IBG Research Computing Cheatsheet

Best practices

Memory and CPU allocation

```
#placeholder
```

Choosing between `preemptable` and `blanca-ibg`

```
#placeholder
```

Monitoring Slurm and Job Activity

Monitoring running jobs

View your running jobs

```
squeue -u $USER
```

- Additional fields displayed using the `-l` flag
- Custom output described in man page. E.g., I find `squeue -o "%.12i %.18j %.16q %.8T %.10M %.12l %.24R"` easier to read

View running jobs on a a particular qos

```
squeue -q <QOS>
```

Measure performance of completed jobs

The following flags can be combined to suit your needs

Display jobs completed since a particular date

```
sacct -S <MMDD>
```

Display jobs' timing and memory usage

```
sacct -o 'jobid%20,jobname%16,state,elapsed,maxrss'
```

View available nodes and their properties

```
 sinfo  --Node -o  "%.12N %.16P %.11T %.4c %.13C %.8e /%.8m %.30f"
```

- `NODELIST` is the name of the node, can be specified when submitting via –nodelist
- `PARTITION` is the name of the partition. You likely will never need to use this unless you want a particular partition of the preemptable queue
- `STATE` - mixed means that some of the CPUS are in use (see below)
- `CPUS(A/I/O/T)` - on a given node: allocated/idle/other/total nodes
- `FREE_MEM / MEMORY` free memory / total memory in MB (divide by 1024 to convert to GB
- `AVAIL_FEATURES` - architectures/instruction sets requestable via the `--constraint` flag

Example jobs

Basics

Interactive jobs

```
#placeholder
```

Preemptable job

```
#placeholder
```

Job arrays

Job arrays provide a clean method for subbmitting collections of jobs and are often preferable to either i. using a loop to submit multiple jobs or ii. using a loop within a single job to accomplish multiple tasks. Note that the resources you request apply to each job individually, not to the collection of jobs. E.g., if each of 10 tasks requires 50gb of memory, you only need to request 50gb, not 500gb.

Job array with a single numeric index

```
#!/bin/bash
#SBATCH --qos=<QOS>
#SBATCH --mem=<MEM>gb
#SBATCH --time=<D-HH:MM> or <HH:MM:SS>
#SBATCH --ntasks=<Number of cpus>
#SBATCH --nodes=1
#SBATCH --array=1-22
#SBATCH -J <jobname>
#SBATCH -o <output dir>/<jobname>_%a

CHR=${SLURM_ARRAY_TASK_ID}


ml load <modules>

program <args> \
    --input <inputprefix>_"$CHR" \
    --output <outputprefix>_"$CHR"
```

Job array with a single non-numeric index

This script submit jobs in parallel for jobs with different inputs

```
#!/bin/bash
...
#SBATCH --array=0-2
#SBATCH -J <jobname>
#SBATCH -o <output dir>/<jobname>_%a

ii=${SLURM_ARRAY_TASK_ID}

inputArray=(phenoA phenoB phenoC)
```

```
input=${inputArray[$ii]}

ml load <modules>

program <args> \
    --input $input
```

Job array with a multiple simultaneous indices

This script submit jobs in parallel for jobs with different inputs/outputs

```
#!/bin/bash
...
#SBATCH --array=0-2
#SBATCH -J <jobname>
#SBATCH -o <output dir>/<jobname>_%a

ii=${SLURM_ARRAY_TASK_ID}

inputArray=(phenoA phenoB phenoC)
outputArray=(outputA outputB outputC)

input=${inputArray[$ii]}
output=${outputArray[$ii]}

ml load <modules>

program <args> \
    --input $input \
    --output $output
```

Job array with a multiple nested non-numeric indices

This script submit jobs in parallel for jobs with arbitrary nested lists of arguments (e.g., each model for each phenotype) using integer arithmetic. If this is unfamiliar, you can google "floor division bash" and "modulo bash".

```
#!/bin/bash
...
#SBATCH --array=0-5
#SBATCH -J <jobname>
#SBATCH -o <output dir>/<jobname>_%a

ii=${SLURM_ARRAY_TASK_ID}

inputArray=(phenoA phenoB phenoC)
modelArray=(modelA modelB)

inputIndex=$(expr $ii % 3)
modelIndex=$(expr $ii / 3)

input=${inputArray[$inputIndex]}
model=${modelArray[$modelIndex]}

ml load <modules>
```

```
program <args> \
    --phenotype "$input" \
    --model $model"
```

You can always double check that you didn't screw something up by running a simple loop (in the shell):

```
inputArray=(modelA modelB)
modelArray=(phenoA phenoB phenoC)

for ii in {0..5}
do
    inputArray=(phenoA phenoB phenoC)
    modelArray=(modelA modelB)
    inputIndex=$(expr $ii % 3)
    modelIndex=$(expr $ii / 3)
    input=${inputArray[$inputIndex]}
    model=${modelArray[$modelIndex]}
    echo input:"$input" model:"$model"
done
```

Job array with a multiple nested indices, one numeric

You can frequently simplify things when one the lists you iterate over is numeric (e.g., each chromsome for each phenotype):

```
#!/bin/bash
...
#SBATCH --array=0-65
#SBATCH -J <jobname>
#SBATCH -o <output dir>/<jobname>_%a

ii=${SLURM_ARRAY_TASK_ID}

inputArray=(phenoA phenoB phenoC)

inputIndex=$(expr $ii / 22)
chrom=$(expr $(expr $ii % 22) + 1)

input=${inputArray[$inputIndex]}

ml load <modules>

program <args> \
    --phenotype "$input" \
    --genotypes <someprefix>_chr"$chrom"
```

Again, you can check that this works via a loop:

```
inputArray=(phenoA phenoB phenoC)

for ii in {0..65}
do
    inputIndex=$(expr $ii / 22)
    chrom=$(expr $(expr $ii % 22) + 1)
    input=${inputArray[$inputIndex]}
    echo chrom:"$chrom" pheno:"$input"
```

```
done
```

Defining custom functions

Commands that are lengthy to type in or that you frequently used can be turned into functions to save time. To do so, add something along the following lines to `~/.my.bashrc`

```
function Sinfo() {
sinfo  --Node -o  "%.12N %.16P %.11T %.4c %.13C %.8e /%.8m %.30f"
}
export -f Sinfo
```

Then call `source ~/.my.bashrc` and you should be able to use `Sinfo` in place of the longer command above.