Setting up Environments for AI Research and Software Development on HPRC

November 21, 2025

Thang Ha

High Performance
Research Computing
DIVISION OF RESEARCH

Motivation

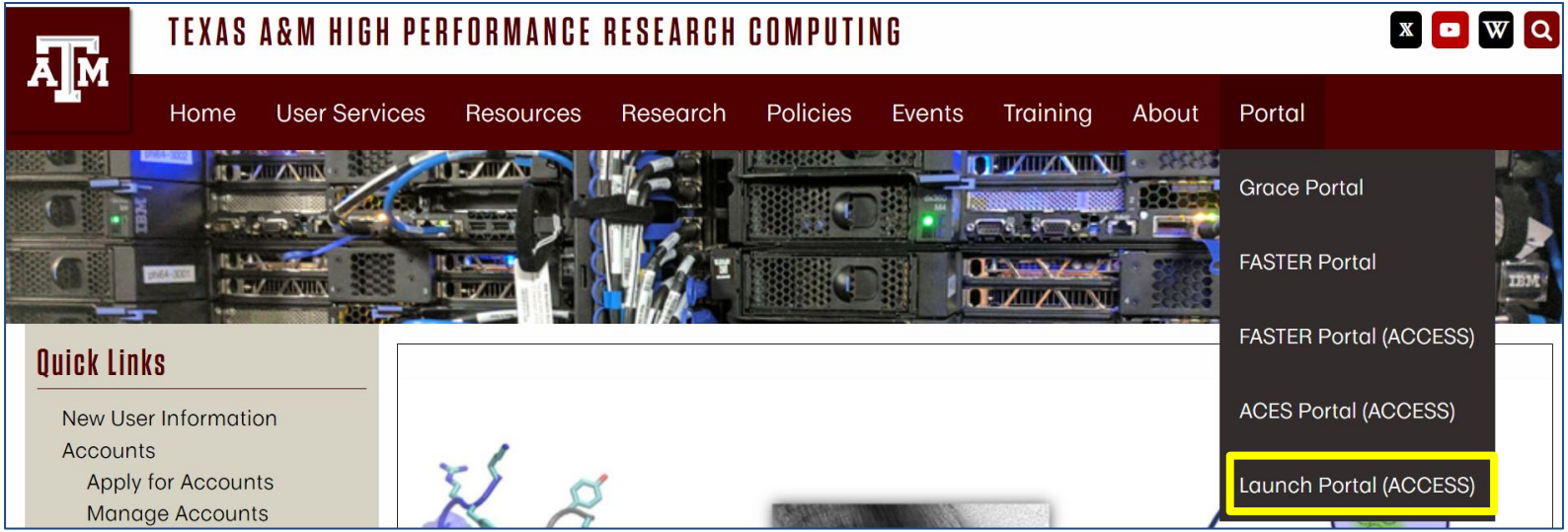
- You found a github repo having some features relevant to your research
- You were unable to follow the instruction of that github repo
- Even if you were able to follow the github repo's instruction, you ended up with errors after errors, filling up your storage quota, effectively locking yourself out of your own account.
- You "translate" the github repo's instruction to match our guidelines and efficiently run the github codes and scripts using powerful supercomputer hardware.

Setting Up Environments for:

- Running Python scripts/notebooks
  - Using ModuLair
  - Using Mamba/Conda
- Running R scripts
- Compiling software
  - Using GNU (FOSS) toolchain
  - Using Intel toolchain

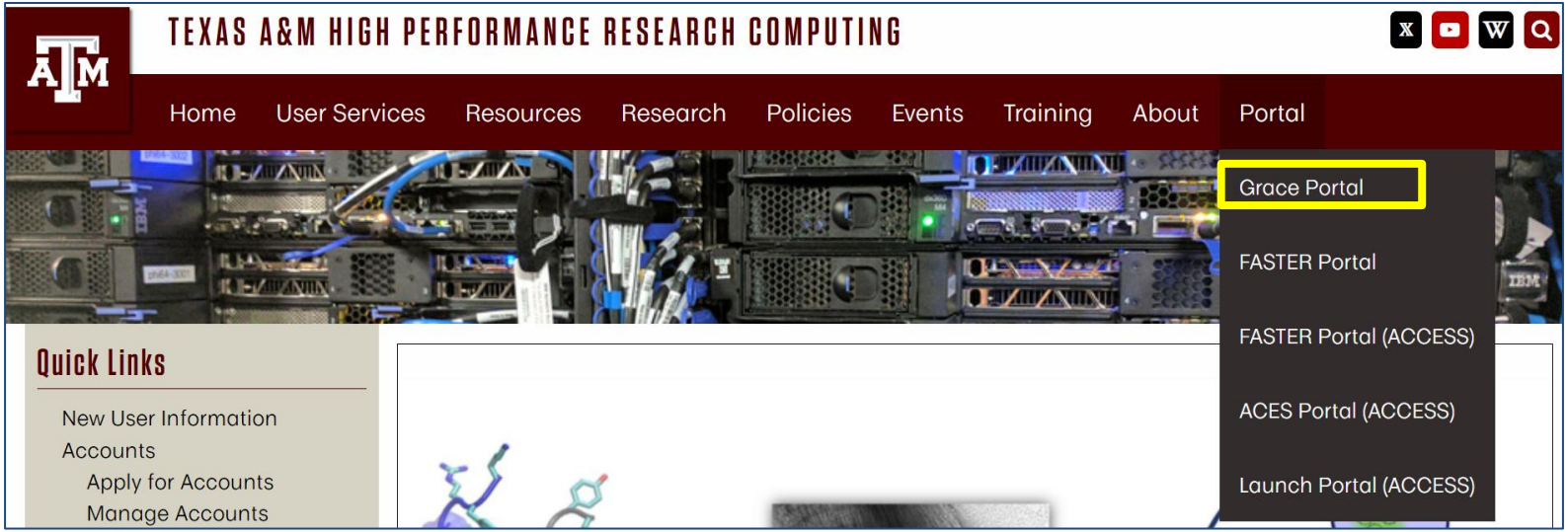
Accessing Launch: via HPRC Portal

- HPRC homepage: hprc.tamu.edu
- Select ‘Launch Portal (ACCESS)’ in Portal tab dropdown:



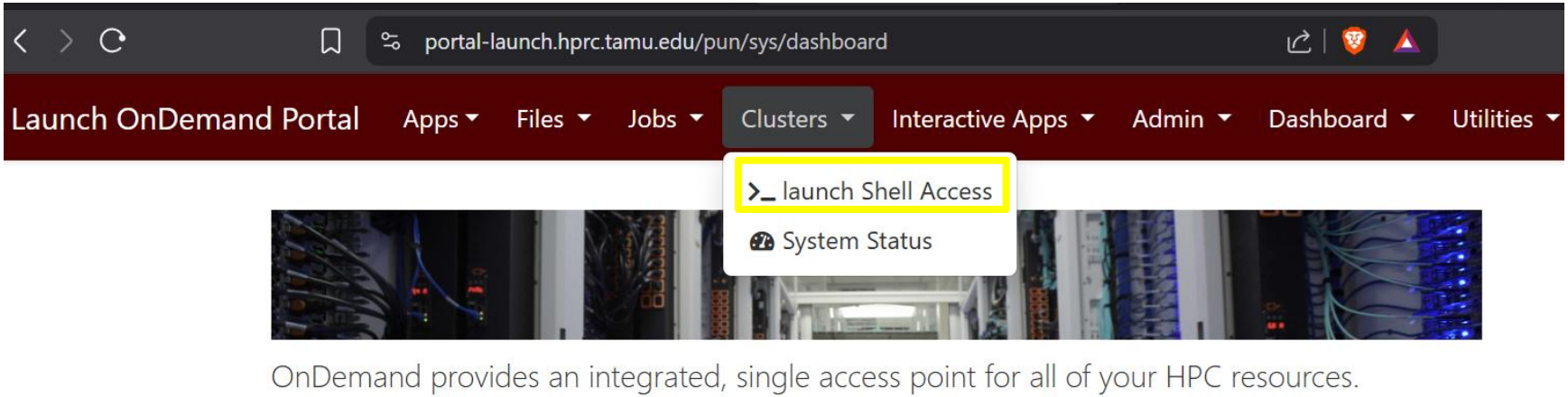
Accessing Grace: via HPRC Portal

- HPRC homepage: hprc.tamu.edu
- Select ‘Grace Portal’ in Portal tab dropdown:



Opening a Terminal

- Navigate to Clusters menu dropdown
- Click Shell Access



Persistent Terminal: tmux

Your terminal connection will be disconnected once in a while. To avoid this intermittent disconnection, you can use tmux to create a persistent terminal that you can reconnect if you get disconnected.
First, take note of which login node you are on. For example:

```
[u.th284212@launch-login2 ~]$
```

This means I'm in Launch's login2. To find out the host name, type:

```
hostname -s
```

```
llogin2
```

Launch's login #2 has host name `llogin2`. You will need to make sure you reconnect to the right login node first, before reconnecting to the corresponding tmux session.

Persistent Terminal: tmux

To start a persistent terminal via tmux, type:

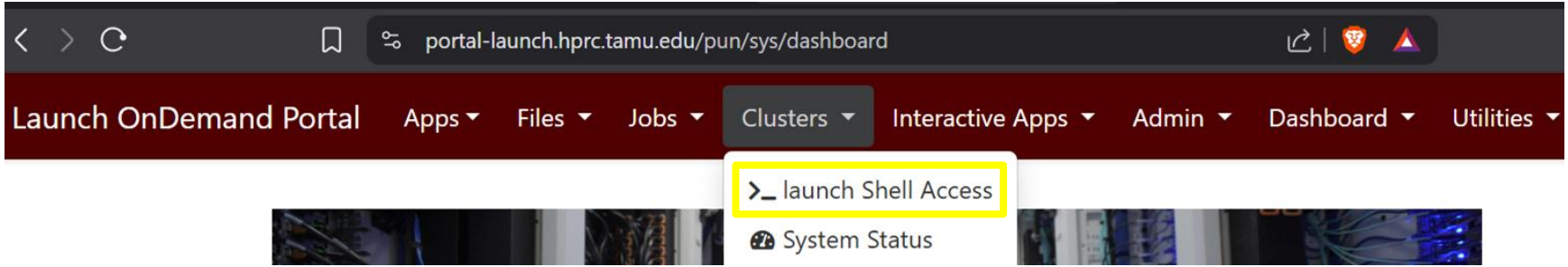
```
tmux
```

You will notice a green bar at the bottom of your terminal screen

```
[0] 0:bash*                    "llogin2.launch" 09:42 08-Oct-25
```

If you get disconnected:

First, open a terminal shell from the Cluster drop-down menu:



Then, notice the beginning of your terminal prompt. If it shows the same login node you were on before, you are good. If not, you need to first connect back to the login node you were previously on, e.g.:

```
ssh llogin2
```

If you get disconnected:

Next, check if you have a tmux session running:

```
tmux ls
```

```
0: 1 windows (created Wed Oct  8 09:42:14 2025)
```

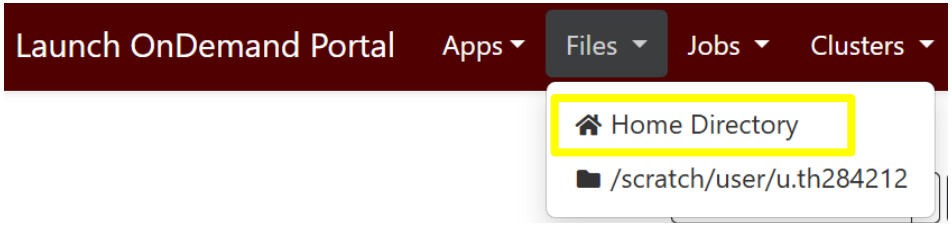
This means you had a tmux session #0 on it. To reconnect to this tmux session, type:

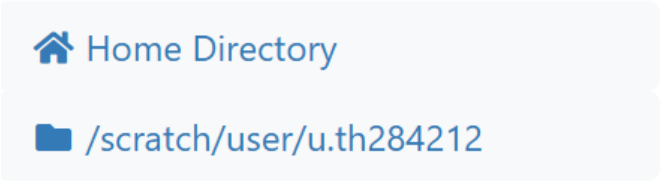
```
tmux a
```

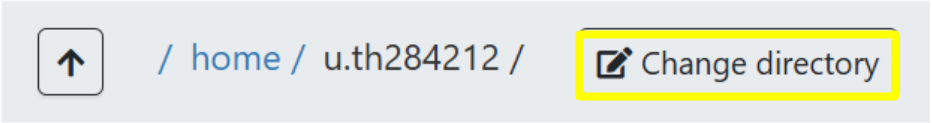
And you should be back to where you were before.

List of commands for this course

If you have trouble typing the commands listed in this course's slide, you can view them in:
`/scratch/training/software_dev_env/list_of_commands.txt`







List of commands for this course

Change Directory

Path

/scratch/training/software_dev_env

OK Cancel

/ scratch / training / software_dev_env / Change directory

Show Owner/Mode Show Dotfiles

Type	Name	Size
	environment.yml	89.00 B
	list_of_commands.txt	357.00 B
	requirements.txt	15.00 B
	wannier90_tutorial02.slurm	787.00 B

Setting Up Environments for:

- Running Python scripts/notebooks
  - ⇨ Using ModuLair
  - Using Mamba/Conda
- Running R scripts
- Compiling software
  - Using GNU (FOSS) toolchain
  - Using Intel toolchain

Identifying Required Python Packages

```
import os
import sys
Import time
import torch.nn as nn
```

os, *sys*, *time*: built-in packages, no need to install
torch: require set up

Searching PyTorch Modules

```
ml spider PyTorch
```

```
PyTorch/1.12.0-CUDA-11.7.0
PyTorch/1.12.1-CUDA-11.4.1
PyTorch/1.12.1-CUDA-11.7.0
PyTorch/1.13.1
PyTorch/2.0.1
PyTorch/2.1.2-CUDA-12.1.1
PyTorch/2.1.2
PyTorch/2.7.0
```

So many modules, which one to choose?
Need to explore further…

Getting a PyTorch Module's Toolchain

```
ml spider PyTorch/2.7.0
```

```
You will need to load all module(s) on any one of the lines below
before the "PyTorch/2.7.0" module is available to load.
      GCC/13.2.0  OpenMPI/4.1.6
```

This means *PyTorch/2.7.0* module is based on *GCC/13.2.0* toolchain.
Different PyTorch modules are based on different toolchain.
Make sure all your python modules are based on the **SAME** version of GCC/GCCcore.

Searching JupyterLab Modules

If you wish to use JupyterLab interactive app with your Python virtual environment, you will also need a compatible JupyterLab module.

```
ml spider JupyterLab
```

```
JupyterLab/3.5.0
JupyterLab/4.0.3
JupyterLab/4.0.5
JupyterLab/4.2.0
```

Which one is compatible with PyTorch/2.7.0?

Getting a Compatible JupyterLab Module

First, load PyTorch/2.7.0:

```
ml GCC/13.2.0 OpenMPI/4.1.6 PyTorch/2.7.0
```

Then, type:

```
ml avail JupyterLab
```

```
JupyterLab/4.2.0
```

This means JupyterLab/4.2.0 is compatible and can be loaded together with PyTorch/2.7.0

Combining All The Required Modules

And you will get the following list of modules for your environments:

```
GCC/13.2.0
OpenMPI/4.1.6
PyTorch/2.7.0
JupyterLab/4.2.0
```

ModuLair: Create a Python venv

First, purge all loaded modules:

```
ml purge
```

Type this command in one line:

```
create_venv mytorchvenv -t "GCC/13.2.0 OpenMPI/4.1.6 PyTorch/2.7.0
JupyterLab/4.2.0"
```

A new Python virtual environment folder named *mytorchvenv* will be created in your `$SCRATCH/virtual_envs` directory, along with a *metadata.json* file describing your virtual environments. Pip will also be updated.

You can edit this *metadata.json* manually to add more modules to the toolchain of the *mytorchvenv* environment if needed.

ModuLair: Activate a Python venv

To activate *mytorchvenv* Python virtual environment, type:

```
source activate_venv mytorchvenv
```

Your terminal will begin with `(mytorchvenv)`. You can then install additional packages not provided by our software modules with `pip install` commands, e.g.:

```
pip install pipreqs==0.4.13
```

Pipreqs is a simple but handy Python package to generate a requirements.txt file from the import section of your Python scripts and notebooks (unlike the command `pip freeze`, which gets package requirement info from your current environment)

Verify Installed Packages

```
pip list | grep torch
```

```
torch                                    2.7.0
```

Or you can also run just the “import torch” command:

```
python -c "import torch; print(torch.__version__)"
```

```
2.7.0+cu126
```

ModuLair: Other Commands

To deactivate a ModuLair environment, type:

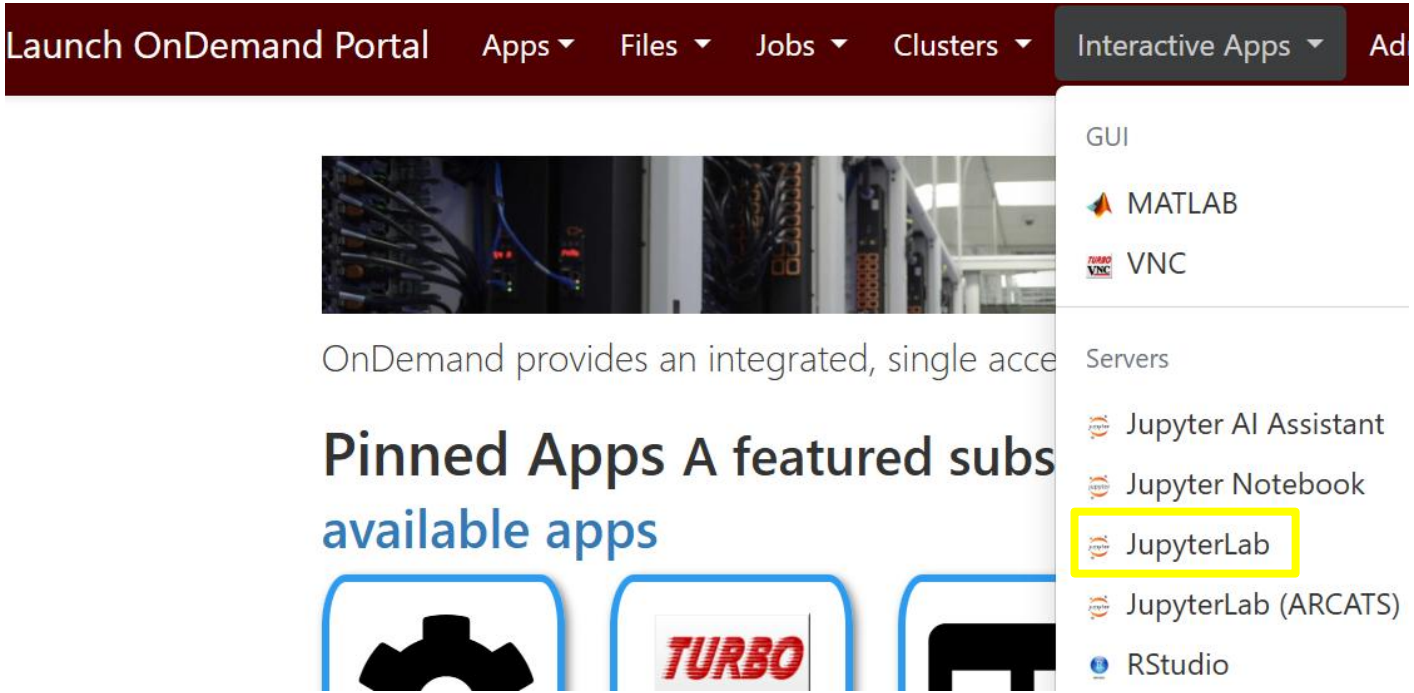
```
deactivate; ml purge
```

To list all ModuLair environments you created, type:

```
list_venvs
```

More info about ModuLair here:
https://hprc.tamu.edu/kb/Software/ModuLair

Setting up JupyterLab with ModuLair



Setting up JupyterLab with ModuLair

JupyterLab

This app will launch a JupyterLab server on the Launch cluster.

Type of environment

TAMU ModuLair (Python virtualenv manager)

Select the type of environment in which Jupyter is installed.
Help me choose

TAMU ModuLair environment (required)

mytorchvenv

Select the name of the TAMU ModuLair environment to be activated.

Your TAMU ModuLair environment is expected to have a Jupyter package installed. Please see instructions

Node type

CPU only

- cpuavail gpuavail select a non-CPU node type only if your software supports the Accelerator

Number of hours (max 48)

1

Number of cores (max 192)

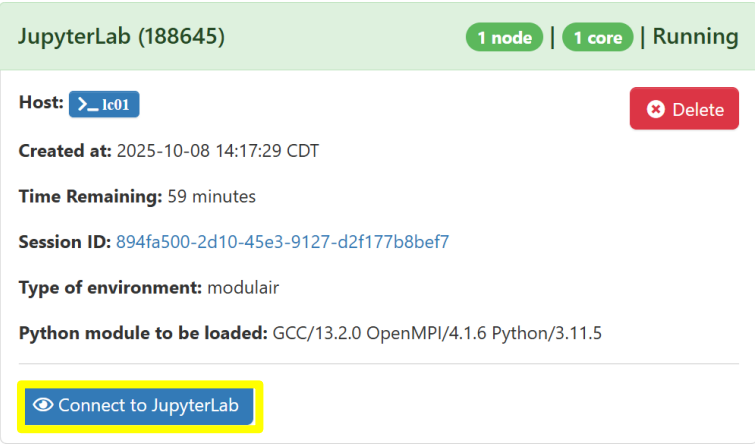
1

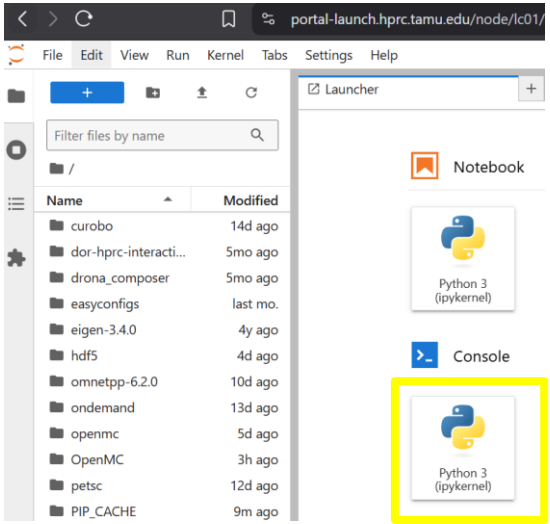
Total GB memory (max 371)

5

Launch

Verify ModuLair Environment in JupyterLab






```
import torch; print(torch.__version__)
```

```
2.7.0+cu126
```

Cleaning up a JupyterLab Session

JupyterLab (188645) 1 node | 1 core | Running

Host: >_ lc01

Delete

Created at: 2025-10-08 14:17:29 CDT

Time Remaining: 59 minutes

Session ID: 894fa500-2d10-45e3-9127-d2f177b8bef7

Type of environment: modulair

Python module to be loaded: GCC/13.2.0 OpenMPI/4.1.6 Python/3.11.5

Connect to JupyterLab

Writing a Slurm Job Script for ModuLair

https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-examples
Follow Example #1 for single-node single-core job. Key things to remember:

- Python jobs are usually limited to 1 node (unless you are running *mpi4py* or training AI/ML on multiple nodes using `torchrun`)
- Request `--mem` as the maximum amount of RAM your job need.
- Use ONE SINGLE command `source activate_venv mytorchvenv` to load required modules and activate your ModuLair venv.

Slurm Job Script Template for ModuLair

First, make sure you are in your $SCRATCH directory:

```
cd $SCRATCH
```

Now, copy my `python_modulair.slurm` job script template and `mytorchscript.py` example python script to the current directory (IMPORTANT: don't forget the last dot ".", it means "current directory")

```
cp /scratch/training/software_dev_env/python_modulair.slurm .
```

```
cp /scratch/training/software_dev_env/mytorchscript.py .
```

ModuLair Job Template Key Points

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=5G
```

Request 1 node, 1 core per node, and 5GB RAM per node

```
source activate_venv mytorchvenv
```

Load required modules and activate ModuLair venv

```
python mytorchscript.py
```

Main python command

mytorchscript.py content

```
import torch
print(torch.__version__)
```

Import torch library and print out the version of torch

Submitting a ModuLair Slurm Job

To submit the Python ModuLair job, type:

```
sbatch python_modulair.slurm
```

```
Submitted batch job 245379
```

This means the job ID is 245379, and the screen output logging file should be named `modulair.245379`. Your job ID will be different. Typically you should use `squeue --me` command to see the status of queuing/running jobs, but since this job finishes quickly, you might need to use `sacct`:

```
sacct -j 245379
```

More useful job monitoring commands here: https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-monitoring-and-control-commands

Exercise Time!

Set up and verify *myleanvenv* ModuLair environment:

- Toolchain:
  `“GCC/13.2.0 OpenMPI/4.1.6 Python/3.11.5”`
- Install *numpy* and *jupyter* via pip
- Verify the installed *numpy* package in JupyterLab
- Copy `mynumpyscript.py` from `/scratch/training/software_dev_env` to your `$SCRATCH` directory
- Modify `python_modulair.slurm` to activate *myleanvenv* and run `mynumpyscript.py`, then submit the job.

Solution for ModuLair myleanvenv

Use ModuLair to create a virtual environment requiring only a Python module and its toolchain, e.g. if you wish to use *Python/3.11.5*:

```
create_venv myleanvenv -t "GCC/13.2.0 OpenMPI/4.1.6 Python/3.11.5"
```

Then activate it:

```
source activate_venv myleanvenv
```

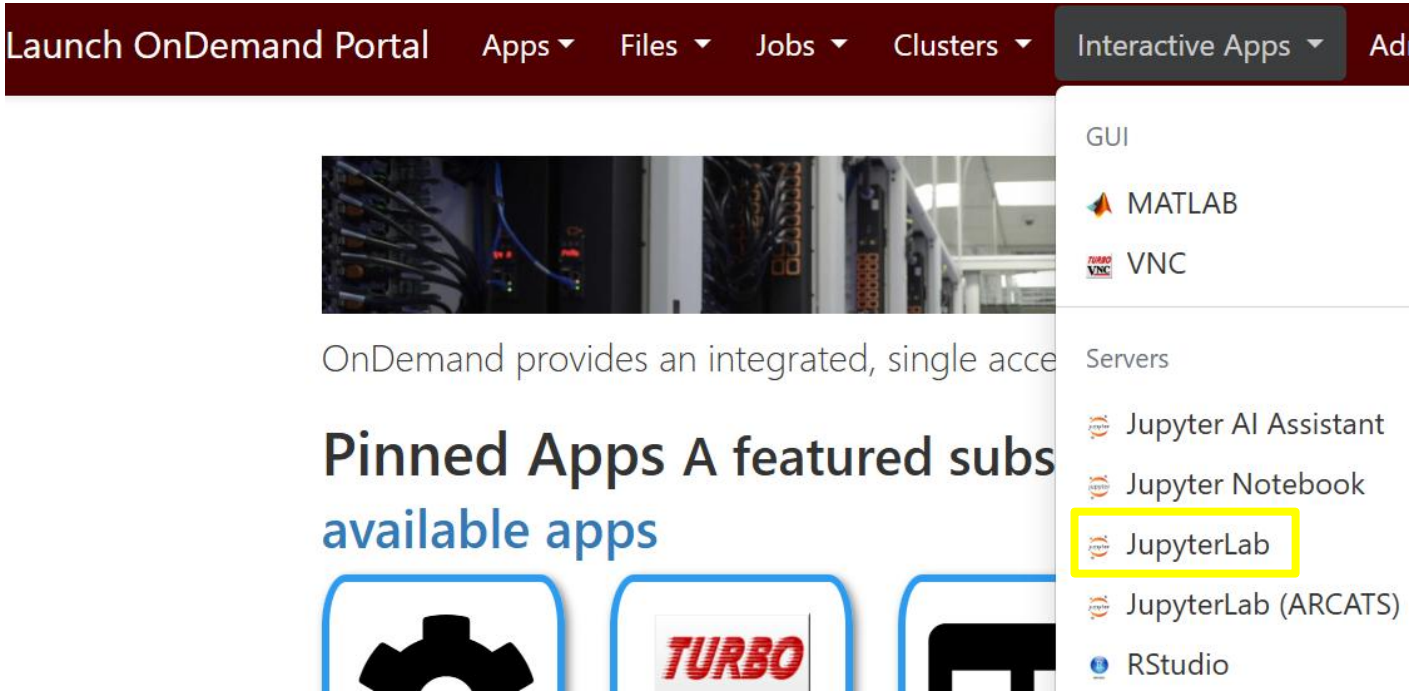
And use `pip install` command to install your required packages, e.g.:

```
pip install jupyter
```

If you have a `requirements.txt` file, follow this command's syntax:

```
pip install -r /scratch/training/software_dev_env/requirements.txt
```

Solution for ModuLair myleanvenv



Setting up JupyterLab with myleanvenv

JupyterLab

This app will launch a JupyterLab server on the Launch cluster.

Type of environment

TAMU ModuLair (Python virtualenv manager)

Select the type of environment in which Jupyter is installed.
Help me choose

TAMU ModuLair environment (required)

myleanvenv

Select the name of the TAMU ModuLair environment to be activated.

Your TAMU ModuLair environment is expected to have a Jupyter package installed. Please see instructions

Node type

CPU only

- cpuavail gpuavail select a non-CPU node type only if your software supports the Accelerator

Number of hours (max 48)

1

Number of cores (max 192)

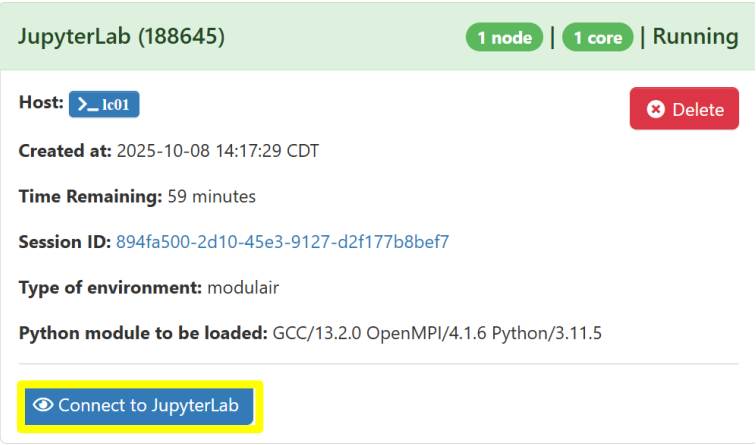
1

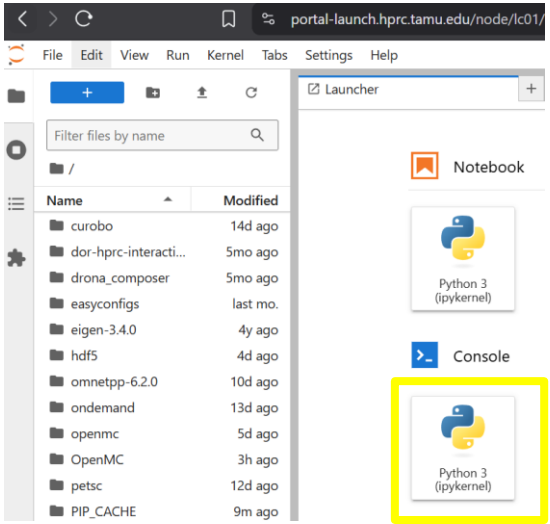
Total GB memory (max 371)

5

Launch

Verify ModuLair Environment in myleanvenv






```
import numpy as np; print(np.__version__)
```

```
1.24.1
```

Cleaning up a JupyterLab Session

JupyterLab (188645) 1 node | 1 core | Running

Host: >_ lc01

Delete

Created at: 2025-10-08 14:17:29 CDT

Time Remaining: 59 minutes

Session ID: 894fa500-2d10-45e3-9127-d2f177b8bef7

Type of environment: modulair

Python module to be loaded: GCC/13.2.0 OpenMPI/4.1.6 Python/3.11.5

Connect to JupyterLab

mynumpyscript.py template

Make sure you are in your $SCRATCH directory:

```
cd $SCRATCH
```

Now, copy `mynumpyscript.py` example python script to the current directory (IMPORTANT: don't forget the last dot ".", it means "current directory")

```
cp /scratch/training/software_dev_env/mynumpyscript.py .
```

mynumpyscript.py content

```
import numpy as np
print(np.__version__)
```

Import numpy library and print out the version of numpy

Modify ModuLair Job Script for myleanvenv

In `python_modulair.slurm`:
Change *mytorchvenv* to *myleanvenv:*

```
source activate_venv myleanvenv
```

Change *mytorchscript.py* to *mynumpyscript.py*:

```
python mynumpyscript.py
```

Submitting a ModuLair Slurm Job

To submit the Python ModuLair job, type:

```
sbatch python_modulair.slurm
```

```
Submitted batch job 245379
```

This means the job ID is 245379, and the screen output logging file should be named `modulair.245379`. Your job ID will be different. Typically you should use `squeue --me` command to see the status of queuing/running jobs, but since this job finishes quickly, you might need to use `sacct`:

```
sacct -j 245379
```

More useful job monitoring commands here: https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-monitoring-and-control-commands

Setting Up Environments for:

- Running Python scripts/notebooks
  - Using ModuLair
  - ⇨ Using Mamba/Conda
- Running R scripts
- Compiling software
  - Using GNU (FOSS) toolchain
  - Using Intel toolchain

Let’s make sure to start fresh: New tmux

Exit out of your current tmux session if you are inside one:

```
exit
```

If your terminal is terminated (no pun intended), open a new terminal and kill all existing tmux sessions:

```
tmux kill-server
```

Then start a new tmux session:

```
tmux
```

Last Resort: Mamba/Conda

We advise AGAINST using Mamba/Conda to set up your Python environment if it can be set up with pip install. This is because, compared to venv, Mamba/Conda environments contain its own Python files and some core dependencies that would **consume a significant part of your file count quota**, typically around **30000 files PER EACH Mamba/Conda environment**.

For reference, your $HOME file count limit is **10000**. This is why you will likely fill your $HOME file count quota and thus get locked out of your account if you try to set up a Mamba/Conda environment directly on your $HOME space.

Last Resort: Mamba/Conda

However, in some cases, Mamba/Conda is the only way to install a package. In this case, use Mamba to set up a conda environment, with some precautions:

- Make sure to **ONLY** load just **ONE** Mamba/Conda module and **NO OTHER MODULES**. Mamba/Conda environments contain all the needed components. Loading additional software modules will likely **BREAK** Mamba/Conda environments.
- Make sure to disable `(base)` conda environment activation
- NEVER RUN **`conda init`** or **`mamba init`**
- Use **`source activate`** instead **`mamba activate`** or **`conda activate`** to activate a Mamba/Conda environment

Disabling (base) Conda Environment

First, find available Mamba modules you can load. We recommend Mamba over Anaconda since Mamba can do the same environment setup task MUCH FASTER.

```
ml spider Mamba
```

```
Mamba/23.11.0-0
```

Load Mamba:

```
ml Mamba/23.11.0
```

Disable (base) conda environment auto-activation:

```
conda config --set auto_activate_base False
```

Setting up an srun Session

First, start an interactive srun session. DO NOT run mamba/conda installation commands directly on the login node!

```
srun --nodes=1 --ntasks-per-node=2 --mem=10G --time=01:00:00 --pty
bash -i
```

Purge all modules:

```
ml purge
```

Load Mamba module:

```
ml Mamba/23.11.0
```

Load WebProxy module to enable internet access in a srun session:

```
ml WebProxy
```

Creating a Mamba/Conda Environment

If you have an environment.yml file for your Conda environment, type:

```
mamba env create -f /scratch/training/software_dev_env/environment.yml
```

Content of the template environment.yml:

```
name: mambanumpyenv
channels:
  - conda-forge
dependencies:
  - numpy=1.23.5
  - jupyter
```

This means the command above will create a mamba environment named *mambanumpyenv* and install *numpy* version 1.23.5 and *jupyter* packages from *conda-forge* repository.

Activating a Mamba/Conda Environment

If you want to set up your Conda environment manually:

```
mamba create -n mymambaenv
```

To activate your Mamba/Conda environment:

```
source activate mymambaenv
```

Note the `source activate`, NOT `mamba activate` or `conda activate`!

Your terminal will begin with `(mymambaenv)`.

Installing Mamba/Conda Packages

To install *jupyter* and *numpy* packages from *conda-forge* repository:

```
mamba install -y -c conda-forge jupyter numpy
```

You can also use `pip install` to install PyPI packages to your Mamba/Conda environment:

```
pip install pipreqs==0.4.13
```

Verify Packages for conda env

```
mamba list | grep pipreqs
```

```
pipreqs                   0.4.13                    pypi_0    pypi
```

```
pip list | grep pipreqs
```

```
pipreqs                   0.4.13
```

Mamba/Conda: Other Commands

To deactivate a mamba/conda environment, type:

```
source deactivate
```

To list all mamba/conda environments you created, type:

```
mamba env list
```

To completely remove a mamba/conda environment and related packages, type:

```
mamba remove --name mymambaenv --all -y
```

More info about mamba/conda here:
https://hprc.tamu.edu/kb/Software/ANACONDA

Cleaning Up Your srun Session

To exit out of the current srun session, type this command ONCE:

```
exit
```

Your terminal should show that you're back to the login node you were on.

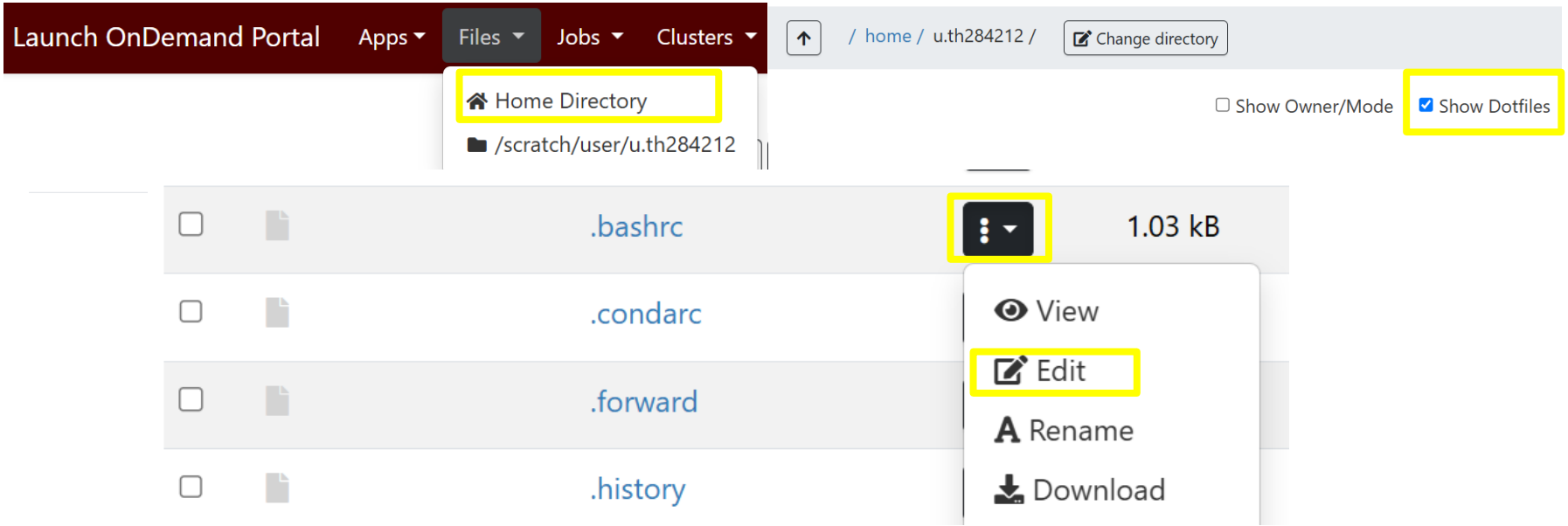
If you ran conda/mamba init:

Type:

```
conda init --reverse --all
```

If you ran conda/mamba init:

OR: modify your ~/.bashrc file and remove the entire section of conda initialization commands. Then, log out and reopen a terminal again for the change to take effect.



Edit ~/.bashrc and remove conda block:

Save

Key Bindings

Default

Font Size

12px

Mode

Text

Theme

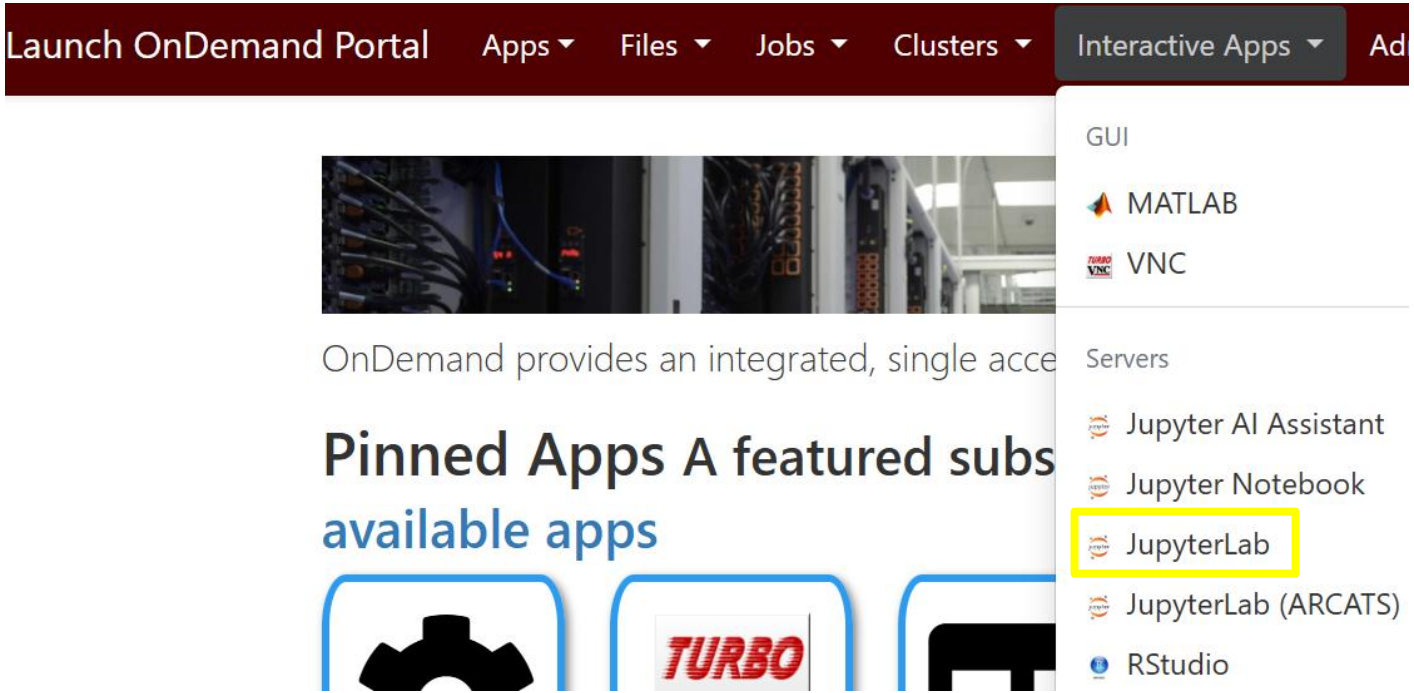
krTheme

Wrap

```
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/ztank/sw/eb/sw/Mamba/23.11.0-0/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/ztank/sw/eb/sw/Mamba/23.11.0-0/etc/profile.d/conda.sh" ]; then
        . "/ztank/sw/eb/sw/Mamba/23.11.0-0/etc/profile.d/conda.sh"
    else
        export PATH="/ztank/sw/eb/sw/Mamba/23.11.0-0/bin:$PATH"
    fi
fi
unset __conda_setup

if [ -f "/ztank/sw/eb/sw/Mamba/23.11.0-0/etc/profile.d/mamba.sh" ]; then
    . "/ztank/sw/eb/sw/Mamba/23.11.0-0/etc/profile.d/mamba.sh"
fi
# <<< conda initialize <<<
```

Setting up JupyterLab with Conda



Setting up JupyterLab with Conda

JupyterLab

This app will launch a JupyterLab server on the Launch cluster.

Type of environment

Anaconda environment

Select the type of environment in which Jupyter is installed.
Help me choose

Conda/Mamba module to be loaded

Anaconda3/2024.02-1

Select a Conda/Mamba module to load.

Optional conda/mamba environment to be activated

mymambaenv

Enter the name of the conda environment to be activated. This field is optional.

Node type

CPU only

- cpuavail gpuavail select a non-CPU node type only if your software supports the Accelerator

Number of hours (max 48)

1

Number of cores (max 192)

1

Total GB memory (max 371)

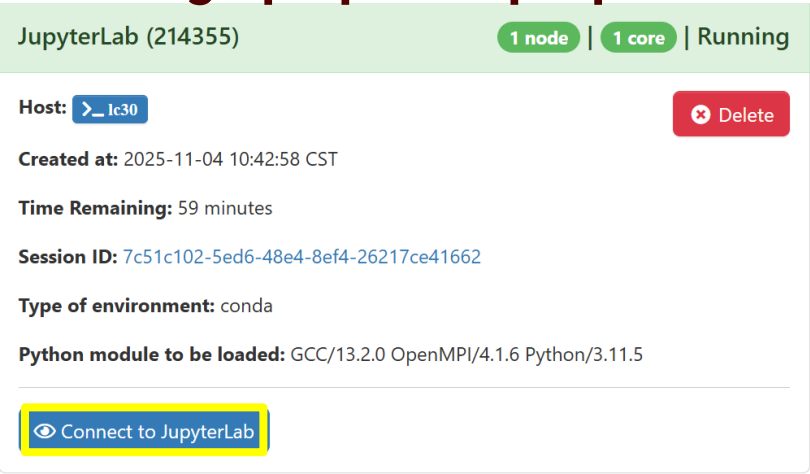
5

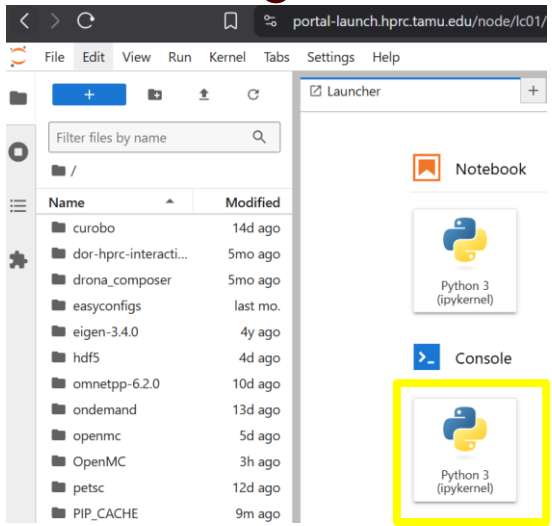
Launch

Exercise time!

- Verify pipreqs package in *mymambaenv* conda environment in JupyterLab
- Verify numpy package in *mambanumpyenv* conda environment in JupyterLab

Verify pipreqs package in mymambaenv

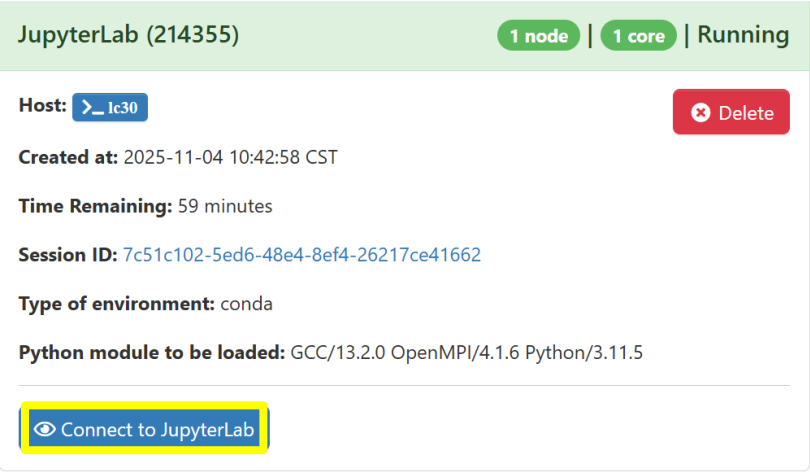


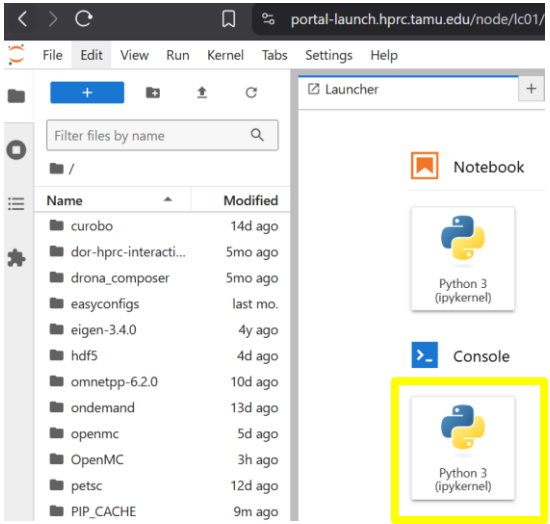



```
!pip list | grep pipreqs
```

```
pipreqs                    0.4.13
```

Verify numpy package in mambanumpyenv






```
import numpy as np; print(np.__version__)
```

```
1.23.5
```

Cleaning up a JupyterLab Session

JupyterLab (214355) 1 node | 1 core | Running

Host: >_ lc30

Delete

Created at: 2025-11-04 10:42:58 CST

Time Remaining: 59 minutes

Session ID: 7c51c102-5ed6-48e4-8ef4-26217ce41662

Type of environment: conda

Python module to be loaded: GCC/13.2.0 OpenMPI/4.1.6 Python/3.11.5

Connect to JupyterLab

Writing a Slurm Job Script for Mamba

https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-examples
Follow Example #1 for single-node single-core job. Key things to remember:

- Python jobs are usually limited to 1 node (unless you are running *mpi4py* or training AI/ML on multiple nodes using `torchrun`)
- Request `--mem` as the maximum amount of RAM your job need.
- Purge and load Mamba module, then use **`source activate`** command to activate your mamba environment.

Slurm Job Script Template for Mamba

First, make sure you are in your $SCRATCH directory:

```
cd $SCRATCH
```

Now, copy my `python_mamba.slurm` job script template to the current directory (IMPORTANT: don't forget the last dot ".", it means "current directory")

```
cp /scratch/training/software_dev_env/python_mamba.slurm .
```

Mamba Job Template Keypoints

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=5G
```

Request 1 node, 1 core per node, and 5GB RAM per node

```
ml purge
ml Mamba/23.11.0
source activate mambanumpyenv
```

Purge and load a Mamba module, then activate a mamba environment

```
python mynumpyscript.py
```

Main python command

Submitting a Mamba Slurm Job

To submit the Python Mamba job, type:

```
sbatch python_mamba.slurm
```

```
Submitted batch job 245394
```

This means the job ID is 245394, and the screen output logging file should be named `mamba.245394`. Your job ID will be different. Typically you should use `squeue --me` command to see the status of queuing/running jobs, but since this job finishes quickly, you might need to use `sacct`:

```
sacct -j 245394
```

More useful job monitoring commands here: https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-monitoring-and-control-commands

Setting Up Environments for:

- Running Python scripts/notebooks
  - Using ModuLair
  - Using Mamba/Conda
- Running R scripts
- Compiling software
  - Using GNU (FOSS) toolchain
  - Using Intel toolchain

Let’s make sure to start fresh: New tmux

Exit out of your current tmux session if you are inside one:

```
exit
```

If your terminal is terminated (no pun intended), open a new terminal and kill all existing tmux sessions:

```
tmux kill-server
```

Then start a new tmux session:

```
tmux
```

Identifying Required R Packages

```
library(datasets)
library(sf)
library(sfExtras)
```

It is a lot harder to identify which R libraries are built-in and which needs to be installed based on the libraries' names.

R_tamu

A specialized module designed by TAMU HPRC, containing prebuilt packages in:

- "Barebone" R module
- R-bundle-CRAN: including many commonly used packages in the **C**omprehensive **R** **A**rchive **N**etwork (**CRAN**) repository: https://github.com/easybuilders/easybuild-easyconfigs/blob/develop/easybuild/easyconfigs/r/R-bundle-CRAN/R-bundle-CRAN-2024.11-foss-2024a.eb
- R-bundle-Bioconductor: including many commonly used packages for bioinformatics: https://github.com/easybuilders/easybuild-easyconfigs/blob/develop/easybuild/easyconfigs/r/R-bundle-Bioconductor/R-bundle-Bioconductor-3.20-foss-2024a-R-4.4.2.eb

Browsing R_tamu Modules

```
ml spider R_tamu
```

```
R_tamu/4.2.1
R_tamu/4.3.2
R_tamu/4.4.1
R_tamu/4.4.2
```

```
ml spider R_tamu/4.4.2
```

```
You will need to load all module(s) on any one of the lines below
before the "R_tamu/4.4.2" module is available to load.
      GCC/13.3.0  OpenMPI/5.0.3
```

Loading R_tamu

```
ml GCC/13.3.0 OpenMPI/5.0.3 R_tamu/4.4.2
```

Once done, you can list the loaded modules to have a feel of how big R_tamu is:

```
ml
```

```
  1) GCCcore/13.3.0        23) libpng/1.6.43        45) pixman/0.43.4           67) GSL/2.8                89) libarchive/3.7.4            111) Qhull/2020.2
  2) zlib/1.3.1            24) Brotli/1.1.0         46) libiconv/1.17           68) GMP/6.3.0              90) PCRE/8.45                   112) LERC/4.0.0
  3) binutils/2.42         25) freetype/2.13.2      47) gettext/0.22.5          69) NLopt/2.7.1            91) nlohmann_json/3.11.3        113) OpenJPEG/2.5.2
  4) GCC/13.3.0            26) ncurses/6.5          48) PCRE2/10.43             70) libogg/1.3.5           92) PROJ/9.4.1                  114) SWIG/4.2.1
  5) numactl/2.0.18        27) libreadline/8.2      49) GLib/2.80.4             71) FLAC/1.4.3             93) libgeotiff/1.7.3            115) GDAL/3.10.0
  6) XZ/5.4.5              28) Tcl/8.6.14           50) cairo/1.18.0            72) libvorbis/1.3.7        94) cffi/1.16.0                 116) MPFR/4.2.1
  7) libxml2/2.12.7        29) SQLite/3.45.3        51) NASM/2.16.03            73) libopus/1.5.2          95) cryptography/42.0.8         117) PostgreSQL/16.4
  8) libpciaccess/0.18.1   30) util-linux/2.40      52) libjpeg-turbo/3.0.1     74) LAME/3.100             96) virtualenv/20.26.2          118) R-bundle-CRAN/2024.11
  9) hwloc/2.10.0          31) fontconfig/2.15.0    53) jbigkit/2.1             75) libsndfile/1.2.2       97) Python-bundle-PyPI/2024.06  119) snappy/1.2.1
 10) OpenSSL/3             32) xorg-macros/1.20.1   54) libdeflate/1.20         76) Szip/2.1.1             98) SciPy-bundle/2024.05        120) RapidJSON/1.1.0-20240815
 11) libevent/2.1.12       33) X11/20240607         55) LibTIFF/4.6.0           77) HDF5/1.14.5            99) libtirpc/1.3.5              121) Abseil/20240722.0
 12) UCX/1.16.0            34) gzip/1.13            56) Java/17 -> Java/17.0.4  78) UDUNITS/2.2.28        100) HDF/4.3.0                   122) RE2/2024-07-02
 13) libfabric/1.21.0      35) lz4/1.9.4            57) libgit2/1.8.1           79) Ghostscript/10.03.1   101) Eigen/3.4.0                  123) utf8proc/2.9.0
 14) PMIx/5.0.2            36) zstd/1.5.6           58) cURL/8.7.1              80) JasPer/4.2.4          102) arpack-ng/3.9.1              124) Arrow/17.0.0
 15) PRRTE/3.0.5           37) libdrm/2.4.122       59) Tk/8.6.14               81) LittleCMS/2.16        103) Armadillo/14.0.3             125) arrow-R/17.0.0.1-R-4.4.2
 16) UCC/1.3.0             38) libglvnd/1.7.0       60) ICU/75.1                82) Pango/1.54.0          104) CFITSIO/4.4.1                126) R-bundle-Bioconductor/3.20-R-4.4.2
 17) OpenMPI/5.0.3         39) libunwind/1.8.1      61) HarfBuzz/9.0.0          83) ImageMagick/7.1.1-38  105) giflib/5.2.1                 127) Pandoc/3.6.2
 18) OpenBLAS/0.3.27       40) LLVM/18.1.8-minimal  62) FriBidi/1.0.15          84) GLPK/5.0              106) json-c/0.17                  128) Rust/1.78.0
 19) FlexiBLAS/3.4.4       41) libffi/3.4.5         63) R/4.4.2                 85) nodejs/20.13.1        107) Xerces-C++/3.2.5             129) R_tamu/4.4.2
 20) FFTW/3.3.10           42) Wayland/1.23.0       64) FFTW.MPI/3.3.10         86) Python/3.12.3         108) Imath/3.1.11
 21) bzip2/1.0.8           43) Mesa/24.1.3          65) ScaLAPACK/2.2.0-fb      87) netCDF/4.9.2          109) OpenEXR/3.2.4
 22) expat/2.6.2           44) libGLU/9.0.3         66) Boost/1.85.0            88) GEOS/3.12.2           110) Brunsli/0.1
```

Creating an R Project Environment:

```
R --rtamuenvs=myrenv --vanilla
```

This will create an R project environment directory named myrenv and set the default library installation path to:
`$SCRATCH/R_tamuenvs/4.4.2-gfbf-2024a/myrenv/`

Once you're inside the R prompt, try loading various R packages, such as *datasets*, *sf*, and *sfExtras*. When it's *sfExtras*:

```
library(sfExtras)
```

```
Error in library(sfExtras) : there is no package called 'sfExtras'
```

This means even the mighty R_tamu module does NOT have R library "sfExtras" yet. Time to install it!

Installing sfExtras to myrenv

https://github.com/spatialanalysis/sfExtras

This is an extension of R *sf* library that add more functionality to sf package for spatial autocorrelation. To install it, type:

```
remotes::install_github("spatialanalysis/sfExtras")
```

Hit Enter to skip updating existing libraries.

Once done, try loading sfExtras library again. You should see no more error (i.e. it just returns nothing). Finally, quit the R prompt by typing:

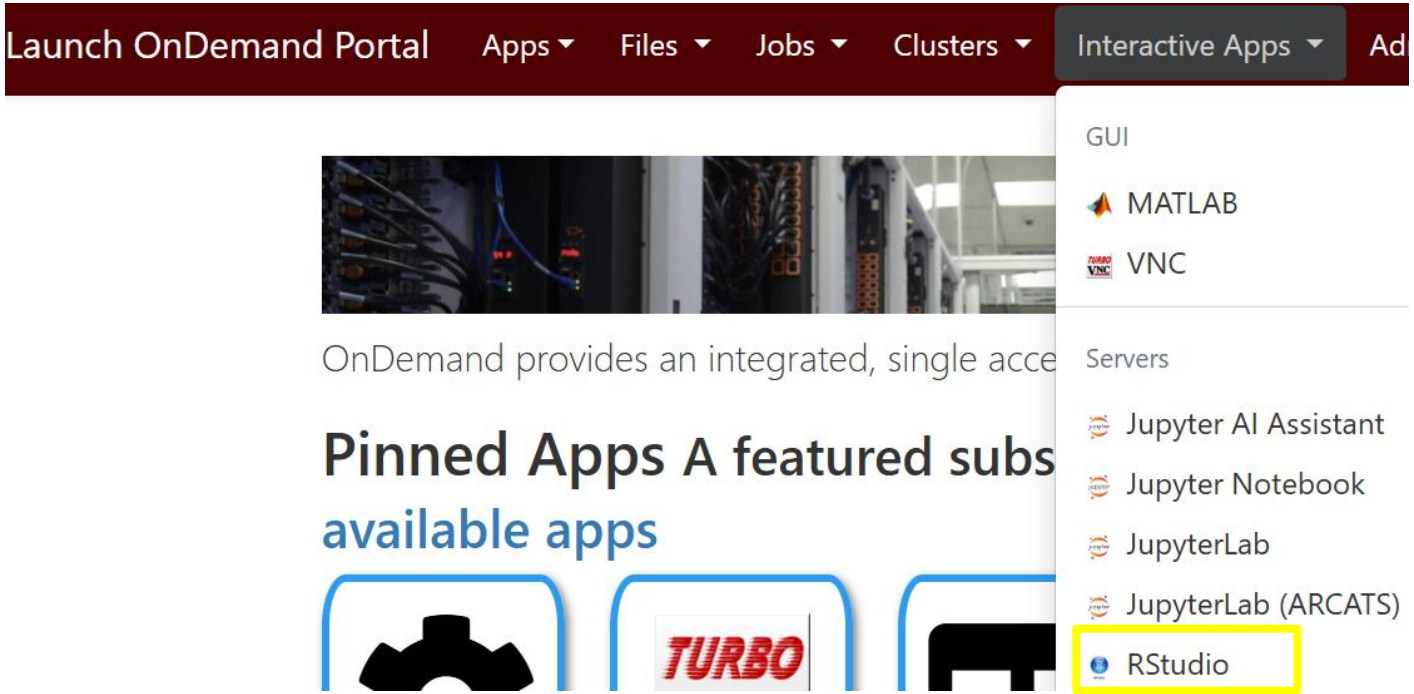
```
q()
```

To see the R libraries you installed in your R project environment, simply list the content of your R project environment directory:

```
ls $SCRATCH/R_tamuenvs/4.4.2-gfbf-2024a/myrenv/
```

```
sfExtras
```

Setting up RStudio



Setting up RStudio

RStudio R 4.1.0+ version: 2023.09.1-494

This app will launch RStudio Server with Singularity and the R_tamu software module on a compute node.

You can install your own R packages directly within RStudio.

R version

R/4.4.2

- 4.4.2 and 4.4.1 also loads the following modules but 4.3.2 and 4.2.1 do not.
  - R-bundle-Bioconductor, R-bundle-CRAN, arrow-R

Number of hours (max 48)

1

Number of cores (max 192)

1

Total GB memory (max 371)

7

Launch

Checking .libPaths() in a RStudio Session

RStudio R 4.1.0+ (17012831) 1 node | 1 core | Running

Host: c273

Delete

Created at: 2025-11-05 13:57:47 CST

Time Remaining: 56 minutes

Session ID: 7d241b77-692f-4350-8931-45d0138fb82d

R version: GCC/13.3.0 OpenMPI/5.0.3 R_tamu/4.4.2

Connect to RStudio Server

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> .libPaths()
```

```
.libPaths()
```

```
[1] "/scratch/user/thangha/project/R/4.4“
[2] "/sw/hprc/sw/dor-hprc-tools-Rtamu/R_LIBS/4.4.2-gfbf-2024a“
[3] "/sw/eb/sw/R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2“
[4] "/sw/eb/sw/arrow-R/17.0.0.1-foss-2024a-R-4.4.2“
[5] "/sw/eb/sw/R-bundle-CRAN/2024.11-foss-2024a“
[6] "/sw/eb/sw/R/4.4.2-gfbf-2024a/lib64/R/library"
```

Adding myrenv to .libPaths()

```
library(sfExtras)
```

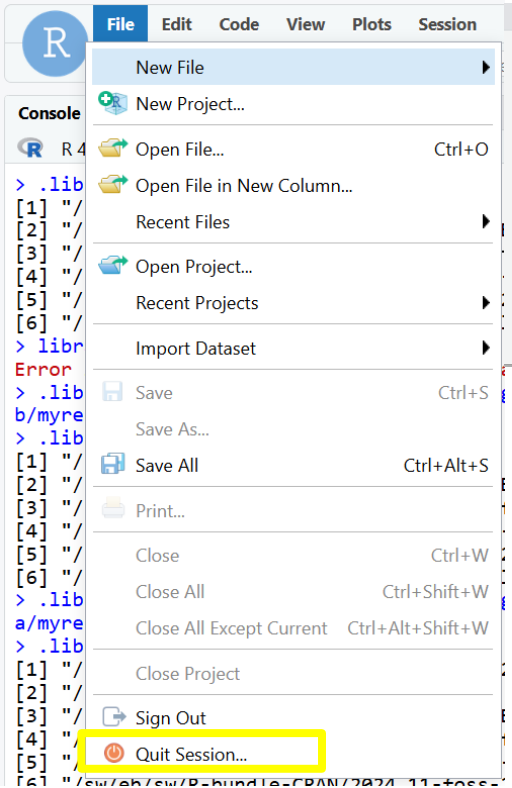
```
Error in library(sfExtras) : there is no package called ‘sfExtras’
```

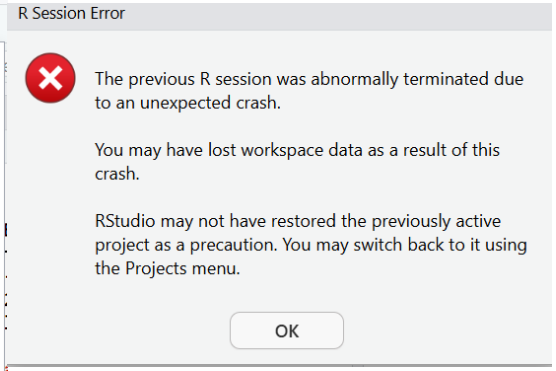
```
.libPaths(c(paste0("/scratch/user/",Sys.getenv("USER"),"/R_tamuenvs/4.4.2-gfbf-2024a/myrenv"), .libPaths()))
```

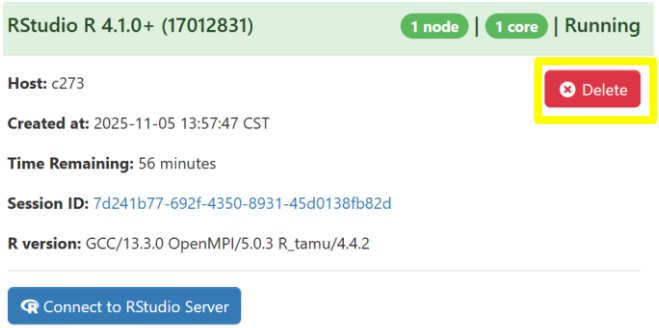
```
.libPaths()
```

```
[1] "/scratch/user/thangha/R_tamuenvs/4.4.2-gfbf-2024a/myrenv"
[2] "/scratch/user/thangha/project/R/4.4"
[3] "/sw/hprc/sw/dor-hprc-tools-Rtamu/R_LIBS/4.4.2-gfbf-2024a"
[4] "/sw/eb/sw/R-bundle-Bioconductor/3.20-foss-2024a-R-4.4.2"
[5] "/sw/eb/sw/arrow-R/17.0.0.1-foss-2024a-R-4.4.2"
[6] "/sw/eb/sw/R-bundle-CRAN/2024.11-foss-2024a"
[7] "/sw/eb/sw/R/4.4.2-gfbf-2024a/lib64/R/library"
```

Cleaning up Your RStudio Session







Writing a Slurm Job Script for R

https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-examples
Follow Example #1 for single-node single-core job. Key things to remember:

- Usually R jobs can only run on one single node (unless you are using *Rmpi* package)
- Load required modules (`GCC/13.3.0 OpenMPI/5.0.3 R_tamu/4.4.2`)
- Use the `--rtamuenvs=myrenv` flag with `Rscript` command to specify your R project environment you wish to use.

Slurm Job Script Template for R

First, make sure you are in your $SCRATCH directory:

```
cd $SCRATCH
```

Now, copy my `rtamu.slurm` job script template and `myrscript.R` example R script to the current directory (IMPORTANT: don't forget the last dot ".", it means "current directory")

```
cp /scratch/training/software_dev_env/rtamu.slurm .
```

```
cp /scratch/training/software_dev_env/myrscript.R .
```

R Job Template Keypoints

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=5G
```

Request 1 node, 1 core per node, and 5GB RAM per node

```
ml purge
ml GCC/13.3.0 OpenMPI/5.0.3 R_tamu/4.4.2
```

Purge, then load required modules

```
Rscript --rtamuenvs=myrenv myrscript.R
```

Main Rscript command with `--rtamuenvs=myrenv` flag

myrscript.R

```
.libPaths()
library(sfExtras)
packageVersion("sfExtras")
```

Print R library paths
Load sfExtras library
Print version of sfExtras library

Submitting a R Slurm Job

To submit the rtamu job, type:

```
sbatch rtamu.slurm
```

```
Submitted batch job 228574
```

This means the job ID is 228574, and the screen output logging file should be named `rtamu.228574`. Your job ID will be different. Typically you should use `squeue --me` command to see the status of queuing/running jobs, but since this job finishes quickly, you need to use `sacct`:

```
sacct -j 228574
```

More useful job monitoring commands here: https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-monitoring-and-control-commands

Setting Up Environments for:

- Running Python scripts/notebooks
  - Using ModuLair
  - Using Mamba/Conda
- Running R scripts
- Compiling software
  - ⇨ Using GNU (FOSS) toolchain
  - Using Intel toolchain

Let’s make sure to start fresh: New tmux

Exit out of your current tmux session if you are inside one:

```
exit
```

If your terminal is terminated (no pun intended), open a new terminal and kill all existing tmux sessions:

```
tmux kill-server
```

Then start a new tmux session:

```
tmux
```

Wannier90 Project

https://github.com/wannier-developers/wannier90

It is the official source code repository for a program to compute maximally-localized Wannier functions (MLWFs), in order to determine advanced electronic properties of different materials.

To obtain the source code of wannier90, first change directory to your $SCRATCH:

```
cd $SCRATCH
```

Then use git to clone the wannier90 repository:

```
git clone https://github.com/wannier-developers/wannier90.git
```

Wannier90 Compile Instruction

https://github.com/wannier-developers/wannier90/blob/develop/README.install

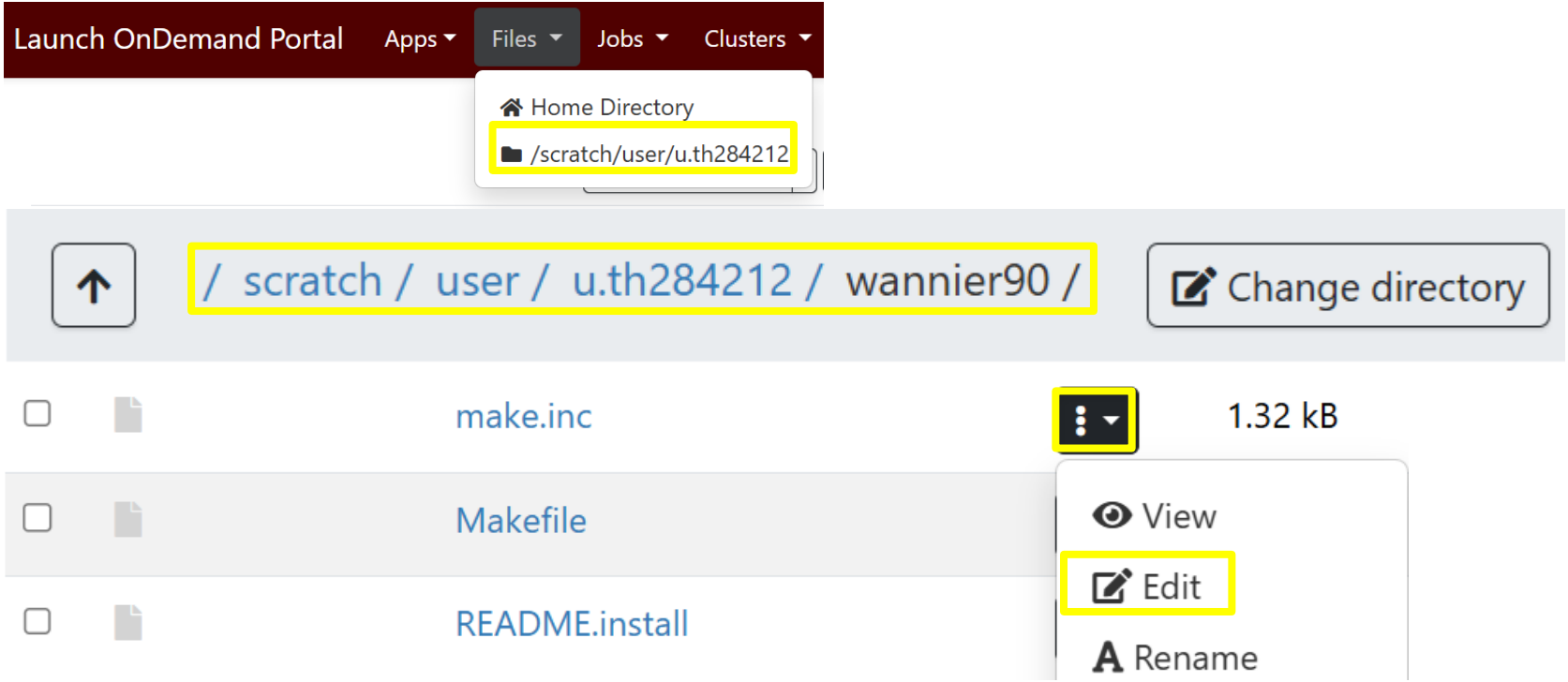
According to the instruction, we first need to copy a template for `make.inc` based on which compiler we use. This is a file that will be loaded (i.e. "included" during the compilation process). For FOSS (GNU) toolchain, we will use the template `make.inc.gfort.dynlib`.
First, enter wannier90 directory:

```
cd $SCRATCH/wannier90
```

Then, copy the template:

```
cp ./config/make.inc.gfort.dynlib ./make.inc
```

Editing FOSS make.inc



A Quick Glance of FOSS make.inc

This line tells the name of FOSS fortran compiler:

```
F90 = gfortran
```

These lines tell the compiler options, or “flags”, for compilation and linking, respectively:

```
FCOPTS = -O3 -fPIC
LDOPTS = -fPIC
```

Editing FOSS make.inc

Uncomment these lines (i.e. delete the "#" at the beginning of the lines) to enable MPI parallelization for a wannier90 module, and change mpgfortran to mpifort (which is the correct name for FOSS MPI fortran compiler):

```
#COMMS  = mpi
#MPIF90 = mpifort #mpif90
```

Finally, change LAPACK and BLAS library linking name to ScaLAPACK and FlexiBLAS:

```
LIBS = -lscalapack -lflexiblas
```

Why ScaLAPACK and FlexiBLAS?

EasyBuild FOSS toolchain uses slightly different mathematic computation libraries than regular LAPACK (Linear Algebra PACKage) and BLAS (Basic Linear Algebra Subprograms) as usually provided by a Linux OS:

- ScaLAPACK enables MPI parallelization for LAPACK to "scale" on multiple machines in an HPC
- FlexiBLAS allows a program to switch between different BLAS "engines" (e.g. Intel MKL, OpenBLAS) at runtime (i.e. without the need to recompile the source code)

Loading a FOSS Toolchain

To list available FOSS toolchains, type:

```
toolchains foss
```

```
foss/2024a   = GCC/13.3.0 & OpenMPI/5.0.3 | Python/3.12.3
foss/2024.05 = GCC/13.3.0 & OpenMPI/5.0.3 | Python/3.12.3
foss/2023b   = GCC/13.2.0 & OpenMPI/4.1.6 | Python/3.11.5
foss/2023a   = GCC/12.3.0 & OpenMPI/4.1.5 | Python/3.11.3
foss/2022b   = GCC/12.2.0 & OpenMPI/4.1.4 | Python/3.10.8
foss/2022a   = GCC/11.3.0 & OpenMPI/4.1.4 | Python/3.10.4
```

In this short course, we will be loading foss/2024a:

```
ml foss/2024a
```

Listing Components of a FOSS Toolchain

```
ml
```

```
Currently Loaded Modules:
  1) GCCcore/13.3.0          9) hwloc/2.10.0       17) OpenMPI/5.0.3
  2) zlib/1.3.1             10) OpenSSL/3          18) OpenBLAS/0.3.27
  3) binutils/2.42          11) libevent/2.1.12    19) FlexiBLAS/3.4.4
  4) GCC/13.3.0             12) UCX/1.16.0         20) FFTW/3.3.10
  5) numactl/2.0.18         13) libfabric/1.21.0   21) FFTW.MPI/3.3.10
  6) XZ/5.4.5               14) PMIx/5.0.2         22) ScaLAPACK/2.2.0-fb
  7) libxml2/2.12.7         15) PRRTE/3.0.5        23) foss/2024a
  8) libpciaccess/0.18.1    16) UCC/1.3.0
```

Clean and Build It!

For best practice, make sure to clean the build first:

```
make veryclean
```

And then build the program:

```
make all
```

It should take about 1 minute to clean and just a bit more than 1 minute to compile all wannier90 components. Once done, type:

```
ls
```

And you should see 4 executable files as the result of the build: `wannier90.x`, `w90chk2chk.x`, `w90spn2spn.x`, and `postw90.x`

Build Configurations

Typically, the file Makefile of a repository specifies different build configurations to be invoked with the make command:

- **clean**: remove all intermediately compiled objects, executable files, and libraries
- **all**: build everything
- **install**: copy the compiled executable files and libraries to a designated place (usually default to `/usr/bin` and `/usr/lib`, which will fail because modifying these locations requires sudo privilege)

For wannier90, the **clean** configuration will not remove the final compiled executable files. Instead, **veryclean** will do that, as well as cleaning up test suites, ensuring a like-new repository condition.

Quick Test-Run

WARNING! We do NOT recommend testing computationally intensive programs directly on the terminal. However, we do recognize the need to quickly test-run a program before you can properly write a job script to submit a batch job later. Please make sure these test runs use **less than 8 parallel threads** and preferably **finishes within minutes**.

To test-run wannier90, change directory to tutorial02:

```
cd $SCRATCH/wannier90/tutorials/tutorial02
```

Then type:

```
mpirun -np 2 $SCRATCH/wannier90/wannier90.x lead
```

This will run wannier90.x with 2 parallel threads, and should return within 2-3 seconds without any message. You should be able to find `lead.chk` and `lead.wout` in the same directory afterward.

Writing a Slurm Job Script for MPI

https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-examples
Follow Example #2 for single-node multi-core job, or Example #3 if you wish to run an MPI program across multiple nodes. Key things to remember:

- Request `--ntasks` as the total number of parallel threads
- Load required modules (e.g. foss/2024a)
- No need to use `-np` flag for mpirun command - Slurm will handle number of parallel threads automatically based on `--ntasks` and `--ntasks-per-node`

Wannier90 Template Slurm Job Script

First, make sure you are inside tutorial02 directory:

```
cd $SCRATCH/wannier90/tutorials/tutorial02
```

Now, copy my wannier90 template to the current directory
(IMPORTANT: don't forget the last dot ".", it means "current directory")

```
cp /scratch/training/software_dev_env/wannier90_tutorial02.slurm .
```

Clean up previously generated lead.chk and lead.wout results:

```
rm lead.chk lead.wout
```

Wannier90 Job Template Keypoints

```
#SBATCH --nodes=1
#SBATCH --ntasks=2
#SBATCH --ntasks-per-node=2
```

Request 1 node, 2 parallel threads per each node

```
ml purge
ml foss/2024a
```

Purge all modules, then load only the required ones

```
mpirun $SCRATCH/wannier90/wannier90.x lead
```

Main command, notice there is no “-np” flag for mpirun

Submitting a wannier90 Slurm Job

To submit the wannier90 tutorial02 job, type:

```
sbatch wannier90_tutorial02.slurm
```

```
Submitted batch job 199852
```

This means the job ID is 199852, and the screen output logging file should be named `wannier90-t02.199852`. Your job ID will be different. Typically you should use `squeue --me` command to see the status of queuing/running jobs, but since this job finishes quickly, you need to use `sacct`:

```
sacct -j 199852
```

More useful job monitoring commands here: https://hprc.tamu.edu/kb/User-Guides/Launch/Batch/#job-monitoring-and-control-commands

HIGH PERFORMANCE RESEARCH COMPUTING
TEXAS A&M UNIVERSITY

https://hprc.tamu.edu

HPRC Helpdesk:

help@hprc.tamu.edu
Phone: 979-845-0219

Take our short course survey!

https://u.tamu.edu/hprc_shortcourse_survey

HPRC Survey

https://u.tamu.edu/hprc_shortcourse_survey

Help us help you. Please include details in your request for support, such as, **Cluster** (ACES, FASTER, Grace, Launch), NetID (UserID), Job information (**JobID**(s), Location of your jobfile, input/output files, Application, Module(s) loaded, Error messages, etc), and Steps you have taken, so we can reproduce the problem.

Extra Slides

What about CMake?

First, find the compatible CMake module with foss/2024a toolchain:

```
ml avail CMake
```

```
CMake/3.29.3 (D)
```

Now load it:

```
ml CMake/3.29.3
```

Change directory back to the root of wannier90 repo:

```
cd $SCRATCH/wannier90
```

Create a new build directory for CMake with foss/2024a:

```
mkdir build_foss_2024a && cd build_foss_2024a
```

CMake Structure and Syntax

- CMakeLists.txt: build options that can be configured. Most notable among the configurable options is CMAKE_BUILD_TYPE. This should be set to “Release” to ensure the compiled executables are optimized.
- CMakePresets.json: configuration presets, each preset contains a set of build options

For wannier90, CMAKE_BUILD_TYPE is Release by default, and we will use mpi preset:

```
cmake .. --preset=mpi
```

```
Could NOT find Python (missing: Python_EXECUTABLE Interpreter)
(Required is at least version "3.8")
```

What happened?

CMake and Python Saga

It turns out, for some reason, wannier90's test module requires Python 3.8 or later. The default Python version of all our supercomputers (RHEL 8 OS) is 3.6.5. To fix this, we need to find a newer Python module compatible with foss/2024a toolchain:

```
ml avail Python
```

```
Python/3.12.3
```

Load Python/3.12.3:

```
ml Python/3.12.3
```

CMake: Configure and Build

Then run the cmake command again:

```
cmake .. --preset=mpi
```

```
-- Generating done (0.3s)
-- Build files have been written to:
/scratch/user/u.th284212/wannier90/cmake-build-mpi
```

Change directory to `cmake-build-mpi`:

```
cd $SCRATCH/wannier90/cmake-build-mpi
```

Then build it with the regular "make" command:

```
make
```

Once done, you should see 4 executable files as the result of the build: `wannier90.x`, `w90chk2chk.x`, `w90spn2spn.x`, and `postw90.x`

Exercise Time!

- Test-run the newly CMake-built wannier90.x executable
  `$SCRATCH/wannier90/cmake-build-mpi/wannier90.x`
  in
  `$SCRATCH/wannier90/tutorials/tutorial02`

- Modify Slurm job script
  `wannier90_tutorial02.slurm`
  To execute the newly CMake-built wannier90.x executable in tutorial02, then submit and monitor the job.

Quick Test-Run

Again, to test-run wannier90, change directory to tutorial02:

```
cd $SCRATCH/wannier90/tutorials/tutorial02
```

Clean up previously generated lead.chk and lead.wout results:

```
rm lead.chk lead.wout
```

Then type:

```
mpirun -np 2 $SCRATCH/wannier90/cmake-build-mpi/wannier90.x lead
```

This will run wannier90.x with 2 parallel threads, and should return within 2-3 seconds without any message. You should be able to find `lead.chk` and `lead.wout` in the same directory afterward.

Submitting a wannier90 Slurm Job

Modify the `wannier90_tutorial02.slurm` job script, change the main `mpirun` command line to:

```
mpirun $SCRATCH/wannier90/cmake-build-mpi/wannier90.x lead
```

Clean up previously generated lead.chk and lead.wout results:

```
rm lead.chk lead.wout
```

Then submit it:

```
sbatch wannier90_tutorial02.slurm
```

```
Submitted batch job 202189
```

And monitor the job (your job ID will be different):

```
sacct -j 202189
```

Setting Up Environments for:

- Running Python scripts/notebooks
  - Using ModuLair
  - Using Mamba/Conda
- Running R scripts
- Compiling software
  - Using GNU (FOSS) toolchain
  - Using Intel toolchain

Let’s make sure to start fresh: New tmux

Exit out of your current tmux session if you are inside one:

```
exit
```

If your terminal is terminated (no pun intended), open a new terminal and kill all existing tmux sessions:

```
tmux kill-server
```

Then start a new tmux session:

```
tmux
```

Wannier90 with Intel Toolchain

For Intel toolchain, we will use the template `make.inc.ifx`.
First, enter wannier90 directory:

```
cd $SCRATCH/wannier90
```

Then, copy the template:

```
cp ./config/make.inc.ifx ./make.inc
```

A Quick Glance of Intel make.inc

These lines tell the names of Intel fortran compiler commands:

```
F90 = ifx
MPIF90 = mpiifx
```

These lines tell the compiler options, or “flags”, for compilation and linking, respectively:

```
FCOPTS=-fPIC -check all -warn all -g -diag-disable 8889,10182,10440
FCOPTS=-O3 -fPIC -warn all -diag-disable 8889
LDOPTS=$(FCOPTS)
```

Finally, this line specifies the Intel Math Kernel Libraries (MKL) that wannier90 should link to. This is the equivalents of ScaLAPACK and FlexiBLAS in FOSS toolchain:

```
LIBS = -L$(MKLROOT) -lmkl_core -lmkl_intel_lp64 -lmkl_sequential
```

Loading an Intel Toolchain

To list available intel toolchains, type:

```
Toolchains intel
```

```
intel/2024a = GCCcore/13.3.0 & impi/2021.13.0 | Python/3.12.3
intel/2023b = GCCcore/13.2.0 & impi/2021.10.0 | Python/3.11.5
intel/2023a = GCCcore/12.3.0 & impi/2021.9.0  | Python/3.11.3
```

In this short course, we will be loading intel/2024a:

```
ml intel/2024a
```

Listing Components of an Intel Toolchain

```
ml
```

```
Currently Loaded Modules:
  1) GCCcore/13.3.0                5) numactl/2.0.18    9) imkl-FFTW/2024.2.0
  2) zlib/1.3.1                    6) UCX/1.16.0       10) intel/2024a
  3) binutils/2.42                 7) impi/2021.13.0
  4) intel-compilers/2024.2.0      8) imkl/2024.2.0
```

Clean and Build It!

Make sure to clean the build first:

```
make veryclean
```

And then build the program:

```
make all
```

It should take about 1 minute to clean and just a bit more than 1 minute to compile all wannier90 components. Once done, type:

```
ls
```

And you should see 4 executable files as the result of the build: `wannier90.x`, `w90chk2chk.x`, `w90spn2spn.x`, and `postw90.x`

Exercise Time!

- Test-run the newly built wannier90.x executable with intel toolchain:
  `$SCRATCH/wannier90/wannier90.x`
  in
  `$SCRATCH/wannier90/tutorials/tutorial02`

- Modify Slurm job script
  `wannier90_tutorial02.slurm`
  To load **intel/2024a** toolchain instead of **foss/2024a**, and execute the newly built wannier90.x executable in tutorial02, then submit and monitor the job.

Quick Test-Run

Again, to test-run wannier90, change directory to tutorial02:

```
cd $SCRATCH/wannier90/tutorials/tutorial02
```

Clean up previously generated lead.chk and lead.wout results:

```
rm lead.chk lead.wout
```

Then type:

```
mpirun -np 2 $SCRATCH/wannier90/wannier90.x lead
```

This will run wannier90.x with 2 parallel threads, and should return within 2-3 seconds without any message. You should be able to find `lead.chk` and `lead.wout` in the same directory afterward.

Submitting a wannier90 Slurm Job

Modify the `wannier90_tutorial02.slurm` job script, change “foss” to “intel” in the module loading command:

```
ml intel/2024a
```

Make sure the main mpirun command point to the correct path:

```
mpirun $SCRATCH/wannier90/wannier90.x lead
```

Clean up previously generated lead.chk and lead.wout results:

```
rm lead.chk lead.wout
```

Then submit it:

```
sbatch wannier90_tutorial02.slurm
```

```
Submitted batch job 202189
```

Using CMake with Intel Toolchain

Intel/2024a is compatible with the same CMake/3.29.3 and Python/3.12.3 under foss/2024a. To load them, type:

```
ml CMake/3.29.3 Python/3.12.3
```

Change directory back to the root of wannier90 repo:

```
cd $SCRATCH/wannier90
```

Clear the foss/2024a cmake build directory:

```
rm -rf cmake-build-mpi
```

Create a new build directory for CMake with intel/2024a:

```
mkdir build_intel_2024a && cd build_intel_2024a
```

Intel Compiler CMake Flags

Remember that intel/2024a depends on GCCcore/13.3.0? This causes CMake to believe the default C, Fortran, and C++ compiler name for Intel toolchain is the GNU variants: gcc, gfortran, and g++, respectively. We need to ensure CMake is using the right compiler names (mpiicx, mpiifx, and mpiicpx) by setting these variables:

`CC=mpiicx FC=mpiifx CXX=mpiicpx`

For wannier90, we only need the fortran compiler:

```
FC=mpiifx cmake .. --preset=mpi
```

```
-- Generating done (0.3s)
-- Build files have been written to:
/scratch/user/u.th284212/wannier90/cmake-build-mpi
```

Build It!

Change directory to `cmake-build-mpi`:

```
cd $SCRATCH/wannier90/cmake-build-mpi
```

Then build it with the regular “make” command:

```
make
```

Once done, you should see 4 executable files as the result of the build: `wannier90.x`, `w90chk2chk.x`, `w90spn2spn.x`, and `postw90.x`

Exercise Time!

- Test-run the newly CMake-built wannier90.x executable
  `$SCRATCH/wannier90/cmake-build-mpi/wannier90.x`
  in
  `$SCRATCH/wannier90/tutorials/tutorial02`

- Modify Slurm job script
  `wannier90_tutorial02.slurm`
  To execute the newly CMake-built wannier90.x executable in tutorial02, then submit and monitor the job.

Quick Test-Run

Again, to test-run wannier90, change directory to tutorial02:

```
cd $SCRATCH/wannier90/tutorials/tutorial02
```

Then type:

```
mpirun -np 2 $SCRATCH/wannier90/cmake-build-mpi/wannier90.x lead
```

This will run wannier90.x with 2 parallel threads, and should return within 2-3 seconds without any message. You should be able to find `lead.chk` and `lead.wout` in the same directory afterward.

Submitting a Wannier90 Slurm Job

Modify the `wannier90_tutorial02.slurm` job script, change the main `mpirun` command line to:

```
mpirun $SCRATCH/wannier90/cmake-build-mpi/wannier90.x lead
```

Then submit it:

```
sbatch wannier90_tutorial02.slurm
```

```
Submitted batch job 202189
```

And monitor the job (your job ID will be different):

```
sacct -j 202189
```