How to use the NC State High Performance Computing (HPC) Facility

Statistical Ecology and Forest Science Lab
North Carolina State University

Fall, 2025

Contents

What is an HPC?

A High-Performance Computing (HPC) system is a network of powerful computers (called nodes) that work together to solve computationally intensive problems much faster than a standard desktop or laptop computer.

How does it work?

Think of an HPC like a factory assembly line rather than a single workbench:

1. **Login nodes:** Where you prepare your work (write scripts, organise files)
2. **Compute nodes:** Where the actual heavy computation happens (workhorse)
3. **Job scheduler:** The manager that assigns work to available compute nodes
4. **Storage systems:** Shared file systems where all nodes can access your data

NC State's Cluster

NC State's HPC cluster is officially called Hazel, though you'll also see it referred to as Henry2. Hazel represents the upgraded version of the cluster running modern software (RHEL 9.2 instead of the older CentOS 7.5). Both names refer to the same system, and you'll see both used in documentation (official NC State HPC tutorials) and when acknowledging the cluster in publications. Hazel uses LSF (Load Sharing Facility) as its job scheduler. This determines the script commands/syntax used when communicating with the cluster.

NOTE: The Office of Information Technology (OIT) at NC State has some great YouTube tutorials on using the HPC, be sure to check them out too!

Citing the HPC

If you have used the HPC at any stage in your research, you should acknowledge these compute resources in all works (theses, publications, etc.): "*We acknowledge the computing resources provided by North Carolina State University High Performance Computing Services Core Facility* (RRID:SCR_022168)."

Key terms to know

Term	Definition
Infrastructure Terms	
Cluster	The entire HPC system consisting of many connected computers
Node	A single computer within the cluster
Login node	The computer you connect to when you SSH into the HPC (for preparing work only)
Computer node	Nodes where your actual computations run
Core/CPU	Individual processing unit within a node (modern nodes have 8−32+ cores)
GPU	Graphics Processing Unit, specialised for parallel calculations
Job Scheduling Terms	
Job	Computational task you submit to run on the cluster
Batch job	Job that runs without user interaction using a script
Interactive job	Job where you can type commands in real-time
Queue	Waiting line for jobs before they run
Job script	File containing instructions for what your job should do
Wall time	Maximum real-world time your job is allowed to run
Storage Terms	
Home directory	Your personal space for scripts and small files ($\sim$ 1GB)
Scratch space	Temporary high-performance storage for active computations ($\sim$ 20TB per project)
Project directory	Shared space for research group members
Job Status Terms	
PEND	Job is waiting in queue for resources

Term	Definition
RUN	Job is currently executing on compute nodes
DONE	Job completed successfully
EXIT	Job finished with an error

Basic storage on Hazel

There are several different places on Hazel that are available for keeping files. These have different amounts of space available and different intended purposes.

Home Directory

When you initially connect to Hazel, your working directory is your home directory `/home/user_name` (or shorthand `~`). **There is a 1GB quota!** Your home directory should be used only for storing scripts, small applications, and temporary files needed by the scheduling system to run your jobs. Your home directory is backed up daily to another data centre.

Scratch Directory

There is a 20TB scratch directory quota. The scratch directory is where data for running jobs and results are stored. Your scratch directory is located at `/share/group_name/user_name`. You can find your group name with the command `echo $GROUP`. Jobs should be submitted from `/share` or written in such a way that they by default read/write to `/share`. Any important files should be moved at the end of a run. Scratch space is not backed up, and files not accessed for 30 days are automatically deleted.

Application Directory

If you (or your group) are installing your own applications, they should be kept in the application directory (unless they are small enough to fit in your home directory). Each project may request an application directory. Your project's application directory would be located at `/usr/local/usrapps/group_name`. Projects receive a 100GB default application directory quota.

Research Storage

Research storage is accessible from all Hazel nodes via `/rs1` or `/rsstu`. The recommended workflow is to copy data from Research Storage to your scratch directory at the beginning of your job script. Then, at the end of your run, copy results back from your scratch directory to Research Storage. This storage must be requested, and generally starts at 2TB, but you can request up to 30TB.

Using the HPC

Step 1: Requesting Access

To use the HPC, a faculty member must first request a project. Students may then be added to a project by their faculty advisor. Graduate students, postdocs, and

other collaborators must be added to a project by the faculty PI.

Step 2: Required software

You can connect to the HPC using a web-based interface (e.g., Open OnDemand) or through a terminal window on your computer (recommended).

- macOS/Linux
  - No additional software is required. Can work directly from "terminal".
- Windows
  - MobaXterm (recommended by NC State)
    - Download and install the MobaXterm Home Edition (Installer edition).
    - To install the text editor nano, open MobaXterm, click **Start local terminal**, and type `apt-get install nano`
      - This is needed to create and edit configuration files, scripts, and code directly on the server.
  - Windows Terminal
    - This is only an option if you can use an administrative shell. Click on the Windows icon, search for "Terminal", right click on the terminal (or PowerShell) icon and select "`Run as administrator`". If you are able to open up a shell then you can continue. If not, then use MobaXterm above.
    - We still need to install the text editor nano. The easiest way to install the nano on Windows is to go through Chocolatey. If you don't have Chocolatey installed, enter the following command to install it:

```
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex
((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

    - Now to install nano, enter the command: `choco install nano -y`

Step 3: Connecting to Hazel

To connect to the shared HPC login nodes via a secure shell (ssh), open a terminal window and type `ssh <user_name>@login.hpc.ncsu.edu`, where `<user_name>` is your NC State Unity ID.

You will then have to supply your password (same as with all Unity ID logins), and then you will be asked to complete a Duo two-factor login, which can be done either by receiving a Duo Push (usually option 1) or SMS passcode (option 2). Annoyingly, if you supplied the wrong password, you will only find out *after* the Duo verification, and you will then have to do it all again.

Once you have successfully logged in, you will see welcome and/or warning messages at the top of the terminal. At the bottom-left of the screen, you should see that you are no longer logged into your computer but rather logged into a login node part of NC State's Hazel cluster: `[user_name@loginXX ~]$`, where `XX` is the reference ID of the specific login node, and ~ is shorthand for your home directory.

NOTE: Hazel can only be accessed when connected to university Wi-Fi (eduroam). If you want to access the cluster from home, then a virtual private network (VPN) is required. NC State provides a VPN connection through the Cisco Secure Client software.

Step 4: Organising and transferring files

There are a few ways to move files to or from Hazel.

1. **Terminal window:** From a terminal window the `sftp` command can be used to transfer files between your local computer and Hazel.
2. **Web browser:**
   - Open OnDemand: The file browser widget available in Open OnDemand provides an easy way to move files up to 10GB between your local computer and Hazel.
   - Globus: Globus is the preferred tool to use for moving larger files. It can restart interrupted network sessions and compute a checksum at the end of the transfer to ensure the data was moved correctly. The location you are moving data to/from will need to be connected to a Globus Connect Server or have Globus Personal Connect installed.

SFTP

SFTP (Secure File Transfer Protocol) is a network protocol for transferring files using your terminal window (e.g., MobaXterm). Like logging into the HPC, you also need to log in to SFTP: `sftp <user_name>@login.hpc.ncsu.edu`. Again, you will be prompted for your password and to complete a Duo verification. You will know you are in when you see `sftp>`. What's nice about SFTP is that it uses the same Linux commands we use with the HPC, such as `pwd`, `ls`, `mkdir`, `cd`, etc. See Table 2 for all basic/essential Linux commands needed to work on the HPC.

You can interact with your local computer by adding `l` before commands, e.g., `lpwd Desktop/` will print to your local desktop:

```
sftp> lpwd Desktop/
Local working directory:  /home/user_name/Desktop
...
sftp> pwd
Remote working directory:  /home/user_name
```

To exit SFTP, we use the command `bye`.

Local to Remote: `put`

To move files from your local machine to the HPC, we use the command `put <local> <remote>`, i.e., command followed by your local directory and then remote directory:

```
sftp> put ./data_for_hpc/data_file.txt ./
```

The code above will move "`data_file.txt`" from your local machine to the current working directory on the HPC (`./`), or you could specify a specific directory in the HPC.
You can also copy entire folders using `put -r <local> <remote>`, e.g.,

```
sftp> put -r ./data_for_hpc ./
```

Remote to Local: `get`
To move files the other way, i.e., to move model outputs from the HPC to your local machine to continue analyses, we use the command `get <remote> <local>`.

GLOBUS

Globas is a file-sharing/moving tool. It was created by the University of Chicago and is available to a variety of institutions.

Log in to Globus

Go to the Globus web app, look up your organisation (e.g., North Carolina State University). You will once again be prompted to log in with your university credentials and complete a Duo verification. Once completed, you will see a Windows-style web interface.

In the collection, search for "NC State Hazel HPC Cluster", and continue with your university credentials or Unity ID. This will open the contents of your home directory once again, i.e., the Path should be `/home/<user_name>/`. If not, you can navigate to your home directory by opening the "`home`" folder and then your "`user_name`" folder.

File Manager tab on Globus

1. Rename folders/files on the cluster by right-clicking on the folder/file, or by pressing the "`Rename`" button
2. Delete folders/files on the cluster by pressing the "`Delete`" button
3. Upload files from your computer to the cluster by pressing the "`Upload`" button, and then directly selecting files on your local machine.
4. Download files from the cluster by pressing the "`Download`" button, and this will initiate a regular web download into the Downloads folder of your local machine.
5. Navigate between folders by either editing the path name or by clicking on the different folders.

However, this has limited functionality, i.e., you can't download entire folders.

Globus transfers between endpoints

1. Under the "`File Manager`" tab, select "`Set two panes`" in the top right corner.
2. Then, in the second pane, you can search for "NC State Google Drive Connector" in the path search bar. You may need to do some additional authentication. You should see the home directory of your university's Google Drive in the right pane.
    - You can also connect to personal folders by downloading the app "`Globus Connect Personal`" which will allow Globus to connect to your local machine.

- If downloaded, you will see an icon in the menu bar at the bottom of your screen. You can right-click and select "`Web: Transfer Files`", which will take you straight to the two Globus panes for transferring files. You can then connect the second pane to the HPC by searching for "NC State Hazel HPC Cluster" again.

3. In both panes, navigate to where the file/folder that you want to transfer is located, and also the folder/path where you want the file/folder to be transferred to.
4. Select the "`Start`" button on the pane that contains the file/folder you are transferring. E.g., if you are transferring a file from the HPC, you would press "`Start`" on the panel connected to the HPC.
5. This creates a "Transfer Task" which can be monitored in the "Activity" tab to the right. You will also receive an email notification when the task is complete.
6. If you have already transferred a file, but now want to update it, you can go to the "`Transfer and Time Options`", tick the "`sync - only transfer new or changed files`", and then choose from different options depending on how you want these updates to function.
7. You can also schedule for transfers to occur regularly in the "`Transfer and Time Options`". Under "`Schedule Start`" you would set the date you want the transfer to start. If you want it to start today, you leave it as is and just press "`Start`". You can also set transfers to repeat at certain time intervals, e.g., days/hours, etc.

Step 5: Basic job scripts

Each time you want to run something on the HPC, you would follow very similar job submission steps:

1. Create/modify your batch script. This is a text file that contains the information necessary for the job scheduler to reserve the resources you need, as well as the actual lines of code that you are trying to run.
2. Navigate to job submission directory (usually `/share/$GROUP/$USER`). This is a location used for rapid reading and writing.
3. Submit using `bsub < [name of batch script]`
4. Check on job progress using `bjobs`
5. Examine outputs:
   - Intended program outputs.
   - Standard output (`stdout`), which is what your program prints when things go well.
   - Standard error (`stderr`), which is what your program prints when things go poorly.

1. Create/modify batch script

Start by creating a batch/shell script in your desired directory: `nano ~<path>/submit.sh`. You can also create shell scripts in any text editor and just save with a "`.sh`" extension.

Within your batch script, you would specify the following information:

```
#!/bin/bash
#BSUB -n [number of cores]
#BSUB -W [max time to complete]
#BSUB -q <queue>
#BSUB -J <name_of_job>
#BSUB -o stdout.%J
#BSUB -e stderr.%J
[set up environment]
[run program]
```

- The first line of the batch script, like all shell scripts, always starts with a shebang line, because it starts with a `#!`, followed by the shell, which is what you will use to run all your commands on the cluster, which is `/bin/bash`, i.e., the location of the bash shell executable.
- The next few lines, starting with `BSUB`, are for the job scheduler LSF. Each shell needs to include:
  1. `-n` followed by the number of cores. If you are running a serial job, you would specify 1 core. Parallel jobs will need more cores (>1). You need to make sure that your job doesn't use more cores than what you specify, as this can lead to issues with jobs not having the necessary resources (because it is being secretly used by another node/job).
  2. `-W` (note that W is a capital) followed by the maximum time to complete your job (seconds/minutes/hours) up to a max time of 72 hours. Once the time "runs out" your job will be shut down regardless of whether it is finished or not.
- The next few lines aren't required, but this is generally the minimum you want to include to be able to keep track of what jobs you ran.
  3. `-q` specifies which queue you want to join. Some colleges have dedicated queues you can join, such as CNR: `cnr`, which will greatly reduce the time spent waiting for the resources you need.
  4. `-J` (note that J is a capital) followed by a job name (no spaces). You can use this job name to identify this job relative to other jobs running at the same time.
  5. `-o` specifies the output file for the standard output of the job. In other words, whatever your program prints to the terminal normally will instead be written to the file you specify after the `-o`.

     `stdout.%J` This is the name of the output file, and `%J` is a job ID placeholder, which will be replaced with the actual numeric job ID. This prevents different job runs from overwriting each other's output.
  6. `-e` same as above but specifies the output file for the standard error. Note, even if your run goes well, if you specify a standard error output, an output file will still be created (it will just be empty).
  7. `[set up environment]` is a placeholder for lines of code to set up your environment. This could be loading in environmental modules or reading in certain config files.
  8. `[run program]` is a placeholder for lines of code to actually run your program.

Environmental modules
Setting up your environment will commonly (but not always) involve models that have been set up by the HPC staff. These are really helpful, and mean you don't have to maintain these modules yourself. However, sometimes these maintained modules can conflict with what you are working on. If that happens, then you can create these yourself using something like conda. However, using a conda environment can lead to path conflicts between modules, so be careful with that!

NOTE: You load modules to your home directory (NOT the job submission (shared) directory)

Some environmental module commands:

- `module list` : Check which modules are currently loaded
- `module avail` : Check which modules are available to be loaded
- `module load <module_name>/<version*>` : Loads in a specific module (where version is ∗optional)
- `module unload <module_name>` : Unloads a currently loaded module
- `module purge` : Unloads ALL currently loaded modules

Loading in modules is required to add specific directories to your path. E.g., you need to load in `R` as a module before you can use it:

```
# First, try running R before the module is loaded
$ R
-bash:  R: command not found

# Load the R module
$ module load R

# Run R again
$ R

R version 4.3.2 (2025-12-05) -- "Hypothetical Example"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform:  x86_64-pc-linux-gnu (64-bit)
...

# Now you are in a working R console
> 1 + 1
[1] 2

# Install any R packages you may need
$ install.packages("spOccupancy")

# Use quit() to exit your R console
> quit
Save workspace image?  [y/n/c]:  n
```

Running a program: R script
In the `[run program]` placeholder in our batch/shell script above, we would use the `Rscript` command to specify the `R` script that contains the code we want to run.

```
# Run R script
Rscript <path/><name_of_script.R>
```

NOTE: that if your `R` script uses any packages outside of base `R` then you need to install these packages on the module, see in code block of "Environmental variables". If the package has quite a few dependencies, then this can take a while... It is generally recommended that you limit the number of packages in any script run through the HPC, i.e., do your data wrangling on your local machine (using `dplyr`, `sf`, etc.), and then just load your formatted data (`.rda`) to the HPC.

2. Job submission directory

Next, we must navigate to our job submission (scratch) directory: `cd /share/$GROUP/$USER`.

For Dr Doser's lab, we can specify: `cd /share/doserlab/<user_name>`, where `user_name` is your NC State Unity ID. If you want to make a sub-directory/folder within this scratch directory (which makes it easier to keep track of jobs/outputs), use: `mkdir <folder_name>`

3. Submit job

Now, we can submit our job using `bsub < ~<path>/submit.sh`. The < redirects your specific file to `bsub` as the input.

```
$ bsub < ~<path>/submit.sh

Default memory limit -R rusage[mem=2.00/task] was added to job
resource request

Job <XXX> is submitted to default queue <debug>
```

Note that your default memory is 2GB per task, which translates to 2GB per core that you reserve. If you want to reserve more memory per task (e.g., 10GB), you can specify this in your batch/shell script:

```
#BSUB -R "rusage[mem=10]"
```

4. Job progress

You can check on your job progress using `bjobs`, this will print general information about the job(s) currently running:

```
$ bjobs

 JOBID  USER     STAT  QUEUE  FROM_HOST   EXEC_HOST  JOB_NAME  SUBMIT_TIME
 12345  UnityID  RUN   debug  loginX.hpc  sd3        script.R  Dec 08 10:38
```

You can check what the different job status (`STAT`) terms mean in Table 1.

5. Examine outputs

We can now explore the various outputs from our run:

- Standard output and job details (`stdout.[job_number]`), which will include information such as which directory was used, start date/time, end/terminate date/time, and some resource usage summaries (i.e., CPU time, average memory, etc.)
  - We can use `less <file_name>` to view and scroll through the different files
- Job errors (`stderr.[job_number]`), which will be empty if everything ran correctly, or will contain error messages which can be used to debug/fix your batch or `R` script.
- Script outputs/products such as `.rda` from a model, etc.
  - Remember, your scratch space is not backed up, and files not accessed for 30 days are automatically deleted. Therefore, it is good practice to download or move outputs to research storage after a successful run.

Step 6: More complex job scripts

When running large models, you'll often hit a RAM ceiling. Instead of forcing one massive job to run three chains (e.g., `n.chains = 3` in `R`), it's much more efficient to run three independent jobs. You could submit these manually, but **Job Arrays** do the heavy lifting for you by automating the submission and naming process. For example:

```
#!/bin/bash
#BSUB -n [number of cores]
#BSUB -W [max time to complete]
#BSUB -q <queue>
#BSUB -J <name_of_job>[1-3]
#BSUB -o stdout.%J.%I
#BSUB -e stderr.%J.%I

module load openmpi-gcc
module load R

Rscript ~/model_script.R $LSB_JOBINDEX
```

You can see a couple of changes to this batch script compared to our basic batch script in Step 5.

1. `[1-3]` Tells LSF, "I want three versions of this job, indexed 1, 2, and 3."
2. `.%I` This acts as a placeholder. LSF swaps it for the index number so your log files stay organized and don't overwrite each other.
3. `$LSB_JOBINDEX` An environment variable that LSF "plugs into" your `R` command. For the first job, it becomes 1; for the second, 2, and so on.

Now in your `R` script, you would have the following lines of code at the top of your script:

```
 args <- commandArgs(trailingOnly = TRUE)

if (length(args) == 0) {
chain_idx <- 1
} else {
chain_idx <- as.numeric(args[1])
}

set.seed(123 + chain_idx)
```

This allows the script to be "self-aware", i.e., it knows which part of the task it is performing based on what the HPC tells it.

1. `args <- commandArgs(trailingOnly = TRUE)` This captures any text you type after `Rscript model_script.R` in your batch script (i.e., the `$LSB_JOBINDEX` we passed from the batch script.).
2. `chain_id <- if(length(args) > 0) as.numeric(args[1]) else 1` This is a bit of "safety" code. It grabs the index number from the HPC, but if you're just testing locally, it defaults to 1 so the script doesn't crash.
3. `set.seed(123 + chain_id)` This is the most important part. By shifting the seed based on the chain ID, you ensure each job explores the posterior from a different starting point, maintaining statistical independence for your Bayesian model.

Essential Linux commands

Navigating and working within the cluster requires the use of Linux commands. For example, instead of using a click-and-drag framework to navigate to and select files (like Windows or Mac), we need to use a text-based interface, where we provide explicit commands.

Linux	Descriptions
General	
`ssh`	Secure shell used to log in and communicate with remote machines.
`clean`	Clear your terminal. This is useful to get rid of welcome/warning messages that might be cluttering up your terminal.
`whoami`	Print username.
`hostname`	What system/node am I on? (login vs compute)
`pwd`	Print working directory.
`echo`	Display a line of text or string as output on the terminal, e.g., `echo $GROUP` will print your group name.
`man`	Searches for the command manual. Must be followed by the command of interest, e.g., `man pwd`.
Working with directories	
`ls`	Show contents of a directory/folder. Additional tags: `-l` : long format, gives additional info about each file. `-h` : human readable sizes. `-t` : sort by time, newest first. `-r` : reverse sorting (e.g. `-tr` makes it oldest first).
`grep`	Search for text or regular expression, e.g., `ls | grep txt` will look in your current directory for matches to "`txt`".
`mkdir`	Make a new directory
`cd`	Change the directory (i.e., move to this directory.) Like in R, you can use tab to quickly complete filenames. Your current directory is shown in the square brackets along with your username: `[user_name@loginXX scripts]`. `cd .` Go "up" a directory level. `cd ..` Called "up up", return to parent directory.
`rmdir`	Remove empty directory, will get an error if non-empty.
Working with text files	
Note: all commands should be followed by file name: `<command> <file_name.txt>` .	
`nano`	Used to create/edit text files. **ctrl X**: close and save file. **y**: yes, then hit **return/Enter** to confirm.
`cat`	Print contents of file to screen (directly to terminal).
`head`	Print first few lines of file.
`tail`	Print last few lines of file.
`less`	Open file and scroll through contents. `q` to exit.

Linux	Descriptions
`mv`	Move/rename file (can be moved to a new location), e.g., `mv <old_name> <new_name>`
`cp`	Copy file with new name. Add `-r` to copy the directory with all contents.
`rm`	Remove a file (be careful – you can't undo!)
Running jobs	
`bsub`	Command for submitting batch or interactive jobs.
`bjobs`	Displays information about jobs, which can be paired with tags for filtering: `-a` Job in all states, including jobs that finished recently. `-d` Only jobs that finished recently. `-r` Only running jobs. `-l` Long format, i.e., multi-line format.
Storage	
`quota`	Allows you to look up how much space is available to you. `-s` to make it "human readable" or in relevant units (KB/MB/GB). `-g <group_name>` to get quota for your group.
`du <path>`	Prints total size of path contents. Used to check what is using most of your storage, this is helpful if you are getting close to your quota. `-s` summarise (total) `-h` human readable (KB/MB/GB)