If you're joining HPC, it's likely because you want to run your analyses on a larger scale than is currently possible with your available resources. However, if you haven't used an HPC system before, you may not know where or how to get started. This page is designed to give new users a quick overview of how to run their work as well as detail some key concepts that you can come back to for reference. 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 dos and don'ts.
Logging In

To start 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 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:

```
This is a bastion host used to access the rest of the RT/HPC environment.

Shortcut commands to access each resource
-----------------------------------------
Ocelote:
$ ocelote

El Gato:
$ elgato

Puma:
$ puma
```

These are the shortcut commands to access each cluster. Type **ocelote** 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 should be run. Running jobs 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.

If you ever want to check whether you're on a login node, try using the command `echo $HOSTNAME`. If you're on a login node, the word "login" will be present, for example:

```
[netid@login2 ~]$ echo $HOSTNAME
login2
```

How Running a Job Works
Users may run a job on HPC by submitting a job request to a **scheduler**. A scheduler, such as **PBS** (installed on Ocelote and Elgato) or **SLURM** (installed on Puma), is software that will reserve resources and run your work on the cluster's compute nodes when space becomes available. To do this, you will need to write a script that requests resources and gives the system a blueprint for 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 join HPC, you become a member of your sponsor's group. Each group has a monthly allocation of standard CPU hours 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 28 CPUs, 28 CPU hours will be charged. To view your group's name and allocation, use:

```
$ va
```

#### Queues

The word **standard** in standard hours above refers to the queueing system. A **queue** determines the order in which jobs are started and run. Every group has access to a standard allocation for job submissions. These jobs 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 queue. Windfall does not consume your monthly allocation so you can continue your work when your hours are used up. Windfall may be used at any time but it should be noted that your job 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 additional allocations in their account. These are for groups that have either submitted special project requests (qualified), or if they have purchased compute resources (high_priority).

#### Walltime and CPU Time

**Walltime** is the amount of time you'd like to reserve for your job. **CPU Time** is time elapsed multiplied by the number of CPUs you request. When you submit your work, the requested CPU hours are immediately deducted from your group's account. When your job terminates, any unused CPU hours are immediately refunded.

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 be charged based on 5 CPUs.
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.

Individual software packages are also available for users as modules. This allows 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 gcc
```

------------- /cm/local/modulefiles -------------
gcc/5.2.0  gcc/5.5.0  gcc/6.1.0(default)  gcc/7.2.0

You'll notice there is more than one gcc available, including one with (default) next to it. To load a specific version into your environment, use `module load <software/version>`. Without the version specified, the module labeled (default) would be selected. We recommend always including the version to ensure you know which one you're using and to maintain consistency in the event of module default updates.

For example, 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 gcc/6.1.0
```

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. 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`) and write the following, then save and exit:

```c
#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 Submission Script
Ocelote uses the scheduling software PBS for job submissions and requires 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 queue, 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 `#PBS`. If you’d like to comment out optional specifications that you don’t want, change these to `### PBS`. You may also delete them.

Note: different clusters have different resources available. For example, Ocelote’s standard nodes have 28 cores each with a ratio of 6gb/core. Elgato’s standard nodes have 16 cores each with a ratio of 4gb/core. For more information, see our page on Compute Resources and Running Jobs with PBS.

**Job Instructions**

The second section tells the system exactly how to do your work. These are all the commands that you would execute immediately after logging your script successfully. This includes loading the necessary modules, changing to the correct directory, and any other relevant instructions.

**Writing the Script**

Let's write a submission script to run the C program we made. We'll create a file called hello_world.pbs by using the command `touch hello_world.pbs`. Open the text file in your favorite editor, write the following, save, and exit:

```bash
#!/bin/bash

# --------------------------------------------------------------
### PART 1: Requests resources to run your job.
# --------------------------------------------------------------
### Optional. Set the job name
#PBS -N hello_world
### REQUIRED. Specify the PI group for this job
#PBS -W group_list=<PI GROUP>
### Optional. Request email when job begins and ends
### PBS -m bea
### Optional. Specify email address to use for notification
### PBS -M <YOUR NETID>@email.arizona.edu
### REQUIRED. Set the queue for your job.
#PBS -q windfall
### REQUIRED. Set the number of nodes, cores and memory that will be used for this job
### pcmem is optional as it defaults to 6gb per core. Only required for high memory = 42gb. Note: mem=ncpus x pcmem
#PBS -l select=1:ncpus=1:mem=6gb:pcmem=6gb
### REQUIRED. Specify "wallclock time" required for this job, hhh:mm:ss
#PBS -l walltime=0:1:0
### Optional. cput = time x ncpus. If this option is not included, it will default to cput = walltime x ncpus.
#PBS -l cput=0:1:0

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

**Run Your Job**

The next step is to submit your job request to the scheduler. To do this, you’ll use the command `qsub`. 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 `qstat`, cancel your job with `qdel`, and get your job’s history with `tracejob`. 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@login2 hello_world]$ qsub hello_world.pbs
3270604.head1.cm.cluster
[netid@login2 hello_world]$ qstat 3270604
Job id       Name             User              Time Use S Queue
------------- --------------- ---------------  ----------- - -----  
3270604.head1 hello_world netid                    0 Q oc_windfall
```

You can see its state is Q 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@login2 ~]$ qstat 3270604
qstat: Unknown Job Id 3270604.head1.cm.cluster
```

After your job completes, you'll find the output files in the directory where you submitted your job. They will be of the form `<job_name>.o<job_id>` and `<job_name>.e<job_id>` for the output and error files, respectively. If you did not assign a job name, you'll find them under your submission script's name with `.o<job_id>` and `.e<job_id>` appended.

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

```
[netid@login2 hello_world]$ cat hello_world.o3270604
Hello world! I am running on host: i0n9
```

Notice the hostname doesn't have the word login in it. Instead, this is the name of the compute node where your job was executed.

Congratulations, you've just run your first job on HPC!