Introduction to the Unix Shell: http://swc-osg-workshop.github.io/2016-01-06-UNL/novice/shell/index.html

1. http://swc-osg-workshop.github.io/2016-01-06-UNL/novice/shell/00-intro.html
2. http://swc-osg-workshop.github.io/2016-01-06-UNL/novice/shell/01-filedir.html
3. http://swc-osg-workshop.github.io/2016-01-06-UNL/novice/shell/02-create.html

Let's look into our first BLAST job.
$ cat blast_nt_single.submit

Explain:
#SBATCH **--nodes=1**
#SBATCH **--ntasks-per-node=1**
module load **blast/2.2.29**
module load **biodata/1.0**
//basic BLAST command =>
blastn -db /tmp/nt.00 **-query** /tmp/yeast.nt.fasta **-out** /tmp/blast_nt_single.alignments
We are performing "blastn" (nucleotide BLAST), alignment of nucleotide query (yeast.nt.fasta) against nucleotide db (nt.00).
We are using part of the non-redundant nucleotide database (nt.00) so the job can finish in few minutes.
For your research, it is recommended to use the full nucleotide non-redundant database (nt). This run can last hours or days sometimes.

$ sbatch blast_nt_single.submit
$ squeue –u <username>

The job was running for almost 3 minutes.
The output file is named "blast_nt_single.alignments"
$ head -n 30 blast_nt_single.alignments

The previous example was basic BLAST run where we used 1 node and 1 core.
However, BLAST can use multiple cores in one node.
Let's try that as an example, and see whether this will be faster or not.
$ cat blast_nt_multi.submit

Explain:
#SBATCH **--nodes=1**
#SBATCH **--ntasks-per-node=2**
//basic BLAST command =>
blastn -db /tmp/nt.00 **-query** /tmp/yeast.nt.fasta **-out** /tmp/blast_nt_multi.alignments **-num_threads $SLURM_NTASKS_PER_NODE**
The command is same as the example before, just we have the addition of "**-num_threads $SLURM_NTASKS_PER_NODE**".
On Crane, the **--ntasks-per-node** value can be up to 16, while on Tusker, this value can go up to 64 cores per node (https://hcc-docs.unl.edu/x/axMF).

$ sbatch blast_nt_multi.submit
$ squeue –u <username>

The job was running for 2 minutes (1 minutes less than the previous run).
The output file is named "blast_nt_multi.alignments".
$ head -n 30 blast_nt_multi.alignments

Beside “blastn”, you can run the following programs as well after the blast module is loaded:

-blastp – aligns amino acid query sequence against a protein sequence database
-blastx – aligns nucleotide query sequence against a protein sequence database such that the query is translated in all reading frames
-tblastn – alignsss protein query sequence against a nucleotide sequence database such that the query is dynamically translated in all reading frames
-tblastx – aligns six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database

Let’s run “blastp” as an example.
$ cat blast_aa_multi.submit

Explain:
We use “blastp”, 2 cores, much smaller yeast protein database (yeast.aa.*), small yeast nucleotide query (yeast.aa.fasta) =>
blastp -db /tmp/**yeast.aa** -query /tmp/**yeast.aa.fasta** –out /tmp/blast_aa_multi.alignments -num_threads $SLURM_NTASKS_PER_NODE

$ sbatch blast_aa_multi.submit
$ squeue –u <username>

The job was running for few seconds.
The output file is named “blast_aa_multi.alignments”.
$ head -n 40 blast_aa_multi.alignments

Every BLAST program has various options/parameters that can be used for more specific results.
So far, we have used the following options for “blastn” and “blastp”: “-db”, “-query”, “-out”, “-num_threads”.
In order to look for the other options available for “blastn” for example, we can type the following:
$ blastn --h
To print detailed descriptions of command line arguments, we can type:
$ blastn -help

One of frequently used BLAST option is the one for the format of the alignments:

*** Formatting options
-outfmt <String>
alignment view options:
0 = pairwise,
1 = query-anchored showing identities,
2 = query-anchored no identities,
3 = flat query-anchored, show identities,
4 = flat query-anchored, no identities,
5 = XML Blast output,
6 = tabular,
7 = tabular with comment lines,
8 = Text ASN.1,
9 = Binary ASN.1,
10 = Comma-separated values,
11 = BLAST archive format (ASN.1)

The default BLAST output is the pairwise one.
However, some tools accept different BLAST outputs as inputs.
For example, Blast2GO, a platform for functional Gene Ontology annotation and analysis of genomic datasets, can use the XML output of BLAST and perform the annotation.

Let's run "blastp" now using the option for XML output.
Afterwards, we will use this output with Blast2GO.

$ cat blast_aa_b2g.submit

Explain:
blastp -db /tmp/yeast.aa -query /tmp/yeast.aa.b2g.fasta –out /tmp/blast_aa_b2g.xml **-outfmt 5** -num_threads $SLURM_NTASKS_PER_NODE

$ sbatch blast_aa_multi.submit
$ squeue –u <username>

The job was running for few seconds.
The output file is named "blast_aa_b2g.xml".
$ head -n 20 blast_aa_b2g.xml

Let's use this output now to run Blast2GO.
$ cat b2g.submit

Explain:
module load **b2g4pipe** => available only for the glu and demo groups
java es.blast2go.prog.B2GAnnotPipe **-in blast_aa_b2g.xml** -prop /home/demo/shared/programs/b2g4pipe/bin/b2gPipe.properties -out results -annot
java es.blast2go.prog.B2GAnnotPipe => should be always used
-prop /home/demo/shared/programs/b2g4pipe/bin/b2gPipe.properties (change demo to glu for glu accounts)
-annot performs Gene Ontology annotations and these annotations will be stored in the output files "**results.annot**"

Additional options that can be used with Blast2GO can be seen when:
$ java es.blast2go.prog.B2GAnnotPipe

$ sbatch b2g.submit
$ squeue –u <username>

The job was running for few seconds.
The output file is named "results.annot".
$ head -n 30 results.annot

HCC-DOCS References:

Biodata Module: https://hcc-docs.unl.edu/x/L4Hn
Bioinformatics Tools: https://hcc-docs.unl.edu/x/-wR9