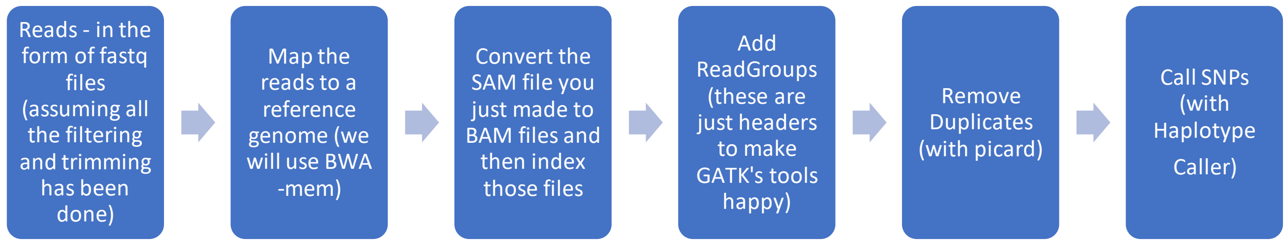
Calling SNPs with BWA and HaplotypeCaller

The general workflow goes as follows:


Reads - in the form of fastq files (assuming all the filtering and trimming has been done)
Map the reads to a reference genome (we will use BWA -mem)
Convert the SAM file you just made to BAM files and then index those files
Add ReadGroups (these are just headers to make GATK's tools happy)
Remove Duplicates (with picard)
Call SNPs (with Haplotype Caller)

This is a guide to help call SNPs in VCF format from reads in fastq files. This assumes the reads are already trimmed. To check your fastq files I would recommend using FastQC which is already loaded on the supercomputer. This will make html files. To understand some of the statistics on it look at this manual. If you don't know how to create and submit jobs, look at the last section of this paper. I also assume you have access to the conda modules

Reference File Indexing

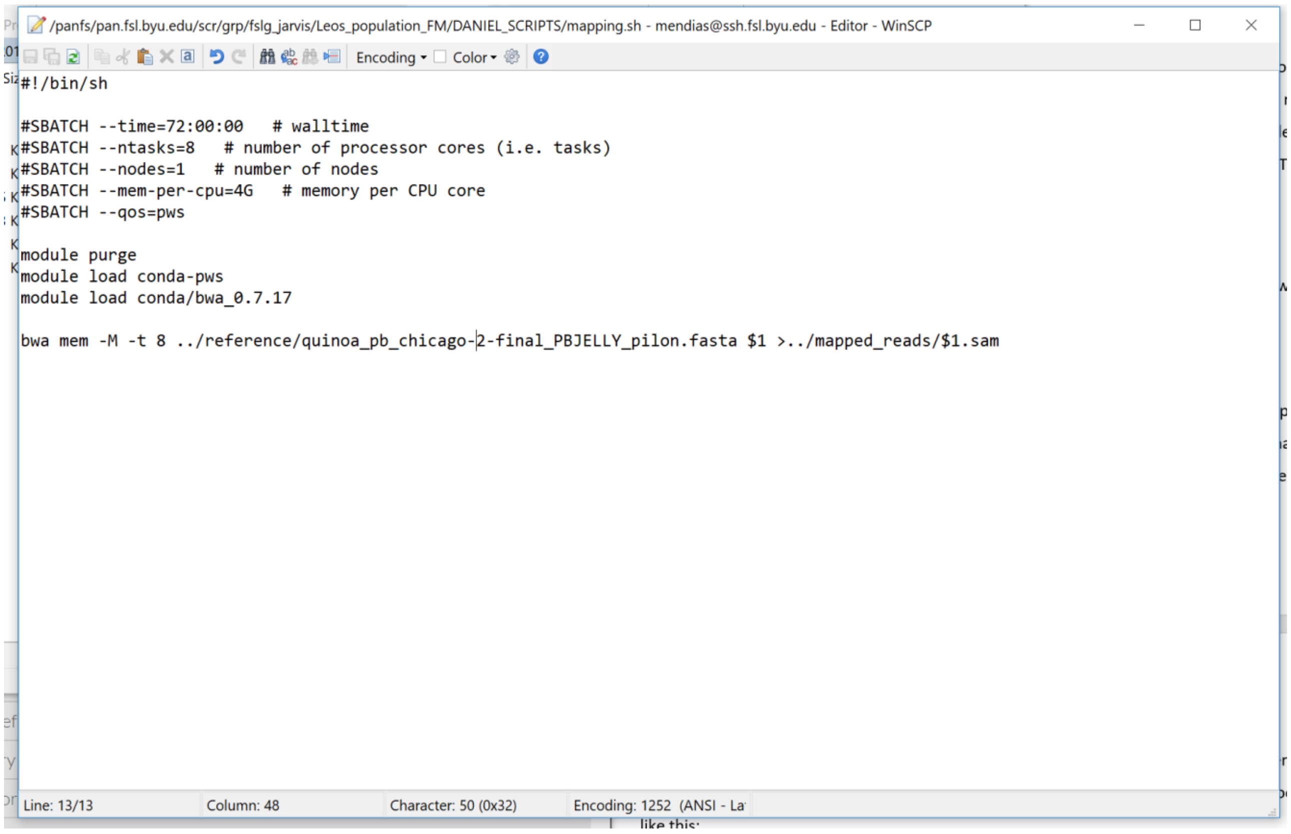
Before starting, you should have your reference genome downloaded as a fasta file and I recommend putting it in a folder called "reference". You will need to index it and make a dictionary file among some other files. I used "bwa index", "samtools faidx", and "gatk CreateSequenceDictionary". This link should help. It does use picard for creating the .dict file, but using GATK like:

```
"gatk CreateSequenceDictionary -R myReference.fasta"
```

works just fine. GATK uses picard. Make sure the fasta file is in the same folder as the new files that are created.

Mapping

Once the reference genome is indexed and everything, the reads can now be mapped. I used the “mapping.sh” script (see picture below) for this which uses bwa.


/panfs/pan.fsl.byu.edu/scr/grp/fslg_jarvis/Leos_population_FM/DANIEL_SCRIPTS/mapping.sh - mendias@ssh.fsl.byu.edu - Editor - WinSCP
Encoding
Color
Line: 13/13
Column: 48
Character: 50 (0x32)
Encoding: 1252 (ANSI - La

```
#!/bin/sh

#SBATCH --time=72:00:00   # walltime
#SBATCH --ntasks=8   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH --mem-per-cpu=4G   # memory per CPU core
#SBATCH --qos=pws

module purge
module load conda-pws
module load conda/bwa_0.7.17

bwa mem -M -t 8 ../reference/quinoa_pb_chicago-2-final_PBJELLY_pilon.fasta $1 >../mapped_reads/$1.sam
```

The last line of the file will have to be changed so that the right reference genome is referred to and the output goes to the correct folder. Make sure the output file name has “.sam” at the end. The “$1” is just whatever argument is put in when you run the script. In this case it will be the file name. the “>” character in the last line means “output to this”. Running this script as a job submission would look something like this:

```
sbatch mapping.sh myFile.fastq
```

“myFile.fastq” is what the “$1” is referring to in this statement and is the file path to file you want to run the script on. If you want to submit a bunch of jobs on different files with this script, consider using a bash for loop. It will look something like this:

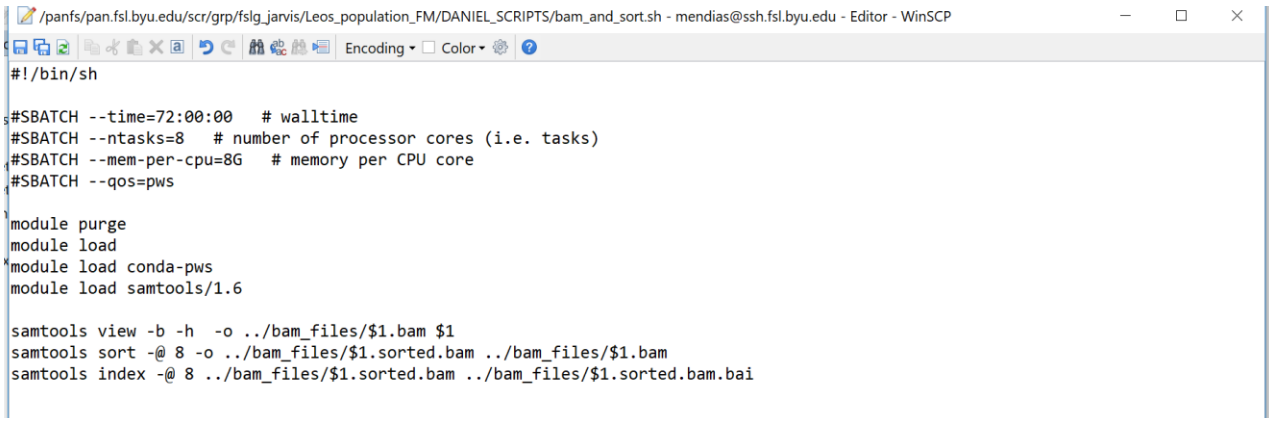
for i in *.fq; do echo “processing $i”; sbatch mapping.sh $i; done

“for i in *.fq” means “for each file with the suffix .fq, assign that file name to variable i”. Then “do echo ‘processing $i’” is optional, it will just output to the screen “processing theFileName”. You do need the “do” keyword though for the next argument and the “do keyword is only required once in the statement. Each of the arguments are separated with a semicolon.

After this is run, you will end up with a SAM file for each fastq file mapped. To keep things organized, I recommend putting each step in its own folder.

SAM to BAM and Indexing

The SAM files are alignment files and need to be converted to BAM files which are just binary versions of SAM files. This can be done with bam_and_sort.sh.

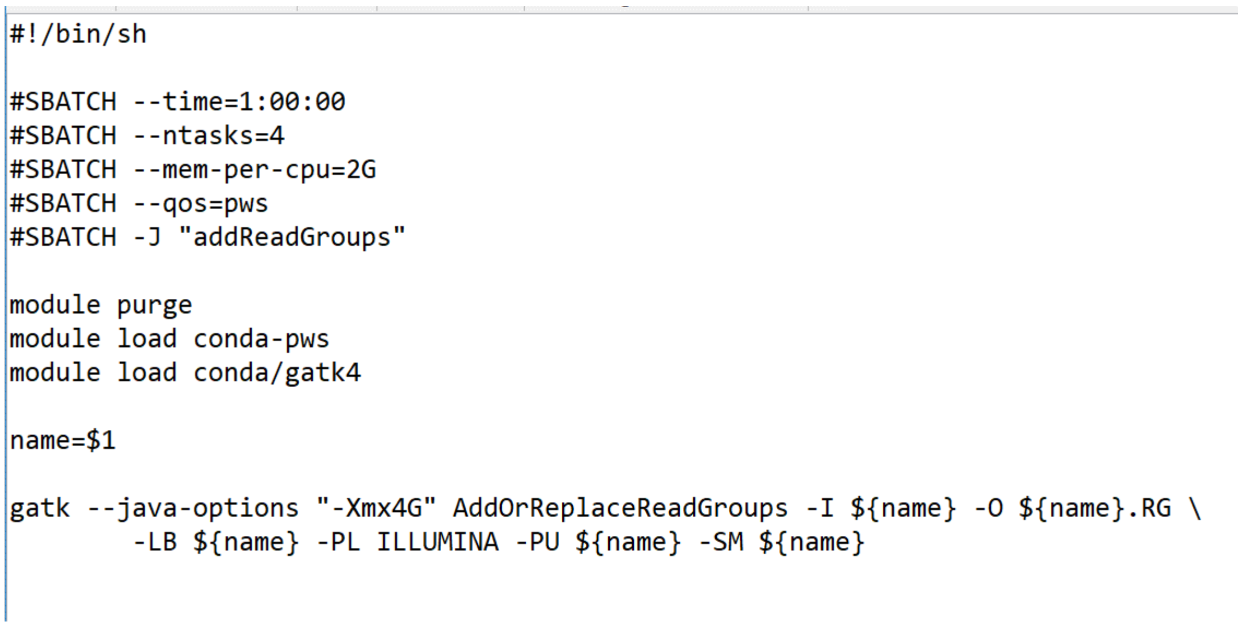

/panfs/pan.fsl.byu.edu/scr/grp/fslg_jarvis/Leos_population_FM/DANIEL_SCRIPTS/bam_and_sort.sh - mendias@ssh.fsl.byu.edu - Editor - WinSCP
Encoding
Color

```
#!/bin/sh

#SBATCH --time=72:00:00   # walltime
#SBATCH --ntasks=8   # number of processor cores (i.e. tasks)
#SBATCH --mem-per-cpu=8G   # memory per CPU core
#SBATCH --qos=pws

module purge
module load
module load conda-pws
module load samtools/1.6

samtools view -b -h  -o ../bam_files/$1.bam $1
samtools sort -@ 8 -o ../bam_files/$1.sorted.bam ../bam_files/$1.bam
samtools index -@ 8 ../bam_files/$1.sorted.bam ../bam_files/$1.sorted.bam.bai
```

In the script, samtools is used and the files are converted from SAM to BAM with “view”, sorted with “sort”, and then indexed (generate .bai files) with “index”. Make sure when running all these scripts that the script is in the file with the folders you are using the script on, unless you change the path names and such. I would also advise making an absolute path for output files which I did not. You should now have .bam and .bai files, I would put these in a sperate folder called bam_files. You can check how the mapping went by calling samtools flagstat on a bam file. Note that we haven’t marked duplicates yet so that field will be 0.

Add Read Groups

Now to add read groups. These are basically just some header stuff that HaplotypeCaller wants. To look at just the header of a bam file, use “samtools view -H myFile.bam”. To see the read groups, use “samtools view -H myFile.bam | grep “@RG””. HaplotypeCaller will want certain ones filled out. Here is how I did it:



```
#!/bin/sh

#SBATCH --time=1:00:00
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=2G
#SBATCH --qos=pws
#SBATCH -J "addReadGroups"

module purge
module load conda-pws
module load conda/gatk4

name=$1

gatk --java-options "-Xmx4G" AddOrReplaceReadGroups -I ${name} -O ${name}.RG \
        -LB ${name} -PL ILLUMINA -PU ${name} -SM ${name}
```

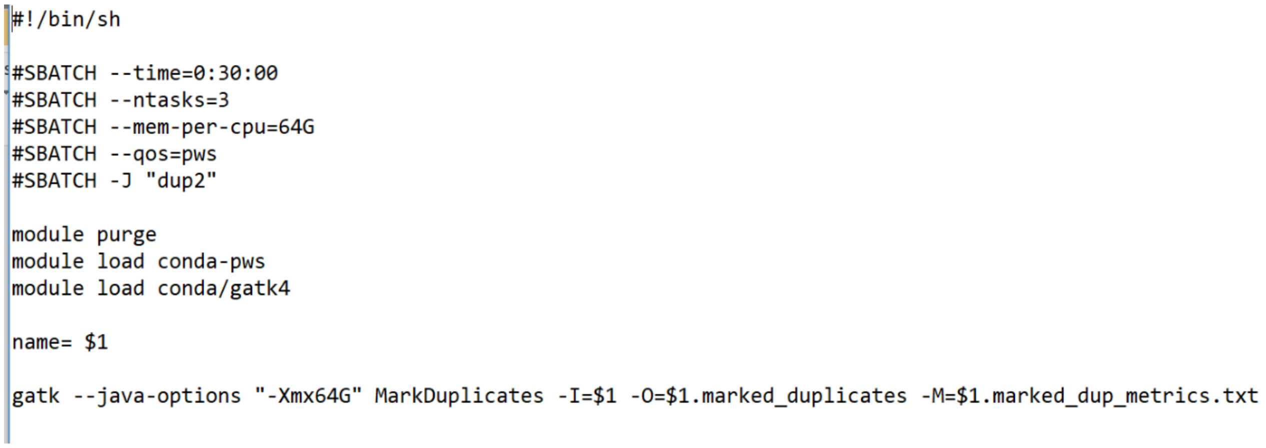
At the bottom of the script you can change the -PL parameter to the appropriate platform, though I’m not sure this will make a difference. For the other read groups you can see that I just threw in the file name just so it can have something unique. I don’t think it actually matters what goes there as long as it is unique. You can probably find what’s actually supposed to go there by looking at the headers of your bam files. I believe you can check whether or not GATK likes the bam file by running ValidateSamFile with:

```
gatk ValidateSamFile -I myFile.bam -IGNORE_WARNINGS=true -M=SUMMARY
```

If this gives you errors, you may be able to just ignore them and move on. If haplotypeCaller doesn’t work because the files are wrong, go back to this.

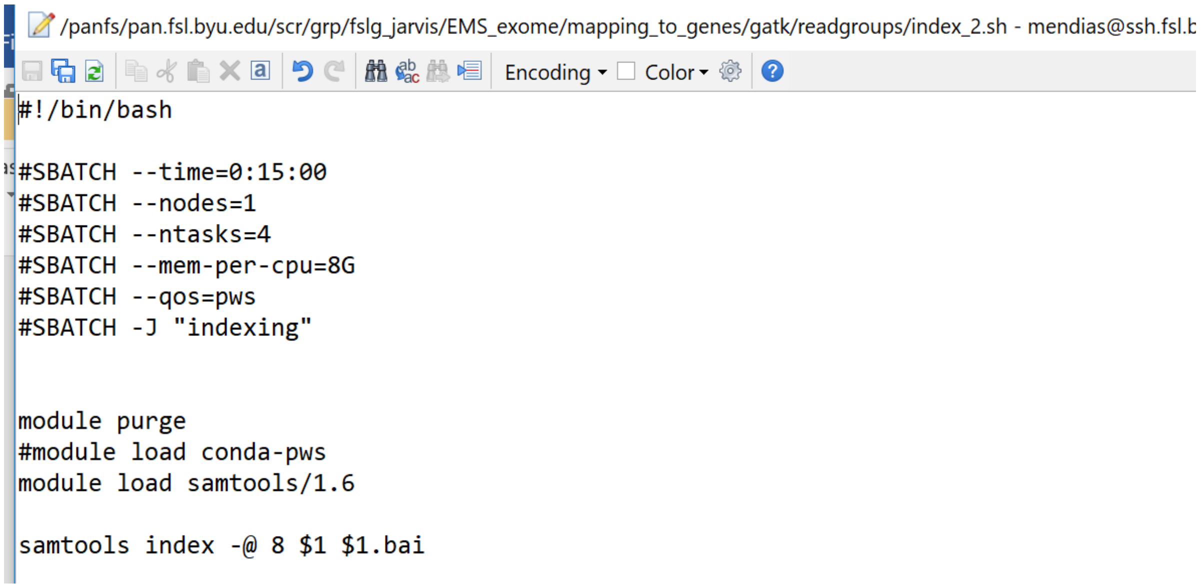
Mark Duplicates

Now to mark duplicates; this is to tag reads that are the exact same because it will give false confidence for SNP calls. This is done with markDups.sh. It will generate metrics file that will tell you how the process went.


```
#!/bin/sh

#SBATCH --time=0:30:00
#SBATCH --ntasks=3
#SBATCH --mem-per-cpu=64G
#SBATCH --qos=pws
#SBATCH -J "dup2"

module purge
module load conda-pws
module load conda/gatk4

name= $1

gatk --java-options "-Xmx64G" MarkDuplicates -I=$1 -O=$1.marked_duplicates -M=$1.marked_dup_metrics.txt
```

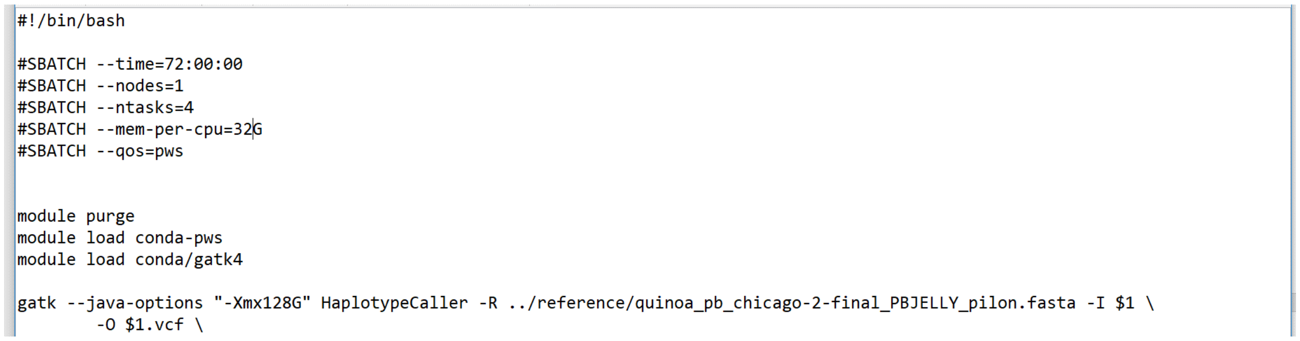
Calling SNPs

Now to call SNPs. First index the .marked_duplicates files. This will make some .bai files that are your index files.


/panfs/pan.fsl.byu.edu/scr/grp/fslg_jarvis/EMS_exome/mapping_to_genes/gatk/readgroups/index_2.sh - mendias@ssh.fsl.b

```
#!/bin/bash

#SBATCH --time=0:15:00
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=8G
#SBATCH --qos=pws
#SBATCH -J "indexing"


module purge
#module load conda-pws
module load samtools/1.6

samtools index -@ 8 $1 $1.bai
```

Next, run haplocaller.sh on all the files. **Make sure the .bai files are in the same folder as the corresponding marked duplicate bam files!** HaplotypeCaller will not work unless you have these in the same folder. This will create .vcf and .vcf.idx files. These contain your SNPs.


```
#!/bin/bash

#SBATCH --time=72:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=32G
#SBATCH --qos=pws


module purge
module load conda-pws
module load conda/gatk4

gatk --java-options "-Xmx128G" HaplotypeCaller -R ../reference/quinoa_pb_chicago-2-final_PBJELLY_pilon.fasta -I $1 \
        -O $1.vcf \
```

Here I included "Xmx128G" to ask for 128G of ram, but I think it's overridden by the parameter I asked the super computer for. It's probably fine to just put "gatk HaplotypeCaller …".

Filtering SNPs

To filter SNPs, I would look at VCFtools and GATK has some as well.

Other Helpful Things

- I make a folder holding the files for each step. That way things are more organized, and I can go back to steps if things need to be redone
- You'll probably have to change some of the scripts for them to work for you. I allocated random amounts of memory, so I would check your job stats on the FSL website after running a job to see how much time and memory you actually need.
- Before committing to a for loop, try one file first to make sure it works
- To look at the available modules: module avail
- To load modules: module load myModule
- To load samtools and gatk and other modules in the conda package (they have a dependency):

  ```
  module load conda-pws
  module load theModuleIWant
  ```

- Once loaded, you can get documentation for the module by entering (using samtools for this example):

  ```
  samtools
  samtools -h
  samtools - -help
  man samtools
  ```

  One of these will probably work for the module loaded. Also look at the online documentation.

- To remove all modules: module purge
- To see what else you can do with the module command: module -h
- To visually see the bam files and other files, IGV viewer is useful
- To look at your current jobs: squeue -u myUsername
- To see the number of jobs running: squeue -u myUsername | wc -l
- To cancel all jobs running: scancel -u myUsername
- To connect to the super computer on a PC, follow this video, and then open up ubuntu and enter: "ssh myUsername@ssh.fsl.byu.edu". I think using Ubuntu this way doesn't partition part of your hard drive. You can also use Putty.

- WinSCP is a great SCP client that lets you move files to and from your computer and the super computer. It also allows you to edit the scripts in notebook instead of in vim or some other linux text editor.
- You can look at all the jobs running on the node with: ssh nodeName. The node name will be the far right column when you type in squeue -u myUsername.

Running jobs

```
#!/bin/sh
#SBATCH --time=24:00:00   # walltime
#SBATCH --ntasks=4   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH --mem-per-cpu=16G   # memory per CPU core
#SBATCH -J "SNP caller"   # job name
#SBATCH --mail-user=myEmail@byu.edu   # email address
#SBATCH --mail-type=FAIL
#SBATCH --qos=pws


module purge
module load conda-pws
module load conda/gatk4

gatk --java-options "-Xmx128G" HaplotypeCaller -R \
../reference/quinoa_pb_chicago-2-final_PBJELLY_pilon.fasta -I $1 \
      -O $1.vcf
```

Basically the files you make to submit jobs will be text file (you can make one in notepad) looking like this with .sh at the end. They need to include #!/bin/sh at the very top so the supercomputer knows it's written in bash (I think it's bash). I'm pretty sure if we wanted to change it to a python script we would just change the file extension and the #!/bin/sh accordingly.

Anyways, the supercomputer will be looking for the #SBATCH headers for information. The #SBATCH is a comment that the supercomputer recognizes. Not all of them are required, I think its fine if you just include the time, number of cores, and memory per CPU. The wall time parameter is the maximum time your job can run for. If you ask for 24 hours and your job goes over it, the job will be killed and all the progress will be lost. Asking for less time will make a huge difference in the queue priority.

A few notes about the headers:

- “ntasks” is the number of CPUs you want. if you multiply this by the mem-per-cpu parameter, that is the total amount of memory you're asking for.
- “nodes” is the number of computers, it can just be left out because most programs probably won't need two computers.
- You can give the job a name if you want and have and email sent you if the job fails. I would recommend not putting your email address on there otherwise each job will send you an email and your inbox will blow up.
- The --qos=pws isn't necessary, it's just says what server we want to use, it can be left out. This is the plant and wildlife server

SUBMITING THE JOB

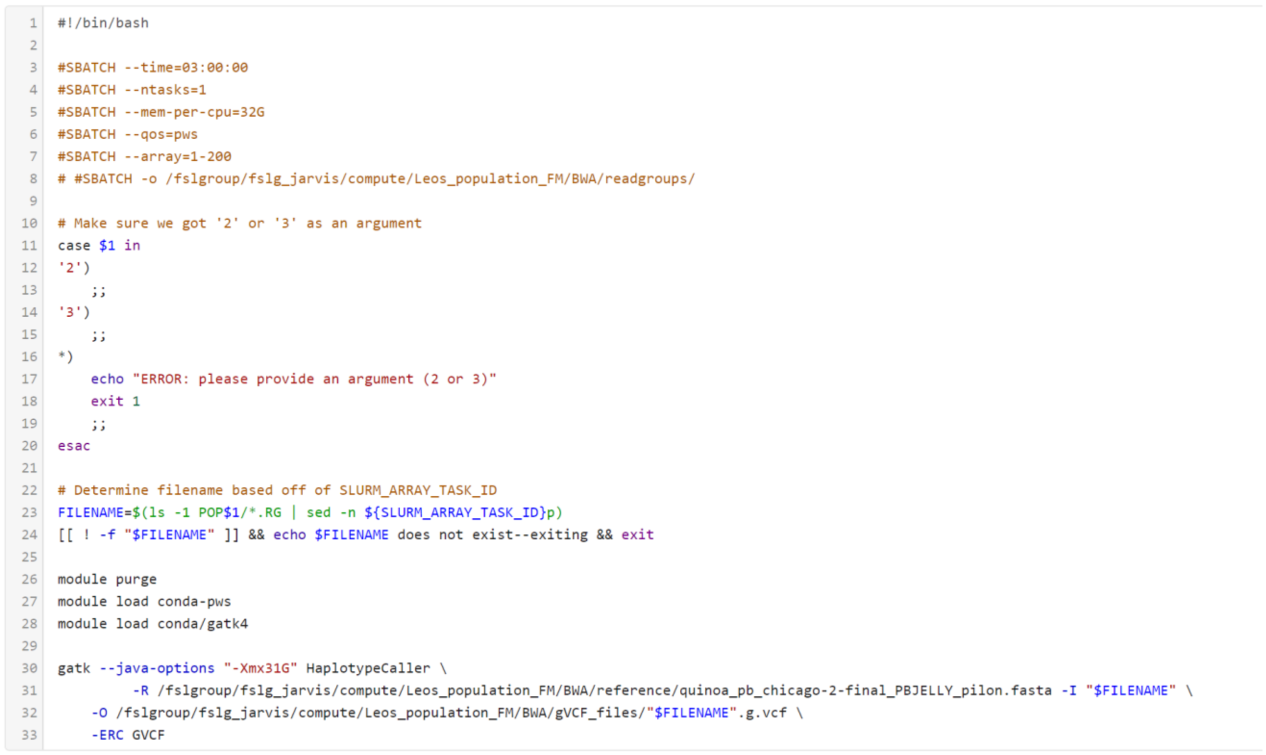
To submit a job, run something like this on the command line:

```
sbatch myNewFunJob.sh myFile.fastq
```

In your .sh file, you can reference the parameter ("myFile.fastq") with $1. If there is a second parameter use $2 and so on. To submit a bunch of jobs in one command, use a bash for loop which looks something like this:

```
for i in *.fq; do echo “processing $i”; sbatch mapping.sh $i; done
```

This will grab every file ending in .fq in the current folder, assign it to “i”, and then use that as the job parameter. This will submit a lot of individual jobs that may take a while to start running. You can also submit an array of jobs like I did in GVCFmaker-array.sh (see below), but it's weird and I don't really understand how it works. The array method gets your jobs submitted a lot faster though.



```
#!/bin/bash

#SBATCH --time=03:00:00
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=32G
#SBATCH --qos=pws
#SBATCH --array=1-200
# #SBATCH -o /fslgroup/fslg_jarvis/compute/Leos_population_FM/BWA/readgroups/

# Make sure we got '2' or '3' as an argument
case $1 in
'2')
    ;;
'3')
    ;;
*)
    echo "ERROR: please provide an argument (2 or 3)"
    exit 1
    ;;
esac

# Determine filename based off of SLURM_ARRAY_TASK_ID
FILENAME=$(ls -1 POP$1/*.RG | sed -n ${SLURM_ARRAY_TASK_ID}p)
[[ ! -f "$FILENAME" ]] && echo $FILENAME does not exist--exiting && exit

module purge
module load conda-pws
module load conda/gatk4

gatk --java-options "-Xmx31G" HaplotypeCaller \
         -R /fslgroup/fslg_jarvis/compute/Leos_population_FM/BWA/reference/quinoa_pb_chicago-2-final_PBJELLY_pilon.fasta -I "$FILENAME" \
    -O /fslgroup/fslg_jarvis/compute/Leos_population_FM/BWA/gVCF_files/"$FILENAME".g.vcf \
    -ERC GVCF
```

To see your current jobs, run:

```
squeue -u myUsername
```

this will pull up all your jobs. When your job finishes (or fails) it should make a slurm file which is just a text file telling you how your job went. If you want to cancel all your jobs, run:

```
scancel - u myUsername
```

To see how much memory your job actually took up and some other stuff, login to the supercomputer website and on the left side there will be a link to "Job Statistics".