Running Batch Jobs on UIUC's Campus Cluster

Access (Via Unix/Linux Shell)

(In Windows, use PuTTY or PowerShell SSH-Sessions.)

```
ssh -Y -l user cc-login.campuscluster.illinois.edu          # log in to head nodes
scp ./local user@ccftp1.campuscluster.illinois.edu:~/remote # copy data to CC
scp user@ccftp1.campuscluster.illinois.edu:~/remote ./local # copy data from CC
```

Module Commands

`module`	command interface	`load module`	load *module* (set $PATH etc.)
`avail`	list available modules	`unload module`	unload *module*
`list`	list loaded modules	`swap mod1 mod2`	swap two modules
`show module`	show commands w/i *module*	`purge`	unload all modules

Batch Commands (Moab/Torque)

`qstat`	query status of batch jobs	`qdel jobid`	delete job
`-u $USER`	limit to all jobs of $USER	`qpeek jobid`	view stdout and stderr of job
`-n jobid`	list nodes alloc. to job	`qhold jobid`	delay a job in queue
`-f jobid`	show details of job	`qrls jobid`	release a job to queue
`-t jobid[]`	give status of all jobs in array	`showstart jobid`	show current estimated start time of job
`js jobid`	shows batch job script		

Job Submission Flags (qsub)

BASIC JOB CONTROL			
`-l walltime=hh:mm:ss`	specify desired job run time	`-q queue`	specify *queue*
`-l nodes=n:ppn=12`	request *n* nodes and 12 processors per node	`-N jobname`	name job
`-d dir`	open job shell in directory *dir*	`-I`	interactive shell
OUTPUT & NOTIFICATION			
`-j oe` `-o a.log`	pipe `stdout` and `stderr` to *a.log*	`-m abe` `-M a@b.com`	send mail at beginning, end, and on job abort to *a@b.com*
ADVANCED JOB CONTROL			
`-t a-b`	specify job array	`-l nodes=n:ppn=12:m96G`	request memory (not CSE queue)
`-W depend=?`	introduce dependency (run before or after another job)	`-v`	export environment variables to script
		`-x`	enable X-forwarding

EFFECTIVE QUEUE UTILIZATION

Requested times should be close to expected run times for scheduling efficiency. There is also a limit to how many simultaneous jobs you can run—the others will sit in the queue until you have more available slots.

With fairshare scheduling, one can adopt different strategies to run jobs quickly on Campus Cluster:

- Submit smaller jobs which will backfill between larger jobs as nodes become available.
- Submit either job arrays or fewer larger jobs with independent processes within them (using `&`).
- Utilize the `test` and `debug` queues when appropriate.
- Jobs shorter than four hours will currently run on either Taub or Golub as resources become available.

Responsible queue usage involves requesting only the nodes you need and using them responsibly.

- Monitor your jobs to make sure that they don't hang (crash without terminating the process) (`qpeek`).
- Submit jobs as close to the necessary size as possible—this makes the queueing algorithm more efficient.
- Only use interactive jobs to edit, compile, and build your programs as necessary—don't let them sit idle.

Moab **Environment Variables**

`$PBS_JOBID`	Batch queue job ID number
`$PBS_NODEFILE`	List of nodes available for batch job
`$PBS_O_WORKDIR`	Directory from which batch job was submitted

Executing an **MPI** Program (Internode, Distributed Memory)

The number of processes available to MPI generally equals the requested number of nodes × cores per node.

```
module load mvapich2
mpiexec /path/to/application ./inputfile &> ./outputfile.$PBS_JOBID
```

Executing an **OpenMP** Program (Intranode, Shared Memory)

```
export OMP_NUM_THREADS=12
./hello_world_openmp
```

Example **PBS Script**

```
#!/bin/csh
#PBS -l walltime=00:05:00
#PBS -l nodes=2:ppn=12
#PBS -q test
#PBS -N hello_world_mpi
#PBS -j oe
#PBS -o hello_world_mpi.log.$PBS_JOBID
#PBS -m abe
#PBS -M $USER@illinois.edu

setenv job "hello_world_mpi"
setenv RUN_DIR $HOME/src/mpi-mwe/cpp/
module load mvapich2/1.6-intel

# Document settings for debugging purposes (also can use ldd, echo $PATH).
set echo ; module list ; which mpiexec
mpiexec $RUN_DIR/$job
```

Automating **Access** (`alias`, `ssh`)

It can become tedious to type in the domain every time you wish to connect. Add this to your `~/.bash_profile`:

```
alias taub='ssh -Y -l username cc-login.campuscluster.illinois.edu'
```

You can also set up your machine to authenticate with a private key rather than inputing a password every time.

```
cd ; mkdir .ssh ; ssh-keygen -t rsa       # use any passphrase (but don't leave it empty)
chmod +600 .ssh/id_rsa                    # make your private key only readable by you
scp .ssh/id_rsa.pub <username>@<remote>: # include the ':'
ssh -l username cc-login.campuscluster.illinois.edu
mkdir .ssh                                # this is on the remote machine
cat id_rsa.pub >> .ssh/authorized_keys    # append ('>>') rather than replace
chmod 700 .ssh                            # make your public key only accessible by you
```

Now log out and test this by `ssh`ing back in again. Both of these need be done only once (on each machine).