Puma Quick Start

Getting Started

If you're just getting started with HPC, you may also want to check out our Intro to HPC workshop in addition to this quick start. A video recording is available from our training page.

This page is designed to give both new and experienced users an overview of how to run their work. By the end of this, you should know:

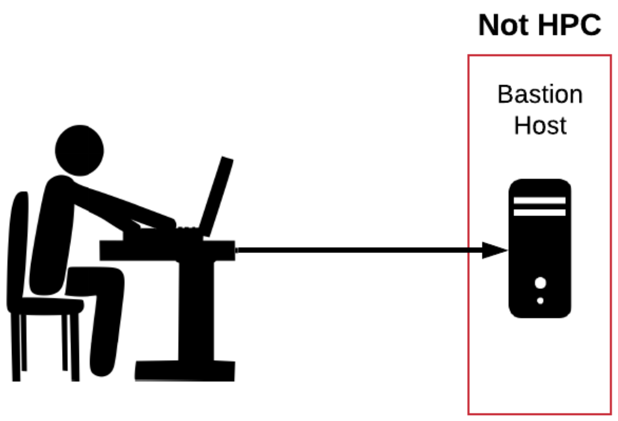
1. How to log in
2. What a login node is
3. What a job scheduler is
4. How to access software
5. How to run a job on HPC

If you have not already, you will need to Register for an HPC Account to follow along. If you encounter any issues, refer to our FAQ page which provides some answers to common user problems.

Being a Good User

Before using HPC, it's important to know that it is a shared system and what you do can affect others. Below, we'll cover some of the basics of appropriate system use to help you develop good habits. Additionally, take a look at our page on Best Practices to familiarize yourself with some of the system's do's and don'ts.

Logging In



To start with the tutorial, we'll log into HPC. To do this, you will need your UArizona NetID and password with two-factor authentication enabled. Logging in will first connect you to something called the **bastion host**, a computer that provides a gateway to our three clusters: Ocelote, ElGato, and Puma. This is the **only** function the bastion host serves. It is not for storing files, running programs, or accessing software.

The steps for connecting via the command line depend on your operating system. You can find more detailed information on logging in on the System Access page.

Mac or Linux users will use the Terminal (on a Mac under Applications Utilities Terminal). On the command line, enter the following, replacing `netid` with your own NetID:

```
$ ssh netid@hpc.arizona.edu
```

Windows users will require an ssh client such as PuTTY. Once the software is installed, 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:

```
Login as: netid
```

Once you establish a connection from any OS, you will be prompted for your UArizona password. Note: when you enter your password on the command line, no characters will appear on your screen. This is perfectly normal and everything is working as expected. Press enter and you will be requested to Duo Authenticate.

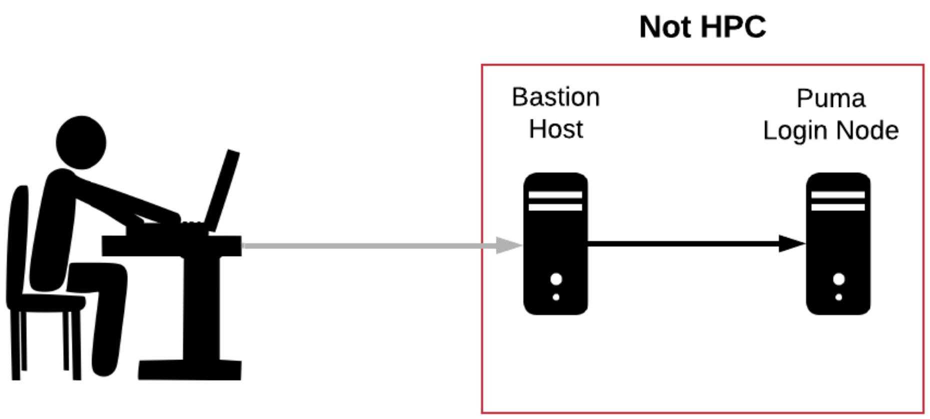
After connecting to the bastion host, you will see:

```
Success. Logging you in...
Last login:
This is a bastion host used to access the rest of the RT/HPC environment.

Type "shell" to access the job submission hosts for all environments
----------------------------------
```

Type **shell** and press enter to connect.

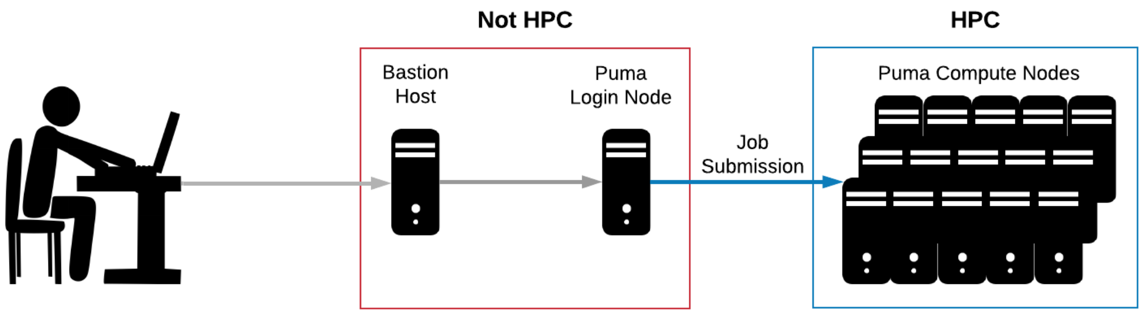
Login Nodes



If you've successfully connected, you're now on a **login node**. A login node serves as a staging area where users may perform housekeeping work, edit scripts, and submit their job requests for execution on one/some of the cluster's compute nodes. We will discuss how to do this in more detail in the sections below.

It is important to know that the login nodes are **not** the location where scripts are run. Heavy computation on the login nodes slows the system down for all users, will not give you the resources or performance you need, and can make it very difficult or impossible for others to use the system. It should also be stressed that **software is not available on the Puma login nodes**. This is a departure from our previous systems and is intended to ensure a fast, high-functioning development environment for all users. We will cover how to access software in the upcoming sections.

How running a Job Works



Users may run their work on HPC by submitting a job request to a **scheduler**. A scheduler, in this case SLURM, is software that will reserve resources and run work on the cluster's compute nodes when space becomes available. To do this, you will need to write a script that requests compute resources and gives the system a blueprint of how to run your work.

In the following sections, we'll learn how to load and use software, write a program and submission script, and run a job. Before we do, let's discuss some key concepts that will come up. These topics are explored in more detail throughout our online documentation, such as in our pages on Allocations and Limits and Running Jobs.

Requesting Resources Overview

Groups and Allocations

When you obtain an HPC account, you become a member of your sponsor's group. You can find all the HPC groups your account is associated with by logging into the HPC portal at portal.hpc.arizona.edu and looking under the Manage Time tab.

Each group has a monthly allocation of 70000 standard CPU hours on Puma and when you run a job the hours used are deducted from your group's account. For example, if you run a job for one hour using 5 CPUs, 5 CPU hours will be charged. To view your group's Puma allocation and usage information, use the command **va**. For example:

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

Windfall: Unlimited

PI: parent_n Total time: 70000:00:00           # Total monthly cpu allocation in hh:mm:ss
        Group: group_name Time used: 32377:07:00   # Group usage for the month
        Total used: 32377:07:00                    # User-specific usage for the month
        Total remaining: 37622:53:00               # Remaining allocation for the month
```

Partitions

The word **standard** in standard hours above refers to the **partitioning** system. A **partition** determines the order in which jobs are started and run. Every group has access to a standard allocation for job submissions. Jobs using standard hours are queued in the standard partition, will not be interrupted once they start running, and will only exit if your work completes (or exits with an error), you manually delete them, or if any requested resources are exceeded.

Windfall is another available partition. Windfall does not consume your standard monthly allocation so you can to continue your work when your hours are used up. Windfall may be used at any time but it should noted that your jobs may be slower to start and can be **interrupted and restarted** by standard jobs. We recommend exhausting your standard allocation before using windfall for better performance.

Some users may see an additional **high_priority** allocation in their account if their group has purchased compute resources. If this is the case, more information can be found on using your buy-in hours in our SLURM documentation.

Time and CPU Time

Time is the amount of time you'd like to reserve for your job. PBS users will know this as Walltime. **CPU Time** is time elapsed multiplied by the number of CPUs you request. When you submit your job, the CPU time requested is immediately deducted from your allocation. Any unused time will be immediately refunded once your job completes.

CPU Time is charged based on the number of CPUs *reserved* and not on the number that are actually used during the job's execution. For example, if you request 5 CPUs but your job is single-threaded and only uses one CPU, you will still be charged for the 5 CPUs.

A CPU is equivalent to a 'core'. Each Puma node has 94 cores/CPUs available for scheduling. A 'node' is a physical computer containing processor sockets with cores, memory, local disk and network adapters.

Accessing Software

Now, let's get down to the nitty-gritty of creating and submitting a job. First, let's talk about what software is available on the system, where it is, and how to use it.

As mentioned a few sections ago, **software is not available on the login nodes**. To see and use software, you must request an **interactive session**. This sends a request to the scheduler to take you from the login node to a compute node. An alias is set up on Puma to do this quickly and easily. Enter **interactive** on the command line to request a single core interactive session for one hour, e.g.:

```
(puma) [netid@junonia ~]$ interactive
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=windfall --partition=windfall
salloc: Granted job allocation 57384
salloc: Waiting for resource configuration
salloc: Nodes r1u03n2 are ready for job
[netid@r1u03n2 ~]$
```

The change in the command line prompt shows you are now on a compute node. From there, you may view and access individual software packages which are made available as **modules**. Modules allow for the customization of your environment, including allowing access to different versions of the same software. Take a look at what’s installed by running the following command:

```
$ module avail
```

You’ll notice there are a *lot* of packages installed. To make things easier, if you are looking for something specific, try adding the name. For example:

```
$ module avail gnu

--------------------- /opt/ohpc/pub/modulefiles ---------------------
   gnu/5.4.0    gnu7/7.3.0    gnu8/8.3.0
```

You'll notice there is more than one gnu module available. To load a specific version into your environment, use **module load <software/version>** If you do not specify the version, you will normally get the latest one. If you prefer a specific version then include the version as in the example below. So think about which method works for your workflow.

In this tutorial, we will be writing and compiling a program written in C. Let’s load the module we need to get started:

```
$ module load gnu/5.4.0
```

You may get an error that gnu8 is already loaded. By default, gnu8, which is the GCC 8.3.0 compiler suite, is already loaded. To use a different version you may see this useful message:

```
$ module swap gnu8 gnu/5.4.0
```

If you ever want to see the modules you currently have loaded, use the command **module list**. To unload a piece of software, use the command **module unload <software>**. For a more comprehensive overview on using system software, take a look at our online documentation on Accessing Software.

Writing a Sample Program for Execution

To run a job on HPC, we’ll first need a script to execute. Let’s write a simple program to get us started. There's more information about jobs and SLURM here. First, make and navigate into a directory:

```
$ mkdir ~/hello_world && cd ~/hello_world
```

Next, make a file by using **touch hello_world.c**. Open the file in your favorite text editor (e.g. use the command **nano hello_world.c**), write the following, then save and exit :

```
#include <unistd.h>
#include<stdio.h>
int main(int argc, char **argv)
{
        char hostname[256];
        gethostname(hostname,255);
        printf("Hello world! I am running on host: %s\n",hostname);
        return 0;
}
```

Now compile using the module we loaded:

```
$ gcc -Wall hello_world.c -o hello_world
```

Writing a SLURM Submission Script

To submit your work for execution with SLURM, you must write a script to tell it how to run your job. These scripts are partitioned into two sections:

Resource Requests

The first portion of your script tells the system the resources you'd like to reserve. This includes the number of **nodes/cores** you need, the **time** it will take to run your job, the **memory** required, your **group's name**, the **partition,** and any special instructions (e.g. requesting **gpus** if applicable). Other optional job specifications may also be set such as a **job name** or requesting **email notifications**. Each line with one of these requests will start with **#SBATCH**. If you'd like to comment out optional specifications that you don't want, change these to **### SBATCH**. You may also delete them.

Job Instructions

The second section tells the system exactly how to do your work. These are all the commands (e.g. loading modules, changing directories, etc) that you would execute **in your current environment** to run your script successfully. This is a departure from the standard behavior of PBS that users may be familiar with. SLURM, by default, inherits the working environment present at the time of job submission. This behavior may be modified with additional SLURM directives.

Writing the Script

This tutorial uses the Standard partition as an example. For the specifics on using different partitions, see: Job Partition Requests.

Let's write a submission script to run the C program we made. We'll create a file called hello_world.slurm by using the command **touch hello_world.slurm**. Open the text file in your favorite editor, write the following, 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.
### SLURM reads %x as the job name and %j as the job ID
#SBATCH --output=%x-%j.out
### REQUIRED. Specify the PI group for this job
#SBATCH --account=<PI GROUP>
### Optional. Request email when job begins and ends
### SBATCH --mail-type=ALL
### Optional. Specify email address to use for notification
### SBATCH --mail-user=<YOUR NETID>@email.arizona.edu
### REQUIRED. Set the partition for your job.
#SBATCH --partition=standard
### REQUIRED. Set the number of cores that will be used for this job.
#SBATCH --ntasks=1
### REQUIRED. Set the number of nodes
#SBATCH --nodes=1
### REQUIRED. Set the memory required for this job.
#SBATCH --mem=1gb
### 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 gnu/5.4.0
### change to your script's directory
cd ~/hello_world
### Run your work
./hello_world
sleep 10
```

The gcc compiler is loaded by default, and is version 8.3.0. In the modules it is referred to as gnu8.

Run Your 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 **sa cct**. 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):

```
[netid@r1u03n2 hello_world]$ sbatch hello_world.slurm
Submitted batch job 58240
[netid@r1u03n2 hello_world]$ squeue --job 58240
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
             58240  standard hello_wo    netid  PD       0:09      1 r1u04n1
```

You can see its state is PD (for pending) which means it's waiting to be executed. Its state will go to R when it's running and if the job has completed running, qstat will return:

```
[netid@r1u03n2 hello_world]$ squeue --job 58240
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

After your job completes, if you included the --output and --job-name options, you'll find the output file in the directory where you submitted your job with the name **<job_name>-<job_id>.out**. If you did not assign an output filename, you'll find it under **slurm-<job_id>.o**.

Users familiar with PBS may expect two output files, one for stdout and the other for stderr. SLURM, by default, only outputs one file that combines the two. Users may request separate output files with optional arguments. Additionally, naming your job will not automatically rename your output file. You must include the --output option to change the default naming scheme. It should be noted that if you do not include the job ID in the output name (you can do this with %j, as shown in the script example), SLURM with overwrite your output files each time the job is run.

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

```
[netid@r1u03n2 hello_world]$ cat hello_world-58240.out
Hello world! I am running on host: r1u04n1.puma.hpc.arizona.edu
```

Additional Information

- Running Jobs with SLURM