To login to hpc from powershell:
ssh lewis.hotchkiss@login-bluec-01

There are different ways of copying files to the cluster. One way is by using the scp command, the other is to set up WinSCP:

To copy files from local to hpc:
scp P:/lewishotchkiss/Documents/filename.py lewis.hotchkiss@login-bluec-01:~/

To copy files from hpc to local:
scp lewis.hotchkiss@login-bluec-01:~/folder/filename.py P:/lewishotchkiss/Documents

Connect folders to hpc via WinSCP:
Use WinSCP program
Host: login-bluec-01
Username: lewis.hotchkiss
Password: **********

You will need a singularity container to be able to run your code, this comes in the form of a SIF file. One of the tools available to automatically create singularity containers for you is neurodocker. Neurodocker allows you to create singularities by choosing your own base operating system, neuroimaging software you might want to use, and programming languages such as Python. Alternatively, you can create your own singularity from scratch.

Neurodocker (Generate singularity recipe file on local):

```
Neurodocker generate singularity \
        --base-image debian:stretch \
        --pkg-manager apt \
        --fsl version=5.0.10 \
        --miniconda version=4.6.14 \
                env_name=neuro \
                pip_install “xnat minio” > SingularityRecipe
```

Build Singularity (on local):
sudo singularity build python-fsl.sif SingularityRecipe

There are two ways of running scripts with singularity:

To run container:
singularity exec ./python-fsl.sif python3 myscript.py

To run container as shell:
singularity shell ./python-fsl.sif

In the first one, you specify the script and what package you want to use to run it. In the second one, you just run the container as shell which then allows an interactive session. An example is shown below:

```
lewis.hotchkiss@login-bluec-01:~$ lewis.hotchkiss@login-bluec-01:~$ singularity exec ./pyfsl.sif python3 test.py
HEllO
lewis.hotchkiss@login-bluec-01:~$ singularity shell ./pyfsl.sif
Singularity> python3
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>> quit()
Singularity>
```

This allows you to quickly test if different parts of your scripts are working as they should.

However, you want to be submitting these scripts to the HPC via SLURM.

Submitting jobs via SLURM

To submit jobs to the cluster via SLURM, you need to create a shell script. You can do this by typing nano into the terminal which will then open up a text editor in the terminal.

```
  GNU nano 6.2
#!/bin/bash

# Name your job
#SBATCH -J job_name

# Define how many nodes you need
#SBATCH --nodes=1
# You can alos ask for number of tasks with --ntasks-per-*
# (If --ntasks=4 then 4 cpus will be used

# Specify how long your job will last for
#SBATCH --time=0-00:05:00
#              d-hh:mm:ss

# Specify how much memory you will need
# Either by specifying --mem= or by --mem-per-cpu= which will assign memory per cpu
#SBATCH --mem=5GB
#SBATCH --mem-per-cpu=1500MB

# Then you can run all of your processes below:

singularity exec ./pyfsl.sif python3 my_script.py
```

As you can see, I used the same command as above ‘singularity exec’, to run the script.

Save this file and exit back to terminal.

You can then run your script by typing sbatch script_name.sh

Below shows a list of commands that you can use with slurm, including job submissions and job management.

slurm®
workload manager

Job Submission

salloc - Obtain a job allocation.
sbatch - Submit a batch script for later execution.
srun - Obtain a job allocation (as needed) and execute an application.

--array=<indexes> (e.g. "--array=1-10")	Job array specification. (sbatch command only)
--account=<name>	Account to be charged for resources used.
--begin=<time> (e.g. "--begin=18:00:00")	Initiate job after specified time.
--clusters=<name>	Cluster(s) to run the job. (sbatch command only)
--constraint=<features>	Required node features.
--cpu-per-task=<count>	Number of CPUs required per task.
--dependency=<state:jobid>	Defer job until specified jobs reach specified state.
--error=<filename>	File in which to store job error messages.
--exclude=<names>	Specific host names to exclude from job allocation.
--exclusive[=user]	Allocated nodes can not be shared with other jobs/users.
--export=<name[=value]>	Export identified environment variables.
--gres=<name[:count]>	Generic resources required per node.
--input=<name>	File from which to read job input data.
--job-name=<name>	Job name.
--label	Prepend task ID to output. (srun command only)
--licenses=<name[:count]>	License resources required for entire job.
--mem=<MB>	Memory required per node.
--mem-per-cpu=<MB>	Memory required per allocated CPU.
-N<minnodes[-maxnodes]>	Node count required for the job.
-n<count>	Number of tasks to be launched.
--nodelist=<names>	Specific host names to include in job allocation.
--output=<name>	File in which to store job output.
--partition=<names>	Partition/queue in which to run the job.
--qos=<name>	Quality Of Service.
--signal=[B:]<num>[@time]	Signal job when approaching time limit.
--time=<time>	Wall clock time limit.
--wrap=<command_string>	Wrap specified command in a simple "sh" shell. (sbatch command only)

Accounting

sacct - Display accounting data.

--allusers	Displays all users jobs.
--accounts=<name>	Displays jobs with specified accounts.
--endtime=<time>	End of reporting period.
--format=<spec>	Format output.
--name=<jobname>	Display jobs that have any of these name(s).
--partition=<names>	Comma separated list of partitions to select jobs and job steps from.
--state=<state_list>	Display jobs with specified states.
--starttime=<time>	Start of reporting period.

SchedMD
Slurm Support and Development

sacctmgr - View and modify account information.
Options:

--immediate	Commit changes immediately.
--parseable	Output delimited by '|'

Commands:

add <ENTITY> <SPECS> create <ENTITY> <SPECS>	Add an entity. Identical to the **create** command.
delete *<ENTITY>* where *<SPECS>*	Delete the specified entities.
list <ENTITY> [<SPECS>]	Display information about the specific entity.
modify *<ENTITY>* where *<SPECS>* set *<SPECS>*	Modify an entity.

Entities:

account	Account associated with job.
cluster	*ClusterName* parameter in the *slurm.conf*.
qos	Quality of Service.
user	User name in system.

Job Management

sbcast - Transfer file to a job's compute nodes.

sbcast [options] SOURCE DESTINATION

--force	Replace previously existing file.
--preserve	Preserve modification times, access times, and access permissions.

scancel - Signal jobs, job arrays, and/or job steps.

--account=<name>	Operate only on jobs charging the specified account.
--name=<name>	Operate only on jobs with specified name.
--partition=<names>	Operate only on jobs in the specified partition/queue.
--qos=<name>	Operate only on jobs using the specified quality of service.
--reservation=<name>	Operate only on jobs using the specified reservation.
--state=<names>	Operate only on jobs in the specified state.
--user=<name>	Operate only on jobs from the specified user.
--nodelist=<names>	Operate only on jobs using the specified compute nodes.

squeue - View information about jobs.

--account=<name>	View only jobs with specified accounts.
--clusters=<name>	View jobs on specified clusters.
--format=<spec> (e.g. "--format=%i %j")	Output format to display. Specify fields, size, order, etc.
--jobs<job_id_list>	Comma separated list of job IDs to display.
--name=<name>	View only jobs with specified names.
--partition=<names>	View only jobs in specified partitions.
--priority	Sort jobs by priority.
--qos=<name>	View only jobs with specified Qualities Of Service.
--start	Report the expected start time and resources to be allocated for pending jobs in order of increasing start time.
--state=<names>	View only jobs with specified states.
--users=<names>	View only jobs for specified users.

sinfo - View information about nodes and partitions.

--all	Display information about all partitions.
--dead	If set, only report state information for non-responding (dead) nodes.
--format=<spec>	Output format to display.
--iterate=<seconds>	Print the state at specified interval.
--long	Print more detailed information.
--Node	Print information in a node-oriented format.
--partition=<names>	View only specified partitions.
--reservation	Display information about advanced reservations.
-R	Display reasons nodes are in the down, drained, fail or failing state.
--state=<names>	View only nodes specified states.

scontrol - Used view and modify configuration and state.
Also see the **sview** graphical user interface version.

--details	Make show command print more details.
--oneliner	Print information on one line.

Commands:

create *SPECIFICATION*	Create a new partition or .
delete *SPECIFICATION*	Delete the entry with the specified SPECIFICATION
reconfigure	All Slurm daemons will re-read the configuration file.
requeue JOB_LIST	Requeue a running, suspended or completed batch job.
show ENTITY ID	Display the state of the specified entity with the specified identification
update *SPECIFICATION*	Update job, step, node, partition, or reservation configuration per the supplied specification.

Environment Variables

SLURM_ARRAY_JOB_ID	Set to the job ID if part of a job array.
SLURM_ARRAY_TASK_ID	Set to the task ID if part of a job array.
SLURM_CLUSTER_NAME	Name of the cluster executing the job.
SLURM_CPUS_PER_TASK	Number of CPUs requested per task.
SLURM_JOB_ACCOUNT	Account name.
SLURM_JOB_ID	Job ID.
SLURM_JOB_NAME	Job Name.
SLURM_JOB_NODELIST	Names of nodes allocated to job.
SLURM_JOB_NUM_NODES	Number of nodes allocated to job.
SLURM_JOB_PARTITION	Partition/queue running the job.
SLURM_JOB_UID	User ID of the job's owner.
SLURM_JOB_USER	User name of the job's owner.
SLURM_RESTART_COUNT	Number of times job has restarted.
SLURM_PROCID	Task ID (MPI rank).
SLURM_STEP_ID	Job step ID.
SLURM_STEP_NUM_TASKS	Task count (number of MPI ranks).

Daemons

slurmctld	Executes on cluster's "head" node to manage workload.
slurmd	Executes on each compute node to locally manage resources.
slurmdbd	Manages database of resources limits, licenses, and archives accounting records.

SchedMD
Slurm Support and Development
slurm® workload manager

http://www.schedmd.com

Submitting jobs in parallel

The command srun is the simplest way of running jobs in parallel. Below shows a simple example of running three python scripts in parallel. This will then produce a file named jobid.out, where jobid is the number of the job submitted. You can open this using nano jobid.out to view the output.

```
  GNU nano 6.2                                                    parallel_test.sh
#!/bin/bash

#SBATCH --job-name parallel
#SBATCH --output slurm-%j.out
# (-%j is the job id)
#SBATCH --ntasks=3
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1G
##SBATCH --partition=normal
#SBATCH --time=0-00:10:00


# Execute job steps
srun --ntasks=1 --nodes=1 --cpus-per-task=1 bash -c "singularity exec ./pyfsl.sif python3 script1.py" &
srun --ntasks=1 --nodes=1 --cpus-per-task=1 bash -c "singularity exec ./pyfsl.sif python3 script2.py" &
srun --ntasks=1 --nodes=1 --cpus-per-task=1 bash -c "singularity exec ./pyfsl.sif python3 script3.py"
wait
```