This is a basic introduction to using clusters, and the BIAC cluster more specifically. For more in depth study of basic cluster principles: <http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html>

For references as to using the biac cluster: <https://wiki.biac.duke.edu/biac:cluster>

(The biac wiki has some out of date elements in its otherwise useful [Advanced Bash/Python Submission Scripts](https://wiki.biac.duke.edu/huettel:using_the_biac_cluster) section which I will strive to correct)

**Connecting and interacting with the cluster:**

Going back and forth between the cluster and local installation is a simple matter as long as one knows the address.

To connect to the BIAC cluster, simply type from the terminal  
  
ssh -Y [{identifier}@cluster.biac.duke.edu](mailto:%7bidentifier%7d@cluster.biac.duke.edu)   
qinteract  
  
with {identifier} one’s specific duke ID. The qinteract is what allows the terminal environment to align with the cluster environment (and therefore have access to any shortcut, command, environment variables such as those defined in .bash)

For transfers, the command scp can be used much like the standard copy cp command to send local files to the remote server:

scp /Users/{user}/data/\* {identifier}@cluster.biac.duke.edu:~/copied\_data/

or vice-versa:

scp {identifier}@cluster.biac.duke.edu:~/data/\* /Users/{user}/data/\*

(the ~/ is a shortcut reference to the local directory in any given environment, which in BIAC is usually /home/{identifier}/linux)

**Launching a job script**

One of the main powers of using a cluster is the ability to send ‘jobs’ to the large array of computers used by BIAC without using the user’s computer processes.

The most important command is qsub, which consists ‘submit job’ command itself. The main use of qsub is to submit **bash** scripts to the cluster. If one has scripts in another language, it is advised to create a bash script that launches the other script from there, though python scripts and others can be launched directly from qsub with some tweaking.

qsub has a wide variety of options and I advise advanced users to browse through them, but here are some of the most important ones:

-o {filepath} : the output stream of the job will be created at specified filepath

-e {filepath} : any error output will be created at specified filepath (does not work well with non bash submissions)

-M {email address} : basic status updates will be sent to specified email address

-m {options} : specifies when to send emails. Main examples are  
b (beginning, when job is submitted) e (end of the job) and a (aborted)

-V : exports all local environment variables to the bash submission

-N : specifies a name for the job (very important for long running jobs)

-wd : use local directory

-b : set as binary. This option is what allows the submission of non bash scripts

Finally, once all options are specified, the last command is the bash/other script itself that is submitted.

One example of bash script submission:

qsub -m bea -e ~/wuconnectomes/errorlog.e -o ~/wuconnectomes/outputlog.o -M first.last@duke.edu bashscript.sh

One example of python submission:

qsub -m bea -e ~/wuconnectomes/errorlog.e -o ~/wuconnectomes/outputlog.o -M jacques.stout@duke.edu -b y python3 ~/wuconnectomes/pythonscript.py

**Interacting with jobs**

Even when a job is submitted, it is still possible to interact with it. All launched jobs are assigned a specified identifier. At any time with a terminal connected to the cluster, the command qstat will output the status of the different jobs currently working:

qstat

job-ID prior name user state submit/start at queue slots ja-task-ID

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

1623734 0.75000 python3 jas297 r 01/22/2020 17:38:39 long.q@node19.dhe.duke.edu 1

1674506 0.36558 python3 jas297 r 02/07/2020 15:35:34 users.q@node25.dhe.duke.edu 1

1693677 0.29835 interact jas297 r 02/10/2020 10:23:49 interact.q@blade08.dhe.duke.ed 1

1709633 0.27252 interact jas297 r 02/11/2020 12:03:39 interact.q@blade17.dhe.duke.ed 1

1709659 0.27132 interact jas297 r 02/11/2020 13:14:48 interact.q@blade09.dhe.duke.ed 1

1710513 0.26857 interact jas297 r 02/11/2020 15:58:50 interact.q@blade17.dhe.duke.ed 1

-job-ID: automatically generated unique ID

-prior: short for priority, priority can be changed in qsub submission for jobs that demand urgency

-name: by default, will be named after bash job script or specified interpreter. -N modifies it.

-user: self-explanatory

-state: specifies the current state of any given job. The most frequent ones are

qw (job is being sent to cluster) and

r (job is being run)

-queue: specifies which computer node the process is running on

slots: number of slots dedicated to job

The job ID is particularly important because of the ‘ qdel’ command.

If at any point one wants to cancel a particular job script, one needs only do

qdel {job-ID}

This will count as an abort which will send a mail if -m a was specified during job submission

**Additionnal advice**

Scripts that are running a job cannot be debugged and finding particular problems is much more difficult than standard run. Referring to the output and error files can be useful, but for long commands one can also ask for emails to be sent at any given time in the script for long runs.

For python, I recommend using this function:

def send\_mail(msg\_txt,subject="Cluster message"):  
 #Send mail with specified txt (and subject) to default address specified in global variable  
 msg\_content = io.StringIO()  
 msg\_content.write("Datetime : %s \n\n" % datetime.datetime.now())  
 msg\_content.write("JobID : %d \n\n" % os.getpid() )  
 msg\_content.write("Message : %s \n\n" % msg\_txt)  
 msg = MIMEText(msg\_content.getvalue())  
 msg\_content.close()  
 to\_addr = "%s@duke.edu" % mylogin  
 from\_addr = "%s@duke.edu" % mylogin  
 msg['Subject']=subject  
 msg['from']=from\_addr  
 msg['to']=to\_addr  
 s = smtplib.SMTP('smtpgw.duhs.duke.edu')  
 s.sendmail(from\_addr, [to\_addr], msg.as\_string())  
 s.quit()

with mylogin either added inside the function or set as a global variable. Once this is done, simply call the function at any time with send\_mail(‘my main message’,’my email subject’)