 [~]$ ml

Currently Loaded Modules:
  1) snicenvironment (S)   5) binutils/2.36.1   9) libreadline/8.1  13) GMP/6.2.1
  2) systemdefault   (S)   6) GCC/10.3.0       10) Tcl/8.6.11       14) libffi/3.3
  3) GCCcore/10.3.0        7) bzip2/1.0.8      11) SQLite/3.35.4    15) OpenSSL/1.1
  4) zlib/1.2.11           8) ncurses/6.2      12) XZ/5.2.5         16) Python/3.9.5

  Where:
   S:  Module is Sticky, requires --force to unload or purge

b-an01 [~]$ ml purge
The following modules were not unloaded:
  (Use "module --force purge" to unload all):

  1) snicenvironment   2) systemdefault
b-an01 [~]$ ml

Currently Loaded Modules:
  1) snicenvironment (S)   2) systemdefault (S)

  Where:
   S:  Module is Sticky, requires --force to unload or purge
```

The Module System

Installed Python packages

- There are several Python packages already installed on Kebnekaise. To find them, use
  `module -r spider '.*Python.*'`
  and
  `module -r spider '.*python.*'`
- Be aware that not all modules output will be Python packages, some will just be programs that are compiled with Python, so you will need to go through the list
- Check with `module spider <package>` to see available versions, any prerequisites, as well as which Python versions they are compatible with

The Module System

List of some installed Python libraries and packages

- ASE
- Keras
- PyTorch
- SciPy-bundle (Bottleneck, deap, mpi4py, mpmath, numexpr, numpy, pandas, scipy - some of the versions have more)
- TensorFlow
- Theano
- matplotlib
- scikit-learn
- scikit-image

Running Python code

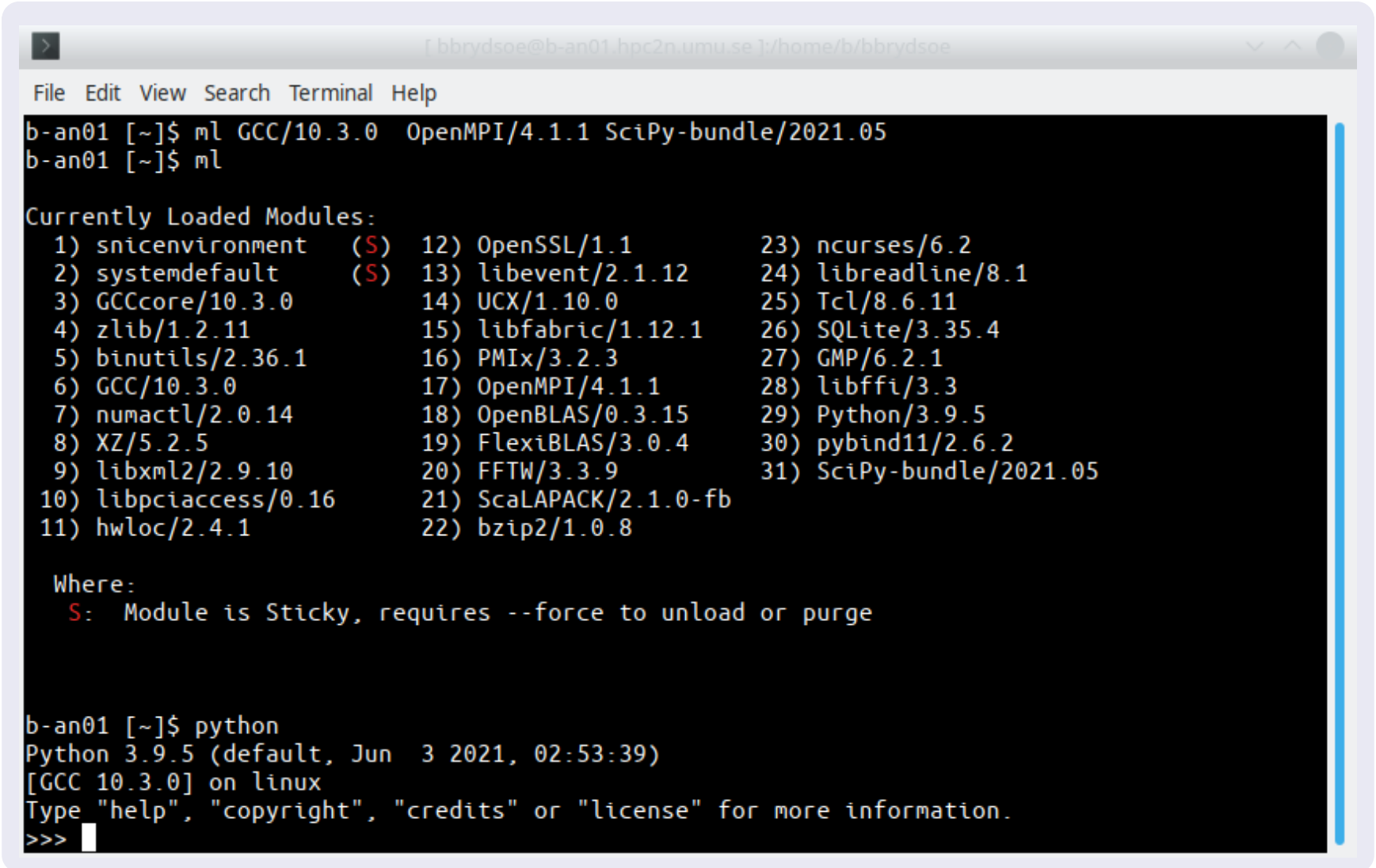
Command line

Only run programs from the command line if it is something short, not parallel, and which does not use a lot of resources!

- Load modules you need (Python, Python packages ...)
- (Install extra packages if needed, using virtual environment)
- Either put your Python program in a script and run it with `python <myscript>.py` or start an interactive Python session with the command `python`
- Example, loading SciPy-bundle v. 2021.05 and Python 3.9.5. Note that SciPy-bundle automatically loads Python!
  - `ml GCC/10.3.0 OpenMPI/4.1.1 SciPy-bundle/2021.05`
  - `python`

Running Python code

Command line, interactive Python session with SciPy-bundle/2021.05 and Python/3.9.5



Running Python code

Batch script

Any longer, resource-intensive, or parallel jobs should be run through a batch script.

- Inside the batch script you need to load the modules you need (Python, Python packages ...)
- (Install extra packages if needed, include via virtual environment, remember to activate script)
- Ask for resources depending on if it is a parallel job or a serial job, if you need GPUs or not, etc.
- Give the command(s) to your Python script
- Submit batch script with `sbatch <my-python-script.sh>`

Running Python code

Batch script, SciPy-bundle/2021.05 and Python/3.9.5, serial code

```
#!/bin/bash
#SBATCH -A SNIC2022-22-641 # Change to your own
#SBATCH --time=00:10:00 # Asking for 10 minutes
#SBATCH -n 1 # Asking for 1 core

# Load any modules you need
ml GCC/10.3.0 OpenMPI/4.1.1 SciPy-bundle/2021.05

python my_program.py
```

Submit the script with `sbatch my-python-script.se` (or whatever you have named it).

Running Python code

Batch script, SciPy-bundle/2021.05 and Python/3.9.5 + Python package you have installed yourself with virtual environment. Serial code

```
#!/bin/bash
#SBATCH -A SNIC2022-22-641 # Change to your own
#SBATCH --time=00:10:00 # Asking for 10 minutes
#SBATCH -n 1 #Asking for 1 core

# Load any modules you need
ml GCC/10.3.0 OpenMPI/4.1.1 SciPy-bundle/2021.05

# Activate your virtual environment.  Note that you
# need to have added the location to your path
source bin/activate

python my_program.py
```

Submit the script with `sbatch my-python-script.se` (or whatever you have named it).

Running Python code

Batch script, SciPy-bundle/2021.05 + Python/3.9.5 + TensorFlow/2.6.0-CUDA-11.3.1, GPU code

The recommended TensorFlow version for this course is 2.6.0. The module is compatible with Python 3.9.5 (automatically loaded when you load TensorFlow and its other prerequisites).

```
#!/bin/bash
#SBATCH -A SNIC2022-22-641 # Change to your own
#SBATCH --time=00:10:00 # Asking for 10 minutes
# Asking for one K80 card
#SBATCH --gres=gpu:k80:1

# Load any modules you need
ml GCC/10.3.0 OpenMPI/4.1.1 TensorFlow/2.6.0-CUDA-11.3.1

python my_tf_program.py
```

Editing your files

- Various editors: vi, vim, nano, emacs ...
- Example, vi/vim:
  - `vi <filename>`
  - Insert before: i
  - Save and exit vi/vim: `Esc :wq`
- Example, nano:
  - `nano <filename>`
  - Save and exit nano: `Ctrl-x`
- Example, Emacs:
  - Start with: emacs
  - Open (or create) file: Ctrl-x Ctrl-f
  - Save: Ctrl-x Ctrl-s
  - Exit Emacs: Ctrl-x Ctrl-c

The Batch System (SLURM)

Useful Commands

- Submit job: `sbatch <jobscript>`
- Get list of your jobs: `squeue -u <username>`
- `srun <commands for your job/program>`
- `salloc <commands to the batch system>`
- Check on a specific job: `scontrol show job <job id>`
- Delete a specific job: `scancel <job id>`
- Useful info about job: `sacct -l -j <jobid> | less -S`
- `job-usage <jobid>` gives you a url to a page with info about the job

The Batch System (SLURM)

Hints for the job script

- Output and errors in:
  `slurm-<job-id>.out`
- To get output and error files split up, you can give these flags in the submit script:
  `#SBATCH --error=job.%J.err`
  `#SBATCH --output=job.%J.out`
- To specify Broadwell or Skylake only:
  `#SBATCH --constraint=broadwell` or
  `#SBATCH --constraint=skylake`
- To run on the GPU nodes, add this to your script:
  `#SBATCH --gres=gpu:<card>:x`
  where <card> is k80 or v100, x = 1, 2, or 4 (4 only if K80).
- http://www.hpc2n.umu.se/resources/hardware/kebnekaise

Dealing with packages

Finding packages

- Already site-installed packages can be found as mentioned earlier
- If it is a package you know is used by many in your field of research, then you can ask support@hpc2n.umu.se if it can be installed
- All other packages you need to install yourself.
  - A good place to find packages to download is "The Python Package Index" (PyPi): https://pypi.org/
  - If the package is not on PyPi you will often download it from GitHub. Installing from this will be covered later in the course.
  - To see which packages you have installed, you can use `pip list --user`

Installing packages

Creating isolated environments

Python modules can often be installed through python, with the tool 'python setup.py', but some module can be difficult, and are easier to install in a stand-alone environment. In these cases we suggest using Virtualenv.

- On Kebnekaise you are strongly recommended not to use Anaconda and instead use **Virtual environments** (Virtualenv)
- Virtualenv is installed with each Python version at HPC2N. It is highly recommended to use that instead of installing yourself
- Your virtualenv need to be activated before installing Python packages and for using them
- More info here: https://www.hpc2n.umu.se/resources/software/user_installed/python

Creating a virtual/isolated environment

Example, Python 3.9.5 and installing to storage dir `/proj/nobackup/mystorage`

Install in $HOME (25GB) or in your project storage. In this example I use `/proj/nobackup/mystorage`

- Load Python: `ml GCC/10.3.0 Python/3.9.5`
- Create your Virtualenv, here called `vpyenv`
  - `virtualenv --system-site-packages /proj/nobackup/mystorage/vpyenv`
- You are done!
- To activate:
  - `cd /proj/nobackup/mystorage/vpyenv`
  - `source bin/activate`
- To deactivate: `deactivate`

Installing a Python package in a virtual environment

Change `/proj/nobackup/mystorage/vpyenv` to your `vpyenv` location

- Activate your Virtualenv. You also need to do this before using the package:
  - `cd /proj/nobackup/mystorage/vpyenv`
  - `source bin/activate`
- If the package to install is on PyPi you can use pip:
  `pip install --no-cache-dir --no-build-isolation <package>`
  - "–no-cache-dir" - required to avoid reuse of earlier installations by same user in different env
  - "–no-build-isolation" - ensures use of loaded modules when building. Also means no dependencies are installed
- If not on PyPi, download it, untar, and install with setup.py:
  `python setup.py install --prefix=<path to install dir>`

Installing a Python package in a virtual environment

Example, pip

Installing "seaborn" from PyPi, using pip, to the vpyenv, my storage directory `/proj/nobackup/mystorage`

- Load the Python version you need. I will use Python 3.9.5
  - `ml GCC/10.3.0 Python/3.9.5`
- Check dependencies for "seaborn" (numpy, pandas, matplotlib. Optionally scipy and/or statsmodels)
- Load compatible versions of site-installed modules for these dependencies (SciPy-bundle, matplotlib)
  - `ml OpenMPI/4.1.1 SciPy-bundle/2021.05 matplotlib/3.4.2`
- `cd /proj/nobackup/mystorage/vpyenv`
- Activate Virtualenv: `source bin/activate`
- Install: `pip install --no-cache-dir --no-build-isolation seaborn`

Installing a Python package with setup.py

- Pick a location (change below to fit)
  - `mkdir /proj/nobackup/mystorage/mypythonpackages`
  - `cd /proj/nobackup/mystorage/mypythonpackages`
- Load Python + site-installed prerequisites (SciPy-bundle, matplotlib, etc.)
- Install any remaining prerequisites. Remember to activate your Virtualenv if installing with pip
- Download Python package, place in install dir, untar/unzip it
- cd into the source directory of the Python package
- Run `python setup.py build`
- Install: `python setup.py install --prefix=<path to install dir>`
- Add to $HOME/.bash_profile (note that it will differ by Python version): `export PYTHONPATH=$PYTHONPATH:<path to your install directory>/lib/python3.9/site-packages`
- Import as usual to Python: `import <python-module>`

Connecting to HPC2N/Kebnekaise

- If you do not already have a preferred client for logging in to Kebnekaise, we recommend ThinLinc: https://www.cendio.com/thinlinc/download
- The ThinLinc login node is: `kebnekaise-tl.hpc2n.umu.se`
- If you are using another SSH client, you will use the regular Kebnekaise login node: `kebnekaise.hpc2n.umu.se`
- There is more information about Thinlinc here: https://www.hpc2n.umu.se/documentation/guides/thinlinc
- Information about other login methods here: https://www.hpc2n.umu.se/access/login

Various useful info

- A project has been set up for the workshop: `SNIC2022-22-641`
- You use it in your batch submit file by adding:

```
#SBATCH -A SNIC2022-22-641
```

- There is a reservation for XXX. This reservation is accessed by adding this to your batch submit file:

```
#SBATCH --reservation=XXX
```

- The reservation is ONLY valid for the duration of the course.