Puma Quick Start

Getting started

If you’re just getting started with HPC, you may also want to check out a video recording of our Intro to HPC workshop in addition to this quick start.

This page is designed to give users an overview of how to run their work on our systems. By the end of this, you should know:

1. What HPC is
2. How to log in
3. What the bastion host, login nodes, and compute nodes are
4. What a job scheduler is
5. How to access software
6. How to run interactive and batch jobs on HPC

If you have not already, you will need to Register for an HPC Account to follow along.

What is HPC?

If you’re looking to tackle complex problems or speed up your data analyses, HPC might be just what you need!

HPC is an acronym for High Performance Computing and is often used interchangeably with supercomputing. As a UArizona affiliate, you can be sponsored by a faculty member (faculty members can sponsor themselves) to receive free access to our three supercomputers Puma, Ocelote, and ElGato. These are clusters of computers that are housed in the lower level of the UITS building and are available for your analyses.

If you’re interested in a more in-depth overview of what a supercomputer is, see our page Supercomputing In Plain English.
Logging in

HPC Credentials

HPC access does not require its own set of credentials. To log in, all you’ll need is your **UA**rizona NetID and password with two-factor authentication enabled.

How to Connect

Despite HPC being housed in the UITS building, you can connect to it from any local workstation either by using a local terminal, or by using our web portal. In this quick start, we’ll use the terminal.

The steps for connecting depend on your operating system. Choose the correct panel below for instructions:
**Mac/Linux Instructions**

**Connecting with Mac/Linux**

Use the Terminal (on a Mac, this can be found under **Applications Utilities Terminal**). On the command line, enter the following, replacing `netid` with your own NetID:

<table>
<thead>
<tr>
<th>Establishing a connection</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ssh netid@hpc.arizona.edu</code></td>
</tr>
</tbody>
</table>

You will then be prompted for your password. Note: when you enter it, you will not see characters appear. This is a security feature and completely normal.

After a successful login, you will be connected to the **bastion host**. This is a single computer that provides a gateway to our three clusters. This is the only function the bastion host serves. It is not for storing files, running programs, or accessing software. The hostname of this machine is `gatekeeper`.

<table>
<thead>
<tr>
<th>Accessing the bastion host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success. Logging you in...</td>
</tr>
<tr>
<td>Last login: This is a bastion host used to access the rest of the RT/HPC environment.</td>
</tr>
<tr>
<td>Type &quot;shell&quot; to access the job submission hosts for all environments</td>
</tr>
</tbody>
</table>

----------------------------------

```
[netid@gatekeeper 13:53:23 ~]$ hostname
gatekeeper.hpc.arizona.edu
[netid@gatekeeper 13:53:26 ~]$
```

**Windows Instructions**

**Connecting with Windows**

You will need an ssh client such as PuTTY. Open a connection and enter `hpc.arizona.edu` under Host Name and press Open. This will open a terminal. At the prompt, enter the following, replacing `netid` with your own NetID:

<table>
<thead>
<tr>
<th>Establishing a connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login as: netid</td>
</tr>
</tbody>
</table>

You will then be prompted for your password. Note: when you enter it, you will not see characters appear. This is a security feature and completely normal.

After a successful login, you will be connected to the **bastion host**. This is a single computer that provides a gateway to our three clusters. This is the only function the bastion host serves. It is not for storing files, running programs, or accessing software. The hostname of this machine is `gatekeeper`.

<table>
<thead>
<tr>
<th>Accessing the bastion host</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success. Logging you in...</td>
</tr>
<tr>
<td>Last login: This is a bastion host used to access the rest of the RT/HPC environment.</td>
</tr>
<tr>
<td>Type &quot;shell&quot; to access the job submission hosts for all environments</td>
</tr>
</tbody>
</table>

----------------------------------

```
[netid@gatekeeper 13:53:23 ~]$ hostname
gatekeeper.hpc.arizona.edu
[netid@gatekeeper 13:53:26 ~]$
```
Login nodes

Once you connect to the bastion host, you'll notice it prompts you to type `shell`. Try typing it now. You should see:

```
[netid@gatekeeper 13:53:45 ~]$ shell
Last login: Fri Apr 28 15:30:08 2023 from gatekeeper.hpc.arizona.edu
***
The default cluster for job submission is Puma
***
Shortcut commands change the target cluster
-----------------------------------------
Puma:
$ puma
(puma) $
Ocelote:
$ ocelote
(ocelote) $
ElGato:
$ elgato
(elgato) $
-----------------------------------------
(puma) [netid@junonia ~]$
```

Note that the hostname has now changed. Where it used to say gatekeeper, it should now say either `wentletrap` or `junonia`. These are the HPC login nodes.

A login node is a small computer that serves as a staging area where you can perform housekeeping, edit scripts, and submit your work to run on the system. It's essential to know that the login nodes are not the location where your analyses are run. Instead, the cluster's compute nodes are where the real work is done.

Compute nodes
What are compute nodes?

In contrast to the login nodes, compute nodes are high-performance machines designed for computationally intensive applications that require a significant amount of processing power and memory. For example, a standard compute node on Puma has 94 CPUs available and 470GB of RAM!

Different clusters, different compute nodes

Each of our supercomputers has its own cluster of compute nodes with different resources available. When you first log into HPC, your environment is configured to submit jobs to our largest (and busiest cluster), Puma. You can see which cluster you're targeting by looking at the beginning of your command line prompt. To switch your target cluster, use one of the following shortcuts: `elgato`, `ocelote`, or `puma`.

For this tutorial, let's switch clusters by entering the shortcut `elgato`. This cluster is less busy than the others so you can get to development fast.

```
(puma) [netid@wentletrap ~]$ elgato
(elgato) [netid@wentletrap ~]$
```

How do you actually access a compute node?

To connect to a compute node, you will need to use the job scheduler. A scheduler, in our case SLURM, is software that will find and reserve resources on a cluster's compute nodes as space becomes available. Resources include things like memory, CPUs, and GPUs that you want to reserve for personal use for a specified period of time. You can use the job scheduler to request two types of jobs: interactive and batch. We will cover both of these in the sections below.

All there any limitations to what I can request?

Yes, there are some limits to what you can request. One of the important limits to understand is your group's CPU hours allocation.

Allocations are a way of being "charged" to use HPC resources. Each group gets:

- 7,000 CPU hours on ElGato
- 70,000 CPU hours on Ocelote
- 100,000 CPU hours on Puma

These allocations are automatically refreshed on the first day of each month. Once your group's allocation runs out, you will need to wait until it is refreshed before using that allocation again.

Your group's account is charged for each request made to the scheduler, in other words for each job you submit. The amount charged is the number of CPUs requested multiplied by the number of hours reserved. For example, if you submit a job that requests 5 CPUs for 10 hours, your group will be charged 50 CPU hours. If your job ends early, you will be refunded any unused time.

To see your group's allocation, use the command `va` which stands for view allocation. For example:
Interactive jobs

Let’s start with a basic interactive job to get a feel for things.

Starting a session

To connect to a compute node to work interactively, use the command `interactive -a your_group` replacing `your_group` with your own group’s name that you found from running `va`. In my case, I’m going to use `hpcteam`.

Virtual Allocating (salloc)

```
(elgato) [netid@wentletrap ~]$ interactive -a hpcteam
Run "interactive -h for help customizing interactive use"
Submitting with /usr/local/bin/salloc --job-name=interactive --mem-per-cpu=4GB --nodes=1 --ntasks=1 --time=01:00:00 --account=hpcteam --partition=standard
salloc: Pending job allocation 807340
salloc: job 807340 queued and waiting for resources
salloc: job 807340 has been allocated resources
salloc: Granted job allocation 807340
salloc: Nodes gpu66 are ready for job
[netid@gpu66 ~]$
```

By default, when you use the command `interactive`, you will be allocated one CPU for one hour. To request custom resources (more CPUs, more time, etc), the interactive command can be modified.

That’s it! You’re now connected to a compute node (in my case, gpu66) and are ready to run some work.

Let’s check out some software

Software packages are not available on the login nodes but are available on the compute nodes. Now that we’re connected to one, we can see what’s available. Software on HPC comes installed as modules. Modules make it easy to load and unload software from your environment. This allows hundreds of packages to be available on the same system without dependency or versioning conflicts. It’s always good practice to specify which version of the software you need when loading to ensure a stable environment.

You can view and load software modules using the command `module avail` and `module load`, respectively. For example,

```
[netid@gpu66 ~]$ module avail python
------------------------------------------------- /opt/ohpc/pub/modulefiles
--------------------------------------------------
   python/3.6/3.6.5   python/3.8/3.8.2 (D)   python/3.8/3.8.12   python/3.9/3.9.10
[netid@gpu66 ~]$ module load python/3.9
[netid@gpu66 ~]$ python --version
Python 3.9.10
```
Try running `module avail` without specifying any arguments. You'll notice we have a lot available.

**Benefits of interactive sessions**

Interactive sessions are excellent development environments. When connected to a compute node, some things you can do are:

- Run, test, and debug your code
- View, test, and use software
- Install your own software
- Run computationally-intensive commands that might impact others on the login nodes

**Drawbacks of interactive sessions**

Interactive sessions are great testing and development environments, but may not be optimally suited for certain types of analyses. Some issues that may arise include:

- Your session may time out due to inactivity
- Your internet connection may get disrupted
- Your computer may get closed or turned off
- You want to run more than one job at a time

What's a good solution to deal with these challenges? The answer: **batch jobs**!

---

**Batch jobs**

**The basics**

Batch jobs are a way of submitting work to run on HPC without the need to be present. This means you can log out of the system, turn off your computer and walk away without your work being interrupted. It also means you can submit multiple (up to 1000) jobs to run simultaneously!

Running these sorts of jobs requires two steps:

1. Create a text file with three sections:
   a. The header. This is called a shebang and goes in every batch script. It tells the system which interpreter to use (i.e., which language to execute the instructions in):

   **Part 1: The shebang**

   ```bash
   #!/bin/bash
   ```

   b. Instructions that tell the job scheduler the resources you need and any other job specifications. This section will look like:

   **Part 2: The SBATCH directives**

   ```bash
   #SBATCH --option1=value1
   #SBATCH --option2=value2
   . . .
   ```

   c. A blueprint of how to run your work. This includes all the commands you would need to run in the terminal. This section might look like:

   **Part 3: The blueprint**

   ```bash
   cd /path/to/directory
   module load python/3.9
   python3 some_script.py
   ```

2. Submit the text file to the scheduler with the command `sbatch`.

Let's try creating our first job now using the outline provided above.
Creating the sample code

Let's start by creating a simple Python script that we'll run in batch.

Create a directory and a blank file

```bash
[netid@gpu66 ~]$ mkdir ~/hello_world && cd ~/hello_world
[netid@gpu66 hello_world]$ touch hello_world.py
```

Now open the file in your favorite text editor (for example, `nano` or `vim`) and add:

**Sample Python script**

```python
import os

hostname = os.uname()[1]
print("Hello world! I'm running on hostname %s"%hostname)
```

Then save and exit. If we run this interactively, we'll see

```
[netid@gpu66 hello_world]$ python3 hello_world.py
Hello world! I'm running on hostname gpu66.elgato.hpc.arizona.edu
```

Creating the batch script

Now, let's make a new file called `hello_world.slurm`

Create the `hello_world.slurm`

```bash
[netid@gpu66 hello_world]$ touch hello_world.slurm
```

Now open it in your favorite text editor.

**Step 1: Add the shebang and SBATCH directives**

In this example, we're using the standard partition. A partition is a job queue and affects a job's priority and how hours are charged. More information on partitions/job queues can be found under [Allocation and Limits](#) and in our [SLURM documentation](#). For now, let's stick to standard.

A comprehensive list of all the options you can specify in your batch script can be found on our [Running Jobs with SLURM](#) page. In this example, we'll stick with some of the basics:
Step 2: Add your code instructions

After the SBATCH directives, we'll add the instructions for executing our code to the same file

```bash
#!/bin/bash

# PART 2: Executes bash commands to run your job
# Load required modules/libraries if needed
module load python/3.9

# change to your script’s directory
cd ~/hello_world

# Run your work
python3 hello_world.py
sleep 10

Now save and exit.
#!/bin/bash

# --------------------------------------------------------------
### PART 1: Requests resources to run your job.
# --------------------------------------------------------------
### Optional. Set the job name
#SBATCH --job-name=hello_world
### Optional. Set the output filename.
#SBATCH --output=%x-%j.out
### REQUIRED. Specify the PI group for this job. Replace <PI GROUP> with your own group.
#SBATCH --account=<PI GROUP>
### REQUIRED. Set the partition for your job. This is a job queue
#SBATCH --partition=standard
### REQUIRED. Set the number of nodes
#SBATCH --nodes=1
### REQUIRED. Set the number of CPUs that will be used for this job.
#SBATCH --ntasks=1
### REQUIRED. Set the memory required for this job.
#SBATCH --mem-per-cpu=5gb
### REQUIRED. Specify the time required for this job, hhh:mm:ss
#SBATCH --time=00:01:00

# --------------------------------------------------------------
### PART 2: Executes bash commands to run your job
# --------------------------------------------------------------
### Load required modules/libraries if needed
module load python/3.9
### change to your script’s directory
cd ~/hello_world
### Run your work
python3 hello_world.py
sleep 10

Submitting the job

In this tutorial, we are submitting our job from an interactive session on a compute node. You may also submit jobs from a login node.

The next step is to submit your job request to the scheduler. To do this, you’ll use the command sbatch. This will place your job in line for execution and will return a job ID. This job ID can be used to check your job’s status with squeue, cancel your job with scancel, and get your job’s history with job-history. A more comprehensive look at job commands can be found in our documentation on monitoring your jobs.

Let’s run our script and check its status (substitute your own job ID below where relevant):

Submitting a batch job and checking its status

[netid@gpu66 hello_world]$ sbatch hello_world.slurm
Submitted batch job 807387
[netid@gpu66 hello_world]$ squeue --job 807387

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>807387</td>
<td>standard</td>
<td>hello_wo</td>
<td>netid</td>
<td>PD</td>
<td>0:06</td>
<td>1 gpu66</td>
</tr>
</tbody>
</table>

You can see its state is PD (for pending) which means it’s waiting to be executed by the system. Its state will go to R when it’s running and when the job has completed running, squeue will return a blank line.

Let’s check the contents of our file with cat. If your run was successful, you should see:
Checking the job's output

```
[netid@gpu66 hello_world]$ cat hello_world-807387.out
Hello world! I'm running on hostname gpu67.elgato.hpc.arizona.edu
[netid@gpu66 hello_world]$
```

Note that the hostname in this run is different from the hostname of the computer we're connected to. This is because it's a separate job from our interactive session and so may run on any other applicable machines on the cluster.

Additional resources

That's it! You've now successfully run both a batch and interactive job on HPC. To continue learning about HPC, our online documentation has a lot more information that can help get you started. For example FAQs, additional SBATCH directives, information on HPC storage, and file transfers.

Other great resources include: virtual office hours every Wednesday from 2:00-4:00pm, consultation services offered through ServiceNow, an examples Github page with sample jobs, and a YouTube channel with training videos.