Advanced HPC Usage

Interactive Jobs from the Command Line

Running Array Jobs with Slurm

Advanced Slurm Job Debugging

Advanced kstat Usage

Advanced File Access

File Sharing using ACLs

HPC Python Virtual Environments

Migrating Jupyter Notebooks to Slurm Python Scripts

Interactive Jobs using Slurm

Interactive jobs from the command line

Users often run interactive jobs from the OnDemand interface.
You can also request an interactive session from the command line.

ssh into the head node and issue an ***srun*** command as below:

srun **-J** myjobname **-N** 1 **-n** 16 **-t** 24:00:00 **--mem**=30G **-C** moles **-p** ksu-cis-hpc.q **--pty** bash

The srun command submits the interactive request to Slurm.
This request sets the job name to *myjobname*, requests *1* node with *16* cores and *30 GB* for *24 hours* with the constraint of the *moles* only and the priority of the *ksu-cis-hpc* group.
The last part chooses the *bash* shell.

Once you submit this request, you may need to wait a while until it is allocated.
Then you will drop into the node requested and be in an interactive shell.

This can be very useful for debugging codes that need more resources than is available on the head nodes.

Array Jobs in Slurm

Running multiple similar jobs from one job script

Some users submit hundreds or thousands of similar jobs using scripts.
This clogs the Slurm queue making it difficult to see what jobs other users have queued.
Slurm can only look a certain depth into the queue, so higher priority jobs or jobs needing less resources that could run immediately will not because they are buried too deep.

Array jobs use a single job script to submit many similar jobs.
This is easier for the user to control as scancel would just require a single job ID.
Having a single job script means no clogging the Slurm queue.
Best yet, it does not change the scheduling priority of your jobs compared to individual scripts.

Constructing an array job script

```
#SBATCH --array=1-5:2
$SLURM_ARRAY_TASK_ID
```

While you submit a single job script, Slurm will treat it like multiple job submissions with the $SLURM_ARRAY_TASK_ID set differently in each run.
The example above is for a range from 1 to 5 with an optional step 2, so there would be 3 jobs with $SLURM_ARRAY_TASK_ID set to 1, 3, and 5.

It is up to the user to determine how to use the $SLURM_ARRAY_TASK_ID variable.
It can be an input to the application or used to choose a different input file for example.

If you need help writing an array job script, please contact the administrators.

Advanced Slurm Job Debugging

kstat -l provides advanced hardware and performance information

/tmp memory size, communication network rate and type, and GPU type and memory
    the GPU type is the string to use when requesting a specific GPU like —gres=gpu:rtx_a4000:1
The current and maximum memory used for each job.
The CPU utilizations for each job include user, system, idle, and IO wait percentages
    these are summed across cores and updated each minute, but otherwise what htop would show on the node
The GPU utilization and memory for each job.

You can also *ssh* into any node you have a job running on

htop then ***choose your user name*** to isolate your job's processes
    This will give you thread usage and memory info updated every second.
nvidia-smi
    This will provide a snapshot of GPU utilization and GPU memory usage

Google any error messages

If you get an error message, Google the app name and the error message
    this is usually what we do first when you contact us
If you still need help provide us lots of information
    The **path to your job**, the **job ID**, the **job script name**, the exact **error message**, your **KSU eID** or **WSU_ID**

Advanced *kstat* Usage

Use kstat to print a table of CPU or GPU utilization and memory

kstat --table-cpu-60min 20826470

CPU and Memory Usage for job **20826470** every 60 minutes

Minutes	User	System	Idle	IO Wait	Memory	Disk Swap
0	97.7%	0.1%	2.2%	0.0%	1.498 gb	0.000 gb
60	99.3%	0.1%	0.6%	0.0%	1.547 gb	0.000 gb
120	99.2%	0.1%	0.8%	0.0%	1.549 gb	0.000 gb
180	98.5%	0.1%	1.5%	0.0%	1.550 gb	0.000 gb
240	97.4%	0.1%	2.5%	0.0%	1.477 gb	0.000 gb
300	99.0%	0.1%	0.9%	0.0%	1.533 gb	0.000 gb

kstat --table-gpu-60min 20826470

GPU usage and memory for job **20826470** every 60 minutes

Minutes	Usage	Memory
0	2.0%	192.867 mb
60	2.0%	193.000 mb
120	2.0%	193.000 mb
180	2.0%	193.000 mb
240	2.0%	193.000 mb
300	2.0%	193.000 mb

kstat graphs

Use kstat to graph CPU or GPU usage or memory

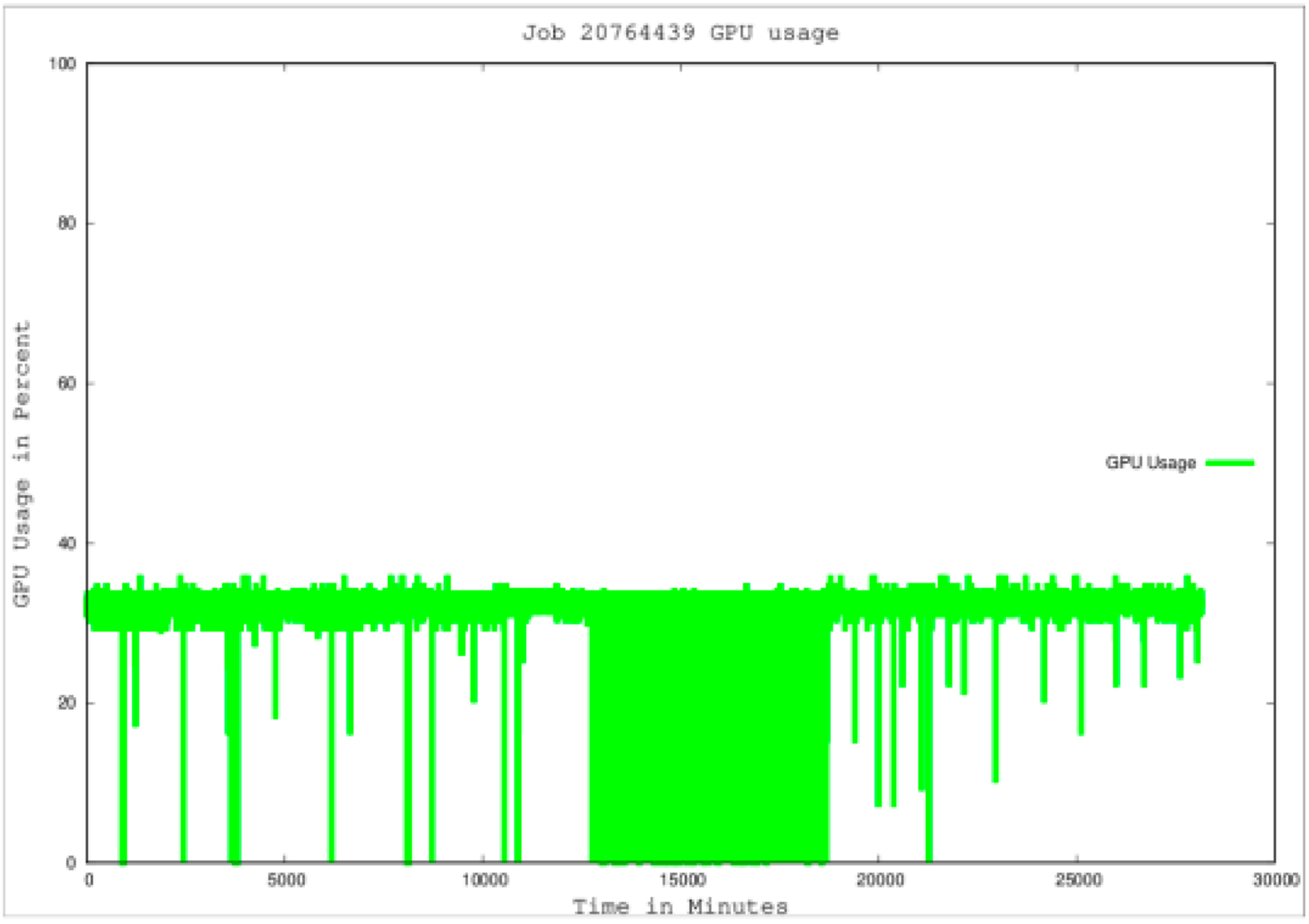
For graphs you must enable X11 forwarding and have some software that can graph on your local system. My Apple laptop uses XQuartz to graph.

ssh -X your_wsu_id@hpc-login.wichita.edu
or
have **ForwardX11 yes** in your **.ssh/config** file for your BeoShock connection

You can just NOT enable X11 forwarding and transfer the resulting graph which gets stored as **.kstat.gnuplot.png** on your **home directory** to your local system and view it there.

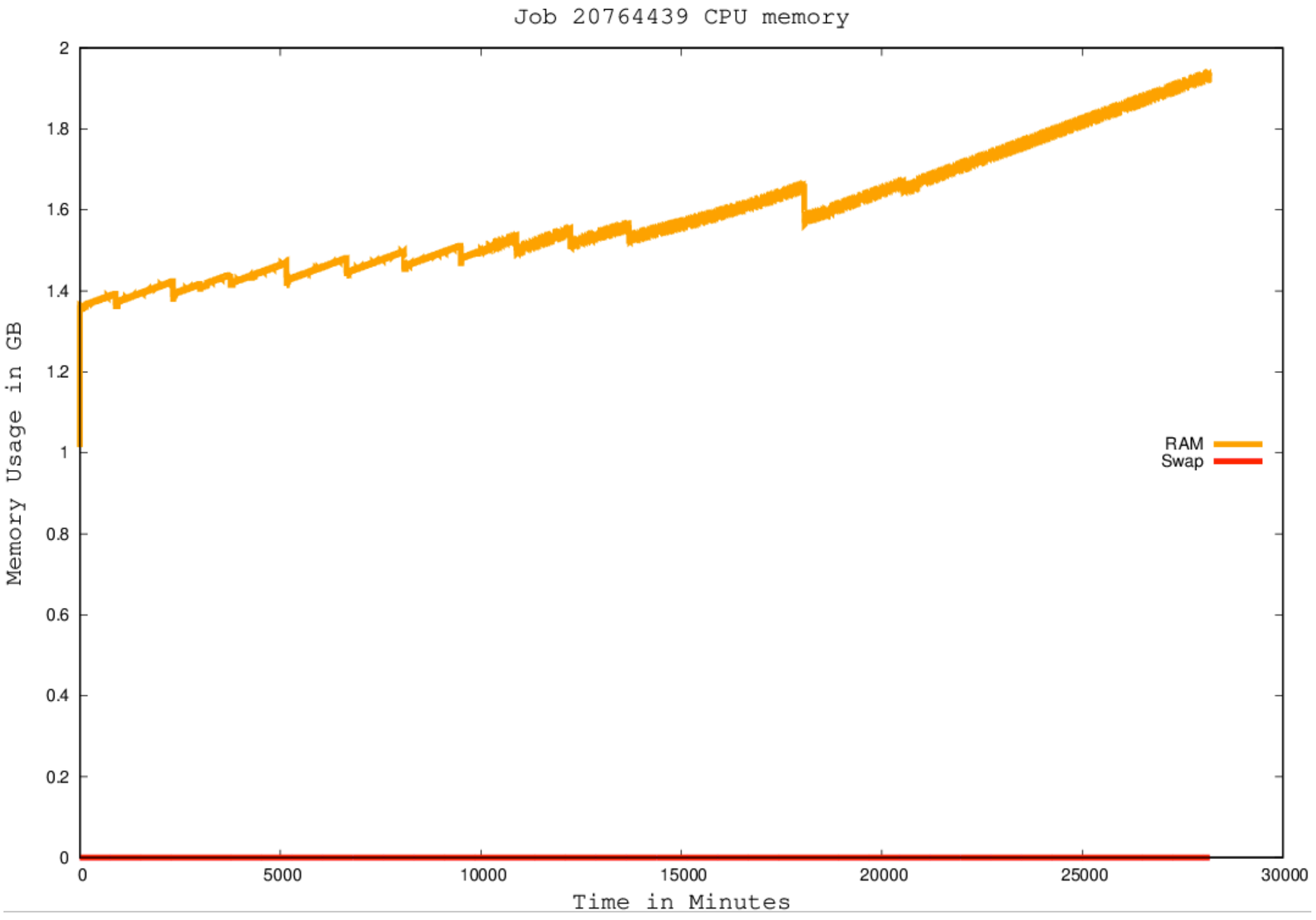
kstat GPU usage graph

kstat —graph-gpu-usage 20764439



kstat GPU usage graph

kstat —graph-cpu-memory 20764439



kstat totals for the entire cluster

Use kstat to print a table of CPU usage and memory for the entire cluster

kstat --table-node all

This cluster has 21 nodes 768 cores 8 GPUs and 7.064 TB memory

Usage and Memory for all nodes for 365 days

Day	Cores	Load	Used	Alloc	Total Memory
0	186	52.8	694	2363	7233 GB
1	314	136.8	799	2711	7233 GB
2	308	113.1	815	2873	7233 GB
3	369	232.2	1026	2826	7233 GB
4	380	237.8	853	2400	7233 GB
5	365	250.3	438	1912	7233 GB
6	474	381.2	538	1973	7233 GB
7	519	386.7	755	2192	7233 GB
8	571	444.5	914	2985	7233 GB
9	517	419.6	820	1869	7233 GB
10	406	329.7	743	1702	7233 GB

kstat --table-node gpu202401 look at a single node

Advanced File Access

Home directories

On **Beocat** each user has 1 TeraByte (1000 GigaBytes) of space on their **/homes** directory.
If you exceed that your **/homes** directory will not be backed up.
If you exceed 2 TeraBytes, you will not be able to run on Beocat until you delete files.

On BeoShock, if you are going to use more than a few TeraBytes please contact the HPC Directory Terrance Figy for approval as resources are limited.

Bulk directories

If you need more storage space on Beocat please contact the administrators.
You can rent **/bulk** storage space at **$45/TB/year** billed monthly.

Scratch space

Beocat has 270 TeraBytes of short term storage space that is purged monthly.
Create your own directory on **/fastscratch**
Keep in mind that this is a shared resource and may fill up.

This is a ZFS file system which is different from the CEPH parallel file servers that supports the **/homes** and **/bulk** directories. It does not suffer performance issues from opening multiple file for writing on the same directory that CEPH can.

File Sharing using ACLs

Sharing files using the *setacls* script

All home directories must be read and write locked so other users cannot access them.
You may share subdirectories with groups or users but need to use Access Control Lists (ACLs).

setacls -h this will provide usage information for the command

setacls [-r] [-w] [-g group] [-u user] -d /full/path/to/directory
Execute pemission will always be applied, you may also choose r or w
Must specify at least one group or user
Must specify at least one directory or file, and it must be the full path

setacls -r -g ksu-cis-hpc -u mozes -d /homes/daveturner/shared_dir

The above example would provide **read access** to the priority group ***ksu-cis-hpc*** plus the user ***mozes*** to the ***shared_dir*** subdirectory on ***/homes/daveturner***.

Priority groups are the same as used to gain access to compute nodes owned by a group.
If you are sharing with only a few individuals, specify each user instead.

As the ***setacls*** script runs, it will print the full set of ***setfacl*** commands that you would otherwise have needed to use instead.

getfacl /homes/daveturner/shared_dir

Once you have shared a subdirectory, you can check the sharing with the ***getfacl*** command.

HPC Python Virtual Environments

It is very useful to have different Python versions and libraries installed for each project you work on to ensure that all software matches the needs of each scientific application. Virtual environments are the way to accomplish this.

Beocat https://support.beocat.ksu.edu/Docs/Installed_software#Python
BeoShock https://docs.hpc.wichita.edu/index.php?title=Installed_software#Python

Load the version of Python you want

```
module avail I& grep -i python
module load Python/3.11.5-GCCcore-13.2.0
```

Create the virtualenv directory and cd to it

```
mkdir -p ~/virtualenv
cd ~/virtualenv
```

Create the virtual environment for this project

```
python -m -venv —system-site-packages py_env_3.11.5
```

Activate the virtual environment

```
source ~/virtualenv/py_env_3.11.5/bin/activate
```

Install any packages you need into that specific environment

```
pip install numpy scipy
```

You can now use this environment and installed packages anytime by activating it.
!Precede everything with an exclamation point if doing this in a Jupyter notebook.

Migrating Jupyter Notebooks to Slurm

Beocat https://support.beocat.ksu.edu/Docs/Installed_software#Jupyter
BeoShock

Jupyter Notebooks in OnDemand are great for developing Python code.
You can interactively add code and immediately test with just a mouse click.

They are not great for doing production runs once the code is completed.
In OnDemand you request a certain runtime, and the Jupyter Notebook
locks the resources up for this entire time even if the code has finished.
This wastes resources for the cluster, while a Slurm job completes and
releases the resources when the code completes.

Convert Jupyter Notebook code.ipynb to Python code.py

From the OnDemand Jupyter **File** menu choose **Download as** and Python (.py)

If you already have a file like code.ipynb on Beocat or BeoShock use:

```
module load JupyterLab
jupyter nbconvert code.ipynb --to script
```