Introduction to High performance computing using the CHPC

Werner Smidt

Outline

- Introduction to modern computing
- Command-line interface explained
- Running BLAST
- Preparing scripts for the CHPC
- Other usage examples

Modern computing

- Computing power has expanded over the last few decades
- Possible to run large bioinformatic applications
- Availability of applications
  - Proprietary
  - Open source
- Local and remote applications
  - BLAST example of both
- GUI interfaces
  - CLC
  - NCBI tool/EBI etc

Modern Computing

- Scalability
- Common for PCs to have multiple CPUs
- Faster completion of tasks
- Larger storage
- More RAM for heavy-duty tasks
- Local PCs -> Servers -> HPC

Limitations of local PCs

- Modern personal computers are powerful, but …
  - Still struggle with certain tasks
- For instance:
  - Genome assemblers - Velvet for a ~300MB genome uses ~48GB RAM
  - Read mapping possible, but takes a long time
  - Large phylogenies

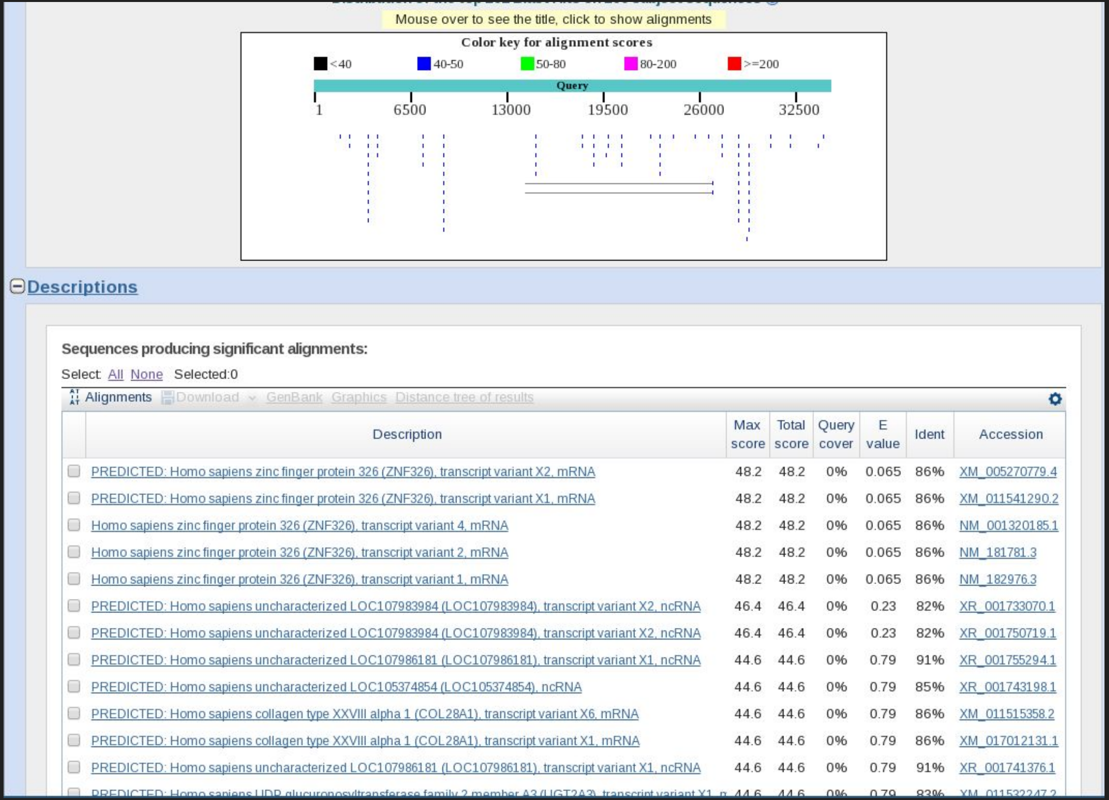
How did I get into this (nut)shell?

- Any sort of interface to the underlying operating system is a “shell”
  - Graphical
  - Text based
- Here, “shell” will exclusively pertain to the textual component
- Many BIF programs have textual interfaces:
  - Genome assemblers
  - Read mappers
  - Phylogenetic software
  - etc.

But … why?

- Hard to make a graphical interface suitable for all the tools you may need
- A shell, in the textual form, enforces a standard
- A form of “cognitive ergonomics” or … whatever
- Allows easy creation of ‘scripts’ (think experimental protocol):
  - Sequential command execution
  - Align -> Trim -> Motif discovery
  - Quality filtering/trimming -> Assembly phase 1 -> Assembly phase 2 -> Annotation
- Personally I like to stay away from the “P” word …
- Easier/more reliable to connect to a remote textual shell

An example



Sequences producing significant alignments:

Description	Max score	Total score	Query cover	E value	Ident	Accession
PREDICTED: Homo sapiens zinc finger protein 326 (ZNF326), transcript variant X2, mRNA	48.2	48.2	0%	0.065	86%	XM_005270779.4
PREDICTED: Homo sapiens zinc finger protein 326 (ZNF326), transcript variant X1, mRNA	48.2	48.2	0%	0.065	86%	XM_011541290.2
Homo sapiens zinc finger protein 326 (ZNF326), transcript variant 4, mRNA	48.2	48.2	0%	0.065	86%	NM_001320185.1
Homo sapiens zinc finger protein 326 (ZNF326), transcript variant 2, mRNA	48.2	48.2	0%	0.065	86%	NM_181781.3
Homo sapiens zinc finger protein 326 (ZNF326), transcript variant 1, mRNA	48.2	48.2	0%	0.065	86%	NM_182976.3
PREDICTED: Homo sapiens uncharacterized LOC107983984 (LOC107983984), transcript variant X2, ncRNA	46.4	46.4	0%	0.23	82%	XR_001733070.1
PREDICTED: Homo sapiens uncharacterized LOC107983984 (LOC107983984), transcript variant X2, ncRNA	46.4	46.4	0%	0.23	82%	XR_001750719.1
PREDICTED: Homo sapiens uncharacterized LOC107986181 (LOC107986181), transcript variant X1, ncRNA	44.6	44.6	0%	0.79	91%	XR_001755294.1
PREDICTED: Homo sapiens uncharacterized LOC105374854 (LOC105374854), ncRNA	44.6	44.6	0%	0.79	85%	XR_001743198.1
PREDICTED: Homo sapiens collagen type XXVIII alpha 1 (COL28A1), transcript variant X6, mRNA	44.6	44.6	0%	0.79	86%	XM_011515358.2
PREDICTED: Homo sapiens collagen type XXVIII alpha 1 (COL28A1), transcript variant X1, mRNA	44.6	44.6	0%	0.79	86%	XM_017012131.1
PREDICTED: Homo sapiens uncharacterized LOC107986181 (LOC107986181), transcript variant X1, ncRNA	44.6	44.6	0%	0.79	91%	XR_001741376.1
PREDICTED: Homo sapiens UDP glucuronosyltransferase family 2 member A3 (UGT2A3), transcript variant X1, n	44.6	44.6	0%	0.79	83%	XM_011532247.2

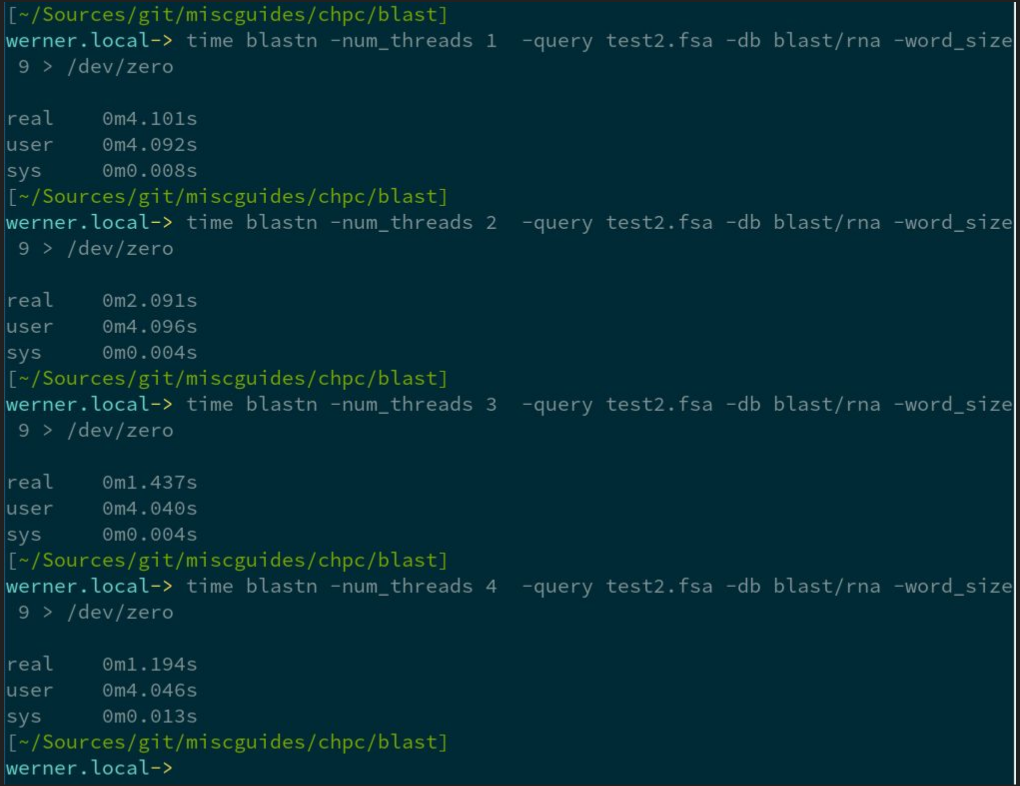
Command line BLAST

Command line BLAST (via SSH)

- Secure SHell is a tool to connect to a remote server
- This happens within the same terminal
- But logging in to another server means:
    - All the commands are executed on that remote server
    - You have access to software on that server
    - When accessing storage, you are accessing the *remote* storage (from within SSH)
    - But you can upload your data to the relevant server

Command line BLAST via SSH

Threads example



```
[~/Sources/git/miscguides/chpc/blast]
werner.local-> time blastn -num_threads 1  -query test2.fsa -db blast/rna -word_size
 9 > /dev/zero

real	0m4.101s
user	0m4.092s
sys	0m0.008s
[~/Sources/git/miscguides/chpc/blast]
werner.local-> time blastn -num_threads 2  -query test2.fsa -db blast/rna -word_size
 9 > /dev/zero

real	0m2.091s
user	0m4.096s
sys	0m0.004s
[~/Sources/git/miscguides/chpc/blast]
werner.local-> time blastn -num_threads 3  -query test2.fsa -db blast/rna -word_size
 9 > /dev/zero

real	0m1.437s
user	0m4.040s
sys	0m0.004s
[~/Sources/git/miscguides/chpc/blast]
werner.local-> time blastn -num_threads 4  -query test2.fsa -db blast/rna -word_size
 9 > /dev/zero

real	0m1.194s
user	0m4.046s
sys	0m0.013s
[~/Sources/git/miscguides/chpc/blast]
werner.local->
```

The Centre for High Performance Computing

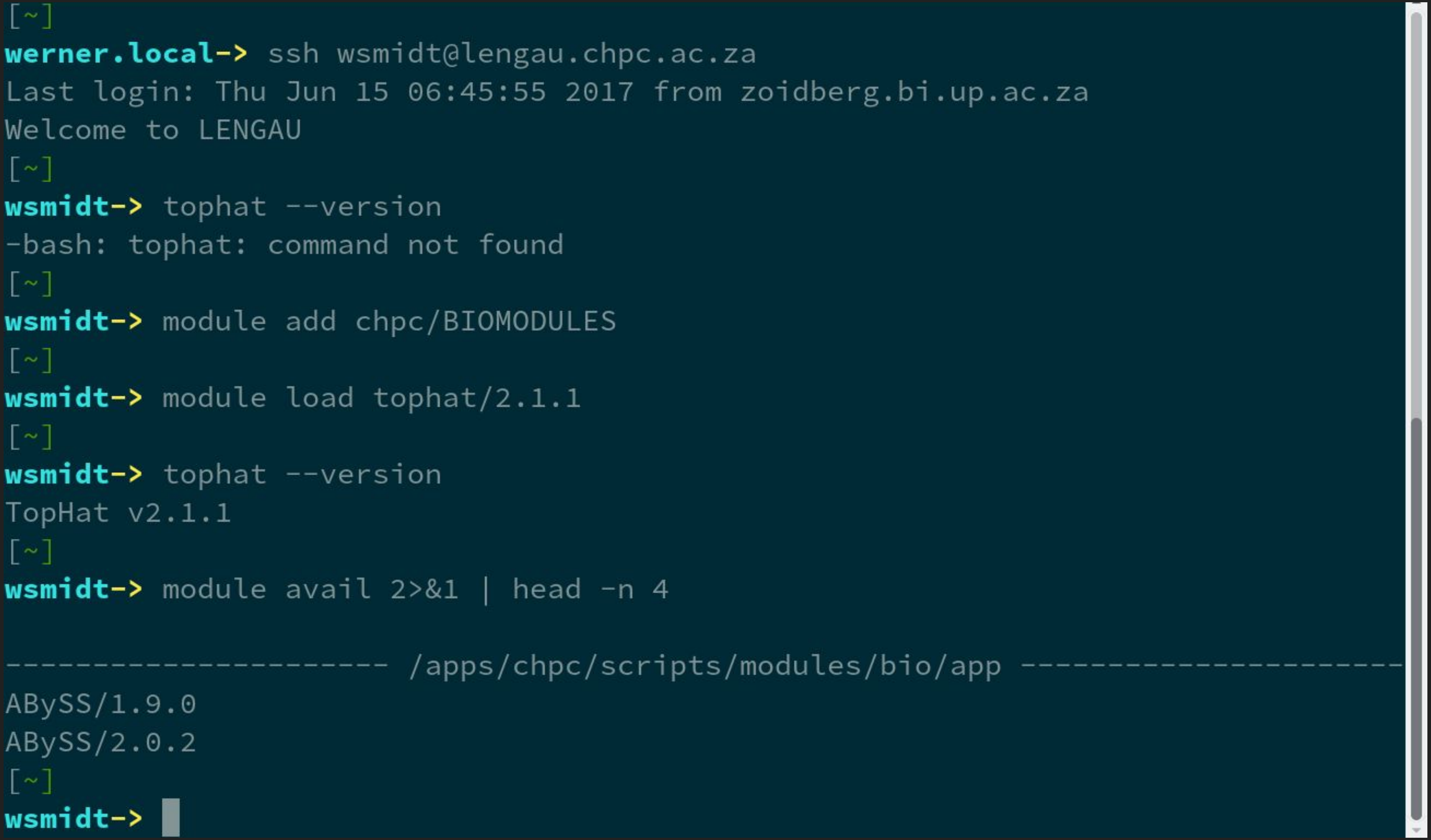
- It’s massive
- There are 1368 compute nodes
  - Equates to 32,832 cores (workers)
- Massive storage capacity
  - 4 PB = 4,000 TB = 4,000,000 GB
- Each node typically has 128GB RAM and 24 cores

The Centre for High Performance Computing

- You can imagine that managing such a resource is a mammoth task
- Also, while there are thousands of nodes . . .
    - Ridiculous to log into each node to see which one is free
    - Better to have a queuing manager that can handle jobs you would like to run
- Because of this:
    - Commands should not be run directly
    - Instead, they need to be “wrapped” in a script that can:
        - Tell the queng manager what is needed i.t.o resources
        - Who is running the job (group)
- It also safeguards against “hogging” the resource
    - Lower priority to too-frequent users ensuring a fair usage policy

The Centre for High Performance Computing

- Most BIF application you would like to run are not available implicitly
- The CHPC uses the GNU Modules suite to handle this
- Allows simultaneous existence of incompatible software/different versions
- Makes the task of adding new applications easier



```
[~]
werner.local-> ssh wsmidt@lengau.chpc.ac.za
Last login: Thu Jun 15 06:45:55 2017 from zoidberg.bi.up.ac.za
Welcome to LENGAU
[~]
wsmidt-> tophat --version
-bash: tophat: command not found
[~]
wsmidt-> module add chpc/BIOMODULES
[~]
wsmidt-> module load tophat/2.1.1
[~]
wsmidt-> tophat --version
TopHat v2.1.1
[~]
wsmidt-> module avail 2>&1 | head -n 4

---------------------- /apps/chpc/scripts/modules/bio/app ----------------------
ABySS/1.9.0
ABySS/2.0.2
[~]
wsmidt->
```

The BLAST example (again)

- So now we’ll do something a little more fancy
- We’ll run BLASTX against the refseq_protein database
- Knowing it would be big task
  - Does translation for the entire sequence in different reading frames and on both strands
  - Tries to find matching proteins
- Perfect opportunity to try out the CHPC
- As stated before, we need to wrap this command in a script (PBS script)
- Before we get too excited, let’s examine the final PBS script we’ll use

BLASTX PBS script

- Let’s look at a typical (and trivial) PBS script

```
#!/bin/bash
#PBS -l select=1:ncpus=24:nodetype=haswell_reg
#PBS -P CBBI0905
#PBS -q smp
#PBS -l walltime=48:00:00
#PBS -o /home/wsmidt/outputs/myblast.out
#PBS -e /home/wsmidt/outputs/myblast.err

#PBS -N mytestblast
#PBS -M werner.smidt@gmail.com

cd $PBS_O_WORKDIR
export BLASTDB="/mnt/lustre/bsp/NCBI/BLAST"

module add chpc/BIOMODULES
module load ncbi-blast-2.6.0

blastx -query test.fsa -db refseq_protein -num_threads 24 #The actual command
```

BLASTX PBS script

- Yikes!
- OK, so this is the command we would like to run

```
blastx -query test.fsa -db refseq_protein -num_threads 24
```

- To do this, we need to set up a few things first
  - Load the appropriate modules
  - Do additional setup procedures in order for the BLAST to run
  - Specify where screen output should go
  - Tell the queue manager the resources we require

Let’s work a little in reverse

- Let’s look at the lines preceding the blastx command we would like to run

```
module load chpc/BIOMODULES
module load ncbi-blast-2.6.0
```

- The command “module” is used to set up our shell to execute the applications we want
- Modules are divided into categories
  - BIF tools are under the chpc/BIOMODULES “category”
  - Now the BLAST suite of utilities can be loaded
-

What do I need?

- We have everything in place to run the command
- Still need to set up the script so it can be added (submitted) to the queue

```
#!/bin/bash
#PBS -l select=1:ncpus=24:nodetype=haswell_reg
#PBS -P CBBI0905
```

- #!/bin/bash : We would like to use the “BASH” shell
- PBS -l: sets multiple options
  - Select 1 node
  - Gimme ncpus=24 cpus on the new shiny haswell_reg nodes
- PBS -P: I’ll be submitting this under project CBBI0905

...continued

```
#PBS -q smp
#PBS -l walltime=48:00:00
```

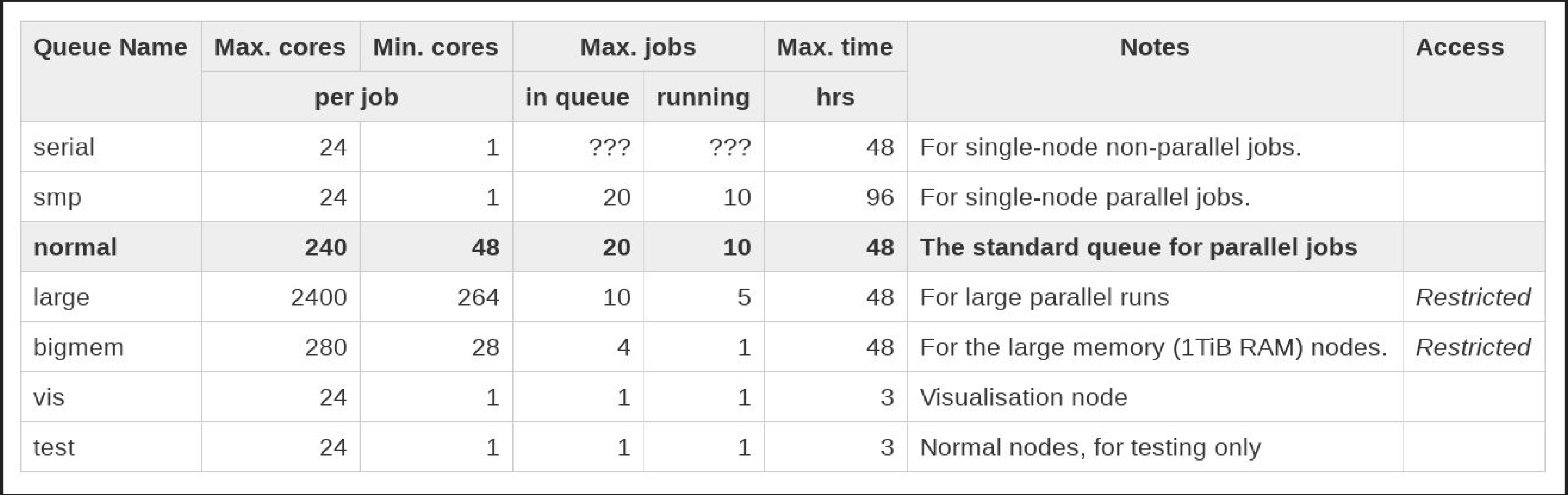
- We would like to use the ‘smp’ queue.
- We expect the job to run no longer than 48 hours
  - You cannot expect things to run forever
  - There is a 96 hour limit on this queue
  - You can make a guess or max it out if you are unsure

Where will the output go?

```
#PBS -o /home/wsmidt/outputs/myblast.out
#PBS -e /home/wsmidt/outputs/myblast.err
```

- BLAST has output options, but I didn’t specify any
- By default, the output will be written to the screen
- Since the job we’re submitting will (eventually) run on a random node
  - How do we get the textual output?
- Two ‘streams of text’:
  - stdout - Normal output of a program
  - stderr - Additional info/error messages
- Can separate the two

Queues



Queue Name	Max. cores	Min. cores	Max. jobs		Max. time	Notes	Access
	per job		in queue	running	hrs		
serial	24	1	???	???	48	For single-node non-parallel jobs.	
smp	24	1	20	10	96	For single-node parallel jobs.	
normal	**240**	**48**	**20**	**10**	**48**	**The standard queue for parallel jobs**	
large	2400	264	10	5	48	For large parallel runs	*Restricted*
bigmem	280	28	4	1	48	For the large memory (1TiB RAM) nodes.	*Restricted*
vis	24	1	1	1	3	Visualisation node	
test	24	1	1	1	3	Normal nodes, for testing only	

Naming and notification

```
#PBS -N mytestblast
#PBS -M werner.smidt@gmail.com
#OPTIONAL: PBS -m abe
```

- We give the job a name that we can identify
- Ask any notification to be sent to my email address
  - By default, it will only send you an email when the job fails
- I didn’t include the “optional” line:
  - Mails can be sent at the (b)eginning, (e)nd and when a job is (a)borted

BLASTX PBS script

- Let’s look at a typical (and trivial) PBS script

```
#!/bin/bash
#PBS -l select=1:ncpus=24:nodetype=haswell_reg
#PBS -P CBBI0905
#PBS -q smp
#PBS -l walltime=48:00:00
#PBS -o /home/wsmidt/outputs/myblast.out
#PBS -e /home/wsmidt/outputs/myblast.err

#PBS -N mytestblast
#PBS -M werner.smidt@gmail.com

cd $PBS_O_WORKDIR
export BLASTDB="/mnt/lustre/bsp/NCBI/BLAST"

module add chpc/BIOMODULES
module load ncbi-blast-2.6.0

blastx -query test.fsa -db refseq_protein -num_threads 24 #The actual command
```

Bear with me . . .

- Assume I “somehow” got my script to the CHPC
- Assume I did everything else by the book
- To submit my job to the queue, I need to run the command

```
qsub myblast.pbs
```

Qsub

Output

What’s next?

- This is just a trivial example of CHPC usage
- Things you can imagine doing:
    - Mapping
    - Genome assembly
    - Population genetics
    - Phylogenetics
    - Molecular dynamics
    - Etc
- Practical sessions?
    - Accounts?
    - I’ll restrict my role to technical parts
    - Members of your group can present what they have done?