## Getting started on Prince

<table>
<thead>
<tr>
<th>Quick Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC at NYU</td>
</tr>
<tr>
<td>HPC Newsletter</td>
</tr>
<tr>
<td>Getting started on Mercer</td>
</tr>
<tr>
<td>Getting an account</td>
</tr>
<tr>
<td>Logging in</td>
</tr>
<tr>
<td>Windows</td>
</tr>
<tr>
<td>Mac / Linux</td>
</tr>
<tr>
<td>Clusters and Storage</td>
</tr>
<tr>
<td>Mercer (HPC)</td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
</tr>
<tr>
<td>Dalmata (NYU Abu Dhabi)</td>
</tr>
<tr>
<td>Transferring data to/from the clusters</td>
</tr>
</tbody>
</table>
Transfering data to/from Mercer cluster using Globus
Writing and submiting jobs with qsub
Available software
Building Software
Tutorials
Research Gallery
FAQs
NYU HPC Cheat Sheet
Wiki Map
(Tip: click "<<" at bottom left to close Confluence sidebar)

Under Construction
This page is under construction

- Logging in to Prince
- In a nutshell
- The full story
  - SSH tunneling for easier login and data transfer
  - What can I do on the login node?
- Adding Mercer to your SSH Tunnel
- Filesystems on Mercer
- Transferring files to Mercer
- How to migrate from another cluster to Mercer
- Compiling Code on Mercer
- Running batch jobs on Mercer
  - Submitting batch jobs on Mercer
    - Requesting CPUs
    - Requesting dedicated nodes
    - Interactive sessions
    - MPI jobs
Logging in to Prince

Logging in to Prince is the same 2-stage process as logging into any other NYU HPC cluster. The host name is 'prince.es.its.nyu.edu'.

The HPC clusters (Mercer and Dumbo) are not directly visible to the internet (outside the NYU Network). If you are outside NYU's Network (off-campus) you must first login to a bastion host named hpc.nyu.edu.

The diagram below illustrates the login path.

![Diagram illustrating the login path from workstation to HPC clusters](image)

**NOTE:** The clusters can still access the internet directly. This may be useful when copying data from servers outside the NYU Network - see: How to copy files to and from the HPC clusters.

**NOTE:** Alternatively, instead of login to the bastion host, you can use VPN to get inside NYU’s network and access the HPC clusters directly. Instructions on how to install and use the VPN client are available here.

**NOTE:** You can't do anything on the bastion host, except ssh to the HPC clusters.

In a nutshell

- From an off-campus location (outside NYU-NET), logging in to the HPC clusters is a two-step process:
  a. First login to the bastion host, hpc.nyu.edu. From a Mac or Linux workstation, this is a simple terminal command (replace my_netid with your NetId). Your password is the same password you use for NYU Home:

```
ssh -X NYUNetID@hpc.nyu.edu
```

  *Windows users* will need to use Putty, see here for instructions.

  You can't do anything on the bastion host, except ssh to the cluster

  b. Next login to the cluster. For Mercer, this is done with:

```
ssh -X mercer
```

  For Dumbo, this is done with:
First you need to ensure your workstation has the necessary software and settings to connect to the clusters, and to use graphical interfaces. We have instructions for Windows and Mac users.

Next, here's how to log in from a Windows / Linux / Mac workstation.

**SSH tunneling for easier login and data transfer**

The two-stage access can be inconvenient, especially when transferring files to and from the clusters. Secure direct access and file transfer is possible by setting up SSH tunneling from your workstation to the HPC clusters. We have instructions on setting this up for Windows / Linux / Mac workstations.

**What can I do on the login node?**

The login nodes (mercer and dumbo) are for preparing, submitting and monitoring scripts, analyzing results, moving data around and code development and simple compilation. **Login nodes are Not suitable for running computational workloads!** - for this use the batch system.

Compiling a large source codebase, especially with heavy use of optimization or -ipo (interprocedural optimization), can use much memory and CPU time. In such circumstances it is best to use the batch system for compilation too, perhaps via an interactive batch job. Click here for more info about interactive batch jobs.

(more...)

**Adding Mercer to your SSH Tunnel**

On a Mac:
Filesystems on Mercer

The filesystems on Mercer are the same as on Bowery, with one minor exception: /home is larger, and served from the same storage server as /archive and /work (on Bowery, /home is local to the cluster and served from Bowery’s master administration node).

Important
Your /home space on Mercer will eventually become your one true home directory, so we recommend transferring important files, such as source code and run configuration files, from other clusters. The /home space on the other clusters will remain available for some time though.

The NYU HPC clusters have five filesystems for users’ files. Each filesystem is configured differently to serve a different purpose:

<table>
<thead>
<tr>
<th>Space</th>
<th>Environment Variable</th>
<th>Space Purpose</th>
<th>Visibility</th>
<th>Backed up?</th>
<th>Flushed?</th>
<th>Allocation</th>
<th>Cost for Additional Storage</th>
<th>Total Size</th>
<th>File System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home</td>
<td>$HOME</td>
<td>Program development space; storing small files you want to keep long term, e.g. source code, scripts.</td>
<td>login and compute nodes. Starting with the installation of Mercer we have a unified /home filesystem served from same 7420 storage system as /archive and /work</td>
<td>Yes</td>
<td>ASCII filenames only</td>
<td>20GB (unified /home, mounted on Mercer)</td>
<td>N/A</td>
<td>600TB (shared with /archive and /work)</td>
<td>NFS ZFS</td>
</tr>
<tr>
<td>/archive</td>
<td>$ARCHIVE</td>
<td>Long-term storage, mounted only on login nodes. Best for large files, please tar collections of small files when archiving.</td>
<td>login nodes only. Common to all clusters.</td>
<td>Yes</td>
<td>ASCII filenames only</td>
<td>2TB</td>
<td>$500/year for 1TB</td>
<td>600TB shared with /work and unified /home</td>
<td>ZFS</td>
</tr>
<tr>
<td>Directory</td>
<td>Environment Variable</td>
<td>Location</td>
<td>Purpose</td>
<td>Access Period</td>
<td>Quota</td>
<td>FileSystem Type</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------------------</td>
<td>----------------</td>
<td>--------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/scratch</td>
<td>$SCRATCH</td>
<td>login and compute nodes.</td>
<td>Computational work space. Best suited to large, infrequent reads and writes. Files are deleted after 60 days without use. Common to all clusters.</td>
<td>No</td>
<td>Files not accessed for 60 days</td>
<td>5TB/note quota: 1 million Policy</td>
<td>N/A</td>
<td>410TB Lustre</td>
<td></td>
</tr>
<tr>
<td>/work</td>
<td>$WORK</td>
<td>login and compute nodes.</td>
<td>Medium term, non-backed up storage mounted on login and compute nodes.</td>
<td>No</td>
<td>No</td>
<td>500GB</td>
<td>N/A</td>
<td>600TB ZFS shared with /arc hive and unified /home</td>
<td></td>
</tr>
<tr>
<td>/state/partition1</td>
<td>$PBS_JOBTMP</td>
<td>compute nodes only. Local to each compute node.</td>
<td>Small, node-local filesystem cleaned up at the end of each Torque job. For small, frequent reads and writes. Environment variable is defined in batch jobs (via qsub wrapper)</td>
<td>No</td>
<td>End of each job</td>
<td>Varies. Generally &gt;100GB</td>
<td>N/A</td>
<td>Varies ext3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$PBS_MEMDISK</td>
<td>compute nodes only. Local to each compute node.</td>
<td>Optional, node-local memory filesystem. Like $PBS_JOBTMP but smaller and faster. See here for usage.</td>
<td>No</td>
<td>End of each job</td>
<td>Default 8GB. Specific amount can be requested (but must fit within node memory)</td>
<td>N/A</td>
<td>Varies tmpfs or ramfs</td>
<td></td>
</tr>
</tbody>
</table>

Only files and directories with ASCII-only filenames are backed up. Our backup system does not handle unicode in file or directory names, such files and directories (including all files and directories under them) will be bypassed.

Important: Of all the space, only /scratch should be used for computational purposes. Please do not write to /home when running jobs as it can easily be filled up.

*Note: Capacity of the /home file system varies from cluster to cluster. Unlike /scratch and /archive, the /home file system is not mounted across clusters. Each cluster has its own /home, its own user base and /home allocation policy.

To purchase additional storage, send email to hpc@nyu.edu.

Transferring files to Mercer

Your files on /scratch and /work are visible on Mercer as well as Bowery, and will remain so.

Moving many small files takes significantly longer than moving one large file. Therefore, to transfer source code with many small files from /home on another cluster, we recommend tarring up the source directory first.

We recommend rebuilding all models on Mercer - this will ensure the best performance and also avoid any problems relating to mismatched OS and library versions between the clusters. Therefore, there is no need to transfer object files or executables.

How to migrate from another cluster to Mercer

If you have a workflow on, say, Bowery, and you wish to replicate it on Mercer:

1. Ensure the files you need are available on Mercer
   Files on $SCRATCH, $ARCHIVE and $WORK are already available on Mercer - there is nothing you need to do with these.
   Files on $HOME of Bowery (and Cardiac and Union Square) are local to each cluster, so you will need to copy them to Mercer. This is a good opportunity for a spring clean of junk files you no longer need!
   The following command, run on Bowery, will replicate your Bowery $HOME in a subdirectory called 'bowery-home' in your unified $HOME on Mercer:

   ```bash
rsumc -avz $HOME/ mercer.es.its.nyu.edu:bowery-home
   ```
2. Check your PBS (qsub) scripts for things which might need adjusting. Some things to look out for are:
   - What modules are used, and what versions of the modules are used?
     Only the latest version of most software has been installed on Mercer, and some modules that the older clusters have are not currently installed.
     Check that the modules you need are available ("module avail <modulename>") and check if the version on Mercer will work for you. If you need other modules, or assistance, contact us.
   - Check your #PBS directives. Especially, if you have "#PBS -q <some-queue>", you should remove that line. On Mercer, not all the queues are the same, and the system works out where to best put the job in any case.
   - Check your #PBS directives for -l nodes=1:ppn=12 and change the ppn to 10 or 20 (Mercer nodes have 20 cores each, so 12 is not a good number).
   - Many scripts have a line like:

     ```
     source /etc/profile.d/env-modules.sh
     ```

     This is no longer needed on Mercer (and will not work), so remove that line.

3. Check your .bashrc, .login and .cshrc files. If on the older cluster, you load certain modules in them, you may wish to add corresponding lines to those files on Mercer. You may also wish to replicate your shell aliases.

4. Try submitting your PBS script on Mercer. If it does not run, or the results are not what you expected, and it is not obvious why, contact us for help.

5. If you are running a model you compiled yourself, you will probably get improved performance by recompiling the model on Mercer. See below for more about compiling code.

### Compiling Code on Mercer

We recommend using the Intel compiler suite (module load intel), and, for MPI programs, OpenMPI (module load openmpi/intel).

The GNU compiler suite is also available (module load gcc) and over time other compiler versions and MPI implementations will be installed. There will be instances in which an alternative compiler or MPI library performs better or avoids a bug in the recommended compilers, but in the long run the functionality and performance of compilers and MPI libraries is very similar, so for easiest maintenance, only use the alternatives if you really need to.

**Mercer will eventually become a heterogeneous system** comprised of the Ivy Bridge nodes now installed, Westmere nodes integrated from Bowery, Sandy Bridge GPU nodes and future nodes based on some future series of CPUs. For the best performance, you should compile your code optimizing for Ivy Bridge, with AVX vector instructions for faster floating point computations. However, Westmere CPUs cannot run such code. **The Intel compiler can compile for two architectures in the same object files and executables, and we recommend that you use this feature.** To do so, use the following flags to icc, icpc and ifort:

```
-xSSE4.2 -axAVX
```

This will compile targeting Westmere (slightly older CPU cores, to be integrated into Mercer from Bowery) as a default, with an alternate path optimized for the newer Ivy Bridge cores of Mercer if they are available when the executable is run.

If you need GNU compilers, we recommend disabling AVX instructions to ensure your executable can run on any compute node. To do this add the flag:

```
-mno-avx
```
Running batch jobs on Mercer

On earlier NYU HPC clusters, the output of batch jobs appeared as it happened, in a file like "my_job.o12345" in the job submission directory. When jobs are submitted from /scratch, this can causes a heavy small-block-I/O load on the Lustre filesystem, which impacts performance.

On Mercer, job stdout and stderr are instead written to hidden files in your $HOME area, under $HOME/.pbs_spool. When the job completes, the my_job.o12345 and my_job.e12345 files are moved to their final location.

You can monitor your job's progress by looking for its output in this hidden directory

If your job writes too much to stdout or stderr, these temporary files might (temporarily) fill up your $HOME space allocation. If this is occurring, first check if you can reduce the amount of stdout and stderr (they can slow down your program, contact us if you would like assistance with this). Another option is to redirect stdout and stderr to a file in some other location:

- With bash:

```
my_program > my_output_file 2>&1
```

- With csh/tcsh

```
my_program >& my_output_file
```

Submitting batch jobs on Mercer

To submit a job on Mercer the minimum detail that you must specify is either the number of nodes and/or CPUs you need (if you need more than 1) or the amount of memory. It is also wise to specify the wallclock time, or your job will be killed after 1 hour.

There are fewer queues on Mercer than were on Bowery. It is almost always best to not specify a queue - the system will place your job in the most appropriate queue based on the resources (memory, walltime and nodes or procs) you request. The queues are described here.

Most nodes have either 24GB, 48GB, 64GB or 192GB of memory. However a small amount of memory is needed by the operating system, so nodes have about 23GB, 46GB, 62GB and 189GB available for jobs. If you request 64GB, you restrict the job to the subset of nodes having more than 64GB of memory, and it will probably take longer to schedule.

Requesting CPUs

On Bowery, nodes had either 8 or 12 CPUs per node. Mercer now contains all of the Bowery nodes plus 160 new nodes with 20 CPUs per node. Consequently, specifying:

```
#PBS -l nodes=1:ppn=12
```

Will get your job running sooner, perhaps on a new or perhaps on an old node, while

```
#PBS -l nodes=1:ppn=20
```

Will ensure you job gets a new node, but it might be queued for longer.
**Requesting dedicated nodes**

Some programs, such as Matlab, try to allocate far more than they actually used. Since the memory limit is enforced normally at allocation, the program might fail. In this case you can try two things:

1. Request more memory  
2. Request exclusive use of a node

When you have exclusive use of nodes, you need not request memory - you get the whole node, as per a parallel job

To request exclusive use of nodes, use:

```
#PBS -n
```

**Interactive sessions**

To start an interactive batch session, use

```
qsub -I
```

If you need a GUI, you should use:

```
qsub -I -X
```

By default, interactive sessions go to the "interactive" queue, which gives them high priority, 1 CPU and 2GB memory per CPU. You can explicitly request more CPUs and more memory with the `-l` options, such as:

```
qsub -I -l nodes=1:ppn=4 -l mem=10GB
```

**MPI jobs**

For MPI parallel jobs you must specify how many cpus you need:

```
## use 4 CPUs, I don't care if they are on the same node or not  
#PBS -l procs=4

## use 4 CPUs all on the same node (eg, for an OpenMP job with no MPI):  
#PBS -l nodes=1:ppn=4

## use 4 CPUs, in groups of 2-on-the-same-node (all three groups might land on the same physical node)  
#PBS -l nodes=3:ppn=2

## use 2 CPUs on each of 2 distinct nodes. I really need them to be on different nodes!  
#PBS -l nodes=2:ppn=2 -W x=xmatchpolicy:exactnode
```

**Running MPI jobs with OpenMPI and MVAPICH2**
Mercer has two implementations of MPI: OpenMPI and MVAPICH2. Overall the performance and behavior of each is similar, but with either there are special things to consider:

- With OpenMPI, processes may change which CPU (within the node) they use during execution. This can reduce performance, so OpenMPI jobs should be started with the `--bind-to-core` flag:

  ```
  mpirun --bind-to-core -np $PBS_NP ./my_program
  ```

- With either MPI implementation, the default behavior is to launch each process on the next available CPU. For hybrid MPI/OpenMP programs, this causes incorrect behavior, as each MPI process needs multiple CPUs for its OpenMP threads.

  For example, if you need two MPI processes with 2 OpenMP threads each, you would request `-l nodes=2:ppn=2`. Your job might then be allocated 2 CPUs on node 14-0 and 2 CPUs on node 14-1. Your nodefile (`$PBS_NODEFILE`) will then look like:

  ```
  compute-14-0.local
  compute-14-0.local
  compute-14-1.local
  compute-14-1.local
  ```

  MPI uses this to place its processes, so it will, by default, place the first process on `compute-14-0.local` and the second process on `compute-14-0.0local`. But for a hybrid program, you have multiple threads (in this case, 2) for each MPI process. Threads cannot cross host boundaries, so you will have 4 threads sharing the 2 CPUs of `compute-14-0.0local`, while `compute-14-1.0local` sits idle.

  To get the correct behavior with OpenMPI, add `--bynode` to the mpirun command:

  ```
  mpirun --bind-to-core --bynode -np $PBS_NP ./my_program
  ```

  With MVAPICH2, you must use a modified nodefile to reserve CPUs for OpenMP threads. The following idiom is effective:

  ```
  #!/bin/bash
  #PBS -l nodes=2:ppn=20
  cd $SCRATCH/my_run_dir
  module load mvapich2/intel/2.0rc1
  # we want 20 threads per MPI process:
  export OMP_NUM_THREADS=$PBS_NUM_PPN
  # make a new nodefile with only two entries, one for each MPI process:
  sed -n "1~${PBS_NUM_PPN}p" $PBS_NODEFILE > my_nodefile
  mpirun -f my_nodefile -np $PBS_NUM_NODES my_program
  ```

**GPU Jobs**

To request GPU nodes:

- `-l nodes=1:ppn=1:gpus=1`
  - 1 node with 1 core and 1 GPU
- `-l nodes=1:ppn=1:gpus=1:titan`
  - 1 node with 1 core and 1 GPU, specifically a Titan Black GPU
- `-l nodes=1:ppn=1:gpus=1:k80`
  - 1 node with 1 core and 1 GPU, specifically an Nvidia K80 GPU
- `-l nodes=1:ppn=4:gpus=4:titan`
  - 1 node with 4 Titan GPUs. Note that we request `ppn=4` too, it is always best to request at least as many CPU cores are GPUs

The available GPU node configurations are shown [here](#).
When you request GPUs, the system will set two environment variables - we strongly recommend you do not change these:

- CUDA_VISIBLE_DEVICES has a comma-separated list of the device IDs this job is allowed to use (eg "2,3"). The CUDA library within the application will use this to prevent multiple GPU jobs on the same node from interfering with each other.
- CUDA_DEVICES has a zero-based sequence of the "logical device IDs" for your job (eg "0 1"). So, if your application expects a list of GPU IDs starting at zero, and you have been allocated GPU numbers 2 and 3, then you can pass $CUDA_DEVICES to your application and it will see 2 devices, named "0" and "1", which happen to correspond (via $CUDA_VISIBLE_DEVICES) to the GPUs whose physical IDs are "2" and "3"

To your application, it will look like you have GPU 0,1... (up to as many GPUs as you requested). So if for example, you request 2 GPUs, and are allocated GPU 2 and GPU 3, you will have:

```
# echo $CUDA_VISIBLE_DEVICES
2,3
# echo $CUDA_DEVICES
0,1
```

Now if your application calls "cudaSetDevice(0)", you will use the GPU that appears as device 0, but is actually device 2.

And a call to "cudaSetDevice(3)", will return an error, because as far as the application can see, the node only has 2 GPUs, numbered 0 and 1.

**Fast local disk on Mercer**

Batch jobs on Mercer can access the local disk on their node through $PBS_JOBTMP. There is one local disk per physical node, so if you use "-l nodes=3:ppn=10", and you job is run on 10 cores of one node, another 10 cores on that same physical node and 10 cores on a different node, then $PBS_JOBTMP will be the same for the first two "nodes" and different for the third one. (Because the first two "nodes" share a physical node)

If your job will do many small reads and writes, you might benefit by using a memory filesystem. To do this:

```
#PBS -l other=memdisk
```

This will give you an 8GB local filesystem, accessible within the job script via $PBS_MEMDISK.

If you don't request memdisk, $PBS_MEMDISK will be an alias for $PBS_JOBTMP.

Both $PBS_JOBTMP and $PBS_MEMDISK are deleted at the end of the job, so you must copy any data you wish to keep back from them before the end of your job script.

**Software on Mercer**

Mercer uses the same Environment Modules setup as Bowery. Not all software from Bowery is on Mercer yet, in particular only the most recent version of most software has been installed.

Different versions of software may give slightly different results, especially if your model is numerically sensitive! Please carefully check the results of simulations on Mercer and compare them to results on the cluster you used previously!