Appendix A: Genomic analysis pipeline

General notes about the project

Below is the pipeline I used to analyze the Pool-seq data. I have generalized each step so that future students can adjust it to their projects. For my project, I analyzed five sites of *Physella johnsoni* and three sites of *Physella gyrina* of 20 to 40 individuals per site. These were sequenced by Genome Quebec on the Illumina HiSeq X, aiming for about 40x coverage (determined to be almost double that in actuality). Each site consisted of half a lane worth of data (total of four lanes).

The pipeline is not linear, in that there are analysis that branches off at certain points. As well, most of this was run on Cedar so the accessing of modules reflects that. Some of it was run on ARC though.

I included the SLURMs because it gives an idea of how long each thing took to run for my files. Rememeber that my files were half a lane each, a college of mine sequenced six pools over one lane and her analysis took a fraction of the time listed below.

Getting started in Cedar

Launch Terminal on a Mac. I think there is something called Putty for Windows?

Logging into Cedar

```
ssh yourcomputecanadaloginname@cedar.computecanada.ca
#ex. bstan@cedar.computecanada.ca
#You will be prompted for your password
#When you type it in nothing will appear but just hit "enter" when you've typed it in
#and it should log you right in!
```

Navigating around Cedar and creating a project directory

```
ls #will show you all the directories in your home folder

#How to make symbolic link to our project folder in def-srogers

rm project #removes the current project directory

ln -s projects/def-rogers/your_user_name project #assigns your account to project

cd project #change directory your in to project

pwd #will give you the path to the directory you are in

mkdir project_name #will make a directory in the project directory named "project_name".

cd project_name

mkdir 00_nameofstep1 #this is a good way to keep your steps in order

cd 00*tab* #by using "tab" on your keyboard it will auto-complete the name
```

It is worth spending some time learning Unix commands. I wish I had spent more time doing this, instead of trying to figure it out as I went.

How to make executable codes

You will need to choose a text editor to use on Cedar or Graham. We chose to use **GNU nano** because it is user and beginner friendly. To make exectuable codes do below:

```
nano codename
#will open a new, blank, nano with the name "codename"

type code

^X #this will close the code and you can select to save it.
#nano has functions listed at the bottom of the file - "^" is "control"

ls #will list all the files in the directory you are in - should see your code here!

chmod +x codename #this changes the code so that it is now exectuable, important!
```

How to submit jobs to run on Cedar

To run code on Cedar, you need to submit them as "jobs". The way this was explained to me is that you have to submit your request to the "secretary" of Cedar and they will send your job to the appropiate place. How we do this is something called "SLURM".

Create a SLURM

```
nano SLURM_name

#copy and paste below and change as necessary

^X #close and save SLURM
```

Example SLURM

```
  #!/bin/bash #Must put this at top
  # ---------------------------------------------------------------------
  # Place you can leave yourself a descriptor of this SLURM
  # ---------------------------------------------------------------------

  #SBATCH --job-name=Nameofyourjob #Make this descriptive but short!
  #SBATCH --account=def-srogers #the account this is under
  #SBATCH --cpus-per-task=XX #how many threads you want
  #SBATCH --time=0-00:00 #time you want - goes days:hours:minutes
  #SBATCH --mem=XXG #amount of memory you want

  # ---------------------------------------------------------------------
  echo "Current working directory: `pwd`"
  echo "Starting run at: `date`"
  # ---------------------------------------------------------------------

  /path/to/thecodeyouwanttorun

  # ---------------------------------------------------------------------
  echo "Job finished with exit code $? at: `date`"
  # ---------------------------------------------------------------------
```

Submit SLURM and check on status

```
sbatch SLURM_name #will submit job to queue

squeue -u your_user_name #will give you the job's status

scancel job_ID_number #will cancel the job
```

Once the job starts running, a SLURM out-file will be created in the format of: slurm-job_ID_number.out. This out-file gives you information on what step your job is on and whether or not it completed successfully. If successful it will have "Job finished with exit code 0". If it doesn't say that, then it may give you a helpful error code or a non-helpful error code.

Getting information on the job after it has run

```
acct -j job_ID_number --format=JobID,JobName,MaxRSS,Elapsed
#Gives JobID (kind of redundant), the name of the job, the memory and the time it took.
```

Convert BCL to Fastq - not allowing barcode mismatches

If working with BCL files you must first change them to fastq format. Genome Quebec will give you your files in fastq format. However, they allow one barcode mismatch. At times when you need 100% confidence of sequence assignment to the right population you will need to allow no mismatches. You may be able to ask Genome Quebec to do this on their end but I didn't know this until after they had done the conversion so I got the BCL files from them.

Manual: https://support.illumina.com/content/dam/illumina-support/documents/documentation/software_documentation/bcl2fastq/bcl2fastq2_guide_15051736_v2.pdf

tiles: these are the tiles that your data is on (because there will likely be other sequence info)

sample-sheet: need to get this from Genome Quebec. Just info about your sequences

-r 32 -p 32 -w 32: reading, processing and writing thread count. I set them all to 32.

use-bases-mask Y151,I6n2,Y151: this is specific about the sequencer - the Y151 is because it is PE 150 reads

Ex. bcl2fastq.1

```
module load bcl2fastq2/2.20 && \
bcl2fastq\
 --runfolder-dir /path/to/BCLfiles/171207_E00434_0072_AHGNNHCCXY_4467HS23A\
#the above is an identifier - change!
 --output-dir /path/to/00_raw_data\
 --tiles s_1\ #change to what tile you are converting - I had s_1 to s_4
 --sample-sheet /path/to/BCLfiles/171207_E00434_0072_AHGNNHCCXY_4467HS23A/SampleSheet.1.csv\
#change the above depending on tile - there is a SampleSheet for each tile
 --create-fastq-for-index-reads\
 -r 32 -p 32 -w 32\
  --barcode-mismatches 0 --use-bases-mask Y151,I6n2,Y151
```

```
  #!/bin/bash
  # ----------------------------------------------------------------------
  # Slurm for bcl2fastq for L001
  # ----------------------------------------------------------------------

  #SBATCH --job-name=bcl2fastq.1
```

```
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=32
#SBATCH --time=0-00:45
#SBATCH --mem=20G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

/path/to/bcl2fastq.1

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Concatenate files together - if sites were run on two or more sites on the flow cells

My sites were split over two lanes on the flow cells, so when I converted them from BCL to fastq I had four files for each site.

```
cat XXPool_L001_R1.fq.gz XXPool_L002_R1.fq.gz > XXPool_R1.fq.gz
```

FastQC - check the quality of your sequencing reads

Fastqc will give you a report on the quality of your sequencing reads. https://dnacore.missouri.edu/PDF/FastQC_Manual.pdf this is a pretty good tutorial on how to interpet them.

The code below will loop through every .fq.gz file in the directory you tell it to look in and make a report for each.

```
#!/bin/bash
# ---------------------------------------------------------------------
# FastQC for  pool sites
# ---------------------------------------------------------------------

#SBATCH --job-name=fastqc
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=1
#SBATCH --time=0-15:00
#SBATCH --mem=5G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load fastqc/0.11.5

for i in /path/to/fastqfiles/*fq.gz
```

```
do
        fastqc -o /path/to/where/you/want/reports $i
done
# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Trimmomatic - cleaning and filtering low-quality reads and removing adaptors

The available Adaptor file doesn't have all of the adaptors in it. Please contact me if you would like the file we created. We included all of the adaptors used in our library preparation.

phred: can be 64. Depends on your sequencing.

threads: change to the number of threads you have available on your cluster. We used 16, half of a node, but I think we could have used more. Just make sure to match this to what you are asking for in your SLURM.

ILLUMINACLIP: path to the adaptor file you made. **2**: seed mismatches **30**: is how accurate the match between the two adaptor ligated reads must be **10**: SimpleClip Threshold (I have a doc that goes in more detail if interested)

CROP: we were having an issue in some of our reads where Trimmomatic just wasn't detecting the repetitive sequences in the last 15 nucleotides. Hence the hard crop to trim the last 15 nucleotides off each read.

LEADING: trim the leading nucleotides if they fall under Q5

TRAILING: trim the trailing nucleotides if they fall under Q5

SLIDINGWINDOW: **5:20** - look at 5 bases at a time and trim if the average Q is less than 20

MINLEN: only keep reads that are minimally 100 bp

```
java -jar $EBROOTTRIMMOMATIC/trimmomatic-0.36.jar PE -phred33 -threads 16 -trimlog logfile \
path/to/fastqfiles/XXPool_R1.fq.gz /path/to/fastqfiles/XXPool_R2.fq.gz  \
XX_R1_P_qtrim.fq XX_R1_U_qtrim.fq XX_R2_P_qtrim.fq XX_R2_U_qtrim.fq \
ILLUMINACLIP:/path/to/Adaptors/TruSeq3-PE-all.fa:2:30:10 CROP:135 LEADING:5 TRAILING:5
SLIDINGWINDOW:5:20 MINLEN:100
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Trim cat files
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_trim
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-20:00
#SBATCH --mem=15G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load trimmomatic/0.36
```

```
/path/to/trimcode/trimcode

# -----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# -----------------------------------------------------------------------
```

Post-trim FastQC - check the quality of your sequencing reads

Same as above but for the post-trim files. Should see the quality go up and there be less sequences. Look for over-represented k-mers, there shouldn't be any!

DeconSeq - removing non-snail contaminants from the sequences

Full disclaimer I made the databases a variety of ways as I found out one way wouldn't work for every organism.

Step 1.1: How I created the databases for Archaea and Algae (Charophyceae, Chlorophyta, Cryptophyta, Eustigmatophyceae and Klebsormidiophyceae)

Create a database from NCBI of the sequences you would like to remove. You need to get the GI list from NCBI (as per http://johnstantongeddes.org/aptranscriptome/2013/12/31/notes.html or https://www.biostars.org/p/6528/).

You will need to install the newest version of NCBI BLAST+ (https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download).

I have heard that moving forward NCBI will be switching to using taxid to create databases rather than GI list but upon publishing this GI list were still being used.

In this example, I am using Archaea. **Key thing is that the nt database and your GI list must exist in the same directory.** It will be messy, which is why I put the "X" in front of the files I was making so that all of them were at the bottom.

```
/path/to/ncbi_directory/ncbi-blast-2.7.1+/bin/blastdb_aliastool -db nt -gilist Archaea.gi
-dbtype nucl -out X_nt_archaea -title "database for Archaea"
```

I just ran this in Cedar without a SLURM. Takes ~ 30 sec to a minute. Creates a .nal file, which if you put "X_nt_archaea" as your database, will mask everything else in the database but those sequences.

Once you've created the .nal file, you need to convert the database to .fasta file. It's small so I ran it in the SLURM.

```
#!/bin/bash
# -----------------------------------------------------------------------
# NCBI to fasta
# -----------------------------------------------------------------------

#SBATCH --job-name=Charo_fasta
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=1
#SBATCH --time=0-05:00
#SBATCH --mem=1G
```

```
# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load perl/5.22.4

/path/to/ncbi/ncbi-blast-2.7.1+/bin/blastdbcmd -entry all
-db X_nt_archaea -out Archaea.fasta

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Step 1.2: Creating the database for Threespine stickleback and Human

Threespine stickleback downloaded from: https://datadryad.org/resource/doi:10.5061/dryad.h7h32 (Peichel et al. 2017)

Human genome was downloaded from: ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/hs_ref_GRCh38.p12_ch.

I used the tutorial provided by DeconSeq to access the human genome.

```
#Download sequence data
for i in {1..22} X Y MT;
do wget
ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/hs_ref_GRCh38.p12_chr$i.fa.gz;
done

#Extracting and joining data
for i in {1..22} X Y MT; do gzip -dvc hs_ref_GRCh38.p12_chr$i.fa.gz
>>hs_ref_GRCh38_p12.fa; rm hs_ref_GRCh38.p12_chr$i.fa.gz;
done
```

Step 1.3: Creation of the Bacterial database

I downloaded all of the full bacterial genomes off of NCBI (I think there was about 10,000 of them?) and all assembly levels for bacteria that have been found in the thermal springs onto a computer (many Gb). I then used Globus to put them on to Cedar. https://www.ncbi.nlm.nih.gov/assembly/?term=bacteria

You will need to unzip any files that are zipped (including your query sequences) because DeconSeq can't use zipped files.

Can use:

```
for file in *.gz #loop through all files with this file extension
do
gunzip $file #unzip them
done #when it's finished, stop.
```

This will had to be done for all of the bacterial genomes. It took 10+ hours so I would consider submiting it as a SLURM.

Then I concatenated the bacterial genomes together, using:

```
#!/bin/bash
# ---------------------------------------------------------------------
# cat FG NCBI database
# ---------------------------------------------------------------------

#SBATCH --job-name=cat_bac
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-10:00
#SBATCH --mem=1G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

find . -name '*.fna' -print0 | xargs -0 cat > bacteria_genomes.fa

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Step 2: Splitting sequences by long repeats of ambiguous base N (this is from the DeconSeq manual

Below is all for the human genome because that is what is listed in the manual but I did this for all of the databases.

```
cat hs_ref_GRCh38_p12.fa | perl -p -e 's/N\n/N/' | \
perl -p -e 's/^N+//;s/N+$//;s/N{200,}/\n>split\n/' \
>hs_ref_GRCh38_p12_split.fa; rm hs_ref_GRCh38_p12.fa
```

Step 3: Filtering databases

Need to download and install **PRINSEQ** - can be found at https://sourceforge.net/projects/prinseq/files/

This step needs a SLURM because it will run out of memory otherwise

```
#!/bin/bash
# ---------------------------------------------------------------------
# Filtering sequences - PRINSEQ
# ---------------------------------------------------------------------

#SBATCH --job-name=human_prinseq
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=1
#SBATCH --time=0-0:10
#SBATCH --mem=10G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------
```

```
module load perl/5.22.4

perl /path/to/prinseq-lite-0.20.4/prinseq-lite.pl -log -verbose \
-fasta hs_ref_GRCh38_p12_split.fa -min_len 200 -ns_max_p 10 -derep 12345 \
-out_good hs_ref_GRCh38_p12_split_prinseq -seq_id hs_ref_GRCh38_p12_ \
-rm_header -out_bad null

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Step 4: Index the databases

For the bacterial database (because it is over 200Gb) you will need to first split it into managable chunks before BWA can use them. Can use fasta splitter (http://kirill-kryukov.com/study/tools/fasta-splitter/).

The files need to be under 3Gb each, so split it to as many chunks as you need. I did 100.

```
#!/bin/bash
# ---------------------------------------------------------------------
# FASTA splitter
# ---------------------------------------------------------------------

#SBATCH --job-name=fasta_split_bac
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-10:00
#SBATCH --mem=100G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load perl/5.22.4

perl /pathto/fasta-splitter.pl --n-parts 100 Bacteria_FG_split_prinseq.fasta

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Next step is to index the databases **You must use the BWA provided in the DeconSeq package!** The newer BWA reads the files incorrectly for this and will only produce 5 of the 8 outfiles necessary. Cedar is a 64 bit Linux system, so use bwa64. I know that the top examples have been using the human (sorry for the lack of consistency) but the bacterial one needed some extra things to make it run. I didn't want to run 100 SLURMs so this is a **batch** job!

Modified from script kindly provided by Dr. Stefan Dennenmoser

```
#!/bin/bash
# ---------------------------------------------------------------------
# BWA
# ---------------------------------------------------------------------
```

```
#SBATCH --job-name=index_bac
#SBATCH --ntasks=1
#SBATCH --account=def-srogers
#SBATCH --time=0-10:00
#SBATCH --mem-per-cpu=20G
#SBATCH --array=1-100

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

cd /path/to/Bacterial_Database

filename=`ls -1 *.fasta*  | tail -n +${SLURM_ARRAY_TASK_ID} | head -1`

filename2=${filename::-6} #the filename without the .fasta part (-6 letters)

/path/to/deconseq-standalone-0.4.3/bwa64 index -p $filename2 -a bwtsw $filename \
>bwa.log 2>&1

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Step 5: Configure the DeconSeq file

You will have to go into the installed DeconSeq directory and set up the configure file **DeconSeqConfig.pm**. You will just need to change the database directory and out directory and the what files are accessed in the "use constant DBS".

Step 6: Split the query sequences

DeconSeq is unable to handle big datasets (e.g. 50+ Gb files that I was using)

Will need to use fastq splitter (http://kirill-kryukov.com/study/tools/fastq-splitter/)

I don't think I ran this in a SLURM and just used the console. If it fails…put it in a SLURM.

```
perl /path/to/fastq-splitter.pl --n-parts 50 --check XX_R1.fq
```

Step 7: ACTUALLY RUNNING DECONSEQ

For DeconSeq you need to choose the identity (-i) which is the percent match between your query sequence and the database and the coverage (-c) which is the amount of the sequence aligns.

I went with the parameters they used in their paper and based on what I had seen other people do, which was 94% identity (-i 94) and 90% to 95% coverage (-c 90 or -c 95).

Submit this as a batch job.

```
#!/bin/bash
# ---------------------------------------------------------------------
# Deconseq
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_decon
#SBATCH --ntasks=1
#SBATCH --account=def-srogers
#SBATCH --time=3-00:00
#SBATCH --mem-per-cpu=7500MB
#SBATCH --array=1-50

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load perl/5.22.4

#put all of the split files from one pop and read direction in its own directory

cd /path/to/trimmed_seq/XX_R1_split

filename=`ls -1 *.fq*  | tail -n +${SLURM_ARRAY_TASK_ID} | head -1`

perl /path/to/deconseq-standalone-0.4.3/deconseq.pl -i 94 -c 90 -f \
/path/to/trimmed_seq/XX_R1_split/$filename -dbs hsref \
-out_dir /path/to/deconseq_out/XX_R1 -id $filename

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Step 8: Creating paired files again

DeconSeq was never designed for paired-end sequencing. Therefore it will process each read direction seperately and this causes sequences to be removed in one direction and not the other.

Firstly, concatenate the 50 (or more or less depending on what you split your in files to) of clean files. I also did this for the .cont files because I wanted to keep them and see how many were removed.

Then you can use this person's script found at: https://github.com/linsalrob/fastq-pair

Step 1: Clone or download - copy URL

Step 2: In Cedar or ARC or whatever cluster type: git clone -should see a directory called "fastq-pair"

Step 3: gcc fastq-pair/*.c -o fastq_pair

Step 4: Should see an exectuable script called "fastq_pair"

Running it is super simple.

```
path/to/fastq_pair -t VALUE path/to/file_R1.fastq path/to/file_R2.fastq
```

Where VALUE is roughly the number of sequences in your file. This is the setting of the hash table size. For some reason I don't know why, I couldn't get this to run as a SLURM. It would make the outfiles (if I gave it over 50Gb) but it would run for far longer than it runs in the terminal and just never finish. I ended up running it in ARC's console because Cedar doesn't have enough resources allocated to their console.

It makes four output files. R1 paired and unpaired and R2 paired and unpaired. Then you will need to zip the files back up!

DISCOVAR *de novo* - assembling a reference genome

DISCOVAR *de novo* is very easy to use but there are downstream challenges of not having a reference genome. As well, it was designed for paired-end sequences of 250 bp sequenced from one individual prepared PCR-free. In my thesis you can see the impact of breaking these assumptions in the quality of the genome produced. If it is at all possible to run an individual with at least two insert sizes (ex. 1 kb and 5,000 kb), you will be able to likely generate a much more robust assembly. Ex. Schell et al. 2017.

You can download DISCOVAR *denovo* from here: https://software.broadinstitute.org/software/discovar/blog/?page_id=98. There are two ways of getting (probably lots more but these are the ones I use) info from the web onto Cedar.

Option 1 Go to the link you want to download. Right click and "Copy Link Address". Go to Cedar. Do below.

```
wget
ftp://ftp.broadinstitute.org/pub/crd/DiscovarDeNovo/latest_source_code/LATEST_VERSION.tar.gz
```

Option 2 Can download to your personal computer (if the file isn't too big) and then use Globus to transfer it from your computer to Cedar. Globus is supported by Compute Canada. You will have to login, download Globus to your computer and make your computer an endpoint.

Once you download DISCOVAR denovo, you will have to unzip it.

A little aside… #Clumpify (belongs to the BBMap/BBTools package)

Before I assembled the sequences, I used Clumpify to remove PCR duplicates (unlike Picard it doesn't need a reference genome, however I don't think it is as robust) because the library prep is intended to be PCR free. This is seen to improve the quality of the assembly. I also tried BBNORM to normalize the sequencing depth at around 60x coverage. It decreased the N50 but increased the MPL1 and decreased the estimated chimera rate. In hindsight, I think I should have investigated this assembly more and maybe used it.

dedupe: The command to remove duplicates

subs=2: This means that there can be two subsitutions between the the compared sequences and it will be considered a duplicate.

```
clumpify.sh in1=/home/youraccount/path/to/trimmed.fq.gz/XX_R1_P.fq.gz \
in2=/path/to/trimmed.fq.gz/XX_R2_P.fq.gz out1=/path/to/Clumpify/XX_R1_P_nodup.fq.gz \
out2=/path/to/Clumpify/XX_R2_P_nodup.fq.gz dedupe subs=2
```

```
#!/bin/bash
# -------------------------------------------------------------------
# Removing duplicates from XX allowing two subs
# -------------------------------------------------------------------

#SBATCH --job-name=nodup_XX
#SBATCH --account=def-srogers
#SBATCH --nodes=1
#SBATCH --cpus-per-task=32
#SBATCH --time=0-00:30
#SBATCH --mem=150G

# -------------------------------------------------------------------
```

```
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09
module load intel/2016.4
module load bbmap/37.36

/home/youraccount/Clumpify/code/XX_clump

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Now that we have files with duplicates removed, back to DISCOVAR de novo

DISCOVAR *de novo* takes A LOT of memory.

```
DiscovarDeNovo READS=/path/to/fastq/files/you/want/to/use/*fq.gz \
OUT_DIR=/path/to/referencegenome \
MAX_MEM_GB=1450 NUM_THREADS=32
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Assembly of genome (using 1 site)
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_ref
#SBATCH --account=def-srogers
#SBATCH --nodes=1
#SBATCH --cpus-per-task=32
#SBATCH --time=0-30:00
#SBATCH --mem=1400G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

path/to/code/DDN_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

BWA (Burrows-Wheeler Aligner) - aligning sequences to the reference genome

Step 1: Index the reference

This step will create a bunch of index files using the name "reference_genome" as the name of the file

```
bwa index -p  reference_genome /path/to/reference/reference_genome.fa
```

```
#!/bin/bash
# ----------------------------------------------------------------------
# BWA
# ----------------------------------------------------------------------

#SBATCH --job-name=ref_index
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-04:00
#SBATCH --mem=5G

# ----------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ----------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4
module load intel/2017.1

module load bwa/0.7.17

/path/to/code/bwa_index_code

# ----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ----------------------------------------------------------------------
```

Step 2: Align to reference

Details about parameters can be found here: http://bio-bwa.sourceforge.net/bwa.shtml

-M: mark shorter split hits as secondary (necessary for using the file in Picard downstream)

-t 16: number of threads. Match with SLURM

-R: complete read group header line

XX: need to change XX to whatever site you are currently working on.

The reference genome needs to be put in without the ".fa" because it will be using all of the indexes that are in the directory too.

```
bwa mem -M -t 16 -R '@RG\tID:XX\tLB:XX\tSM:XX\tPL:ILLUMINA' \
/path/to/ref/reference_genome \
path/to/XX_R1.fq.gz path/to/XX_R2.fq.gz  > XX_out.sam
```

```
#!/bin/bash
# ----------------------------------------------------------------------
# sam
# ----------------------------------------------------------------------

#SBATCH --job-name=XX_align
```

```
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-05:00
#SBATCH --mem=10G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4
module load intel/2017.1

module load bwa/0.7.17

/path/to/code/align_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Samtools - sort the .sam file into a .bam by chromosome number

There are two options for sorting but it needs to be sorted by chromosome for downstream applications.

-@: this argument is where you set the number of threads

-T: this argument is where you set the indentifier - change to which ever site you are currently working on

-o: write to this outfile

```
samtools sort -@ 16 -T XX -o XX.bam XX_out.sam
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Samtools  Sort
# ---------------------------------------------------------------------

#SBATCH --job-name=XX.bam
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-04:00
#SBATCH --mem=5G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4
module load samtools/1.5

/path/to/code/samsort_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Samtools - filter low quality reads and where only one read mapped

Details can be found at: http://www.htslib.org/doc/samtools.html

Need to remove where only one read mapped, duplicates and low alignment quality reads (Q20). I did this in three steps, where I removed 1 read mapped first and then removed dups, and then filtered for Q20.

-@ : number of threads

-f 2: this means only keep it if there is paired reads

-o: write to this outfile

samtools has good documentation online. Be advised that they updated the package in April 2018! Just need to look at the right info.

```
samtools view -@ 16 -f 2 -o XX_rm1mate.bam /path/to/XX.bam
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Samtools  Sort
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_rm_Q20
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-02:00
#SBATCH --mem=25G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4

module load samtools/1.5

/path/to/code/rm1mate_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Picard - removing duplicates

Picard will remove duplicates using the MarkDuplicates function. You need to add REMOVE_DUPLICATES=TRUE to remove them. GATK suggests that you keep the marked duplicates but PoPoolation1 and 2 needs them removed.

```
java -jar $EBROOTPICARD/picard.jar MarkDuplicates \
INPUT=/path/to/XX_rm1mate.bam OUTPUT=XX_rm1mate_nodup.bam \
METRICS_FILE=XX_rm1mate_Q20_nodup.txt REMOVE_DUPLICATES=TRUE
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Picard Dup removal
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_nodup_rm1mate_Q20
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-02:00
#SBATCH --mem=40G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09

module load picard/2.17.3

/path/to/code/removeduplicates_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Flagstat - get stats on your alignment

Flagstat will give you stats on the alignment - like percent mapped.

```
#!/bin/bash
# ---------------------------------------------------------------------
# Flagstat
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_flagstat

#SBATCH --account=def-srogers

#SBATCH --cpus-per-task=16
#SBATCH --time=0-00:20
#SBATCH --mem=1G
```

```
# -----------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# -----------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4

module load samtools/1.5

samtools flagstat path/to/XX_rm1mate_nodup.bam

# -----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# -----------------------------------------------------------------------
```

Picard - Validate the .bam file before moving forward

Check if the bam file is not broken.

```
java -jar \
$EBROOTPICARD/picard.jar ValidateSamFile I=/path/to/XX_rm1mate_nodup.bam MODE=SUMMARY
```

```
#!/bin/bash
# -----------------------------------------------------------------------
# Validate Bam Test
# -----------------------------------------------------------------------

#SBATCH --job-name=Vali
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-01:00
#SBATCH --mem=15G

# -----------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# -----------------------------------------------------------------------

module load nixpkgs/16.09

module load picard/2.17.3

/path/to/code/validate_code

# -----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# -----------------------------------------------------------------------
```

Samtools - Q20 filter

-q 20: filter any sequence that the alignment score is less than 20 “Minimum mapping quality for an alignment to be used”

```
samtools view -@ 16 -q 20 -o XX_rm1mate_Q20_nodup.bam /path/to/XX_rm1mate_nodup.bam
```

```
#!/bin/bash
# ---------------------------------------------------------------------
# Samtools  Sort
# ---------------------------------------------------------------------

#SBATCH --job-name=XX_Q20
#SBATCH --account=def-srogers
#SBATCH --cpus-per-task=16
#SBATCH --time=0-02:00
#SBATCH --mem=25G

# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4

module load samtools/1.5

/path/to/code/Q20_code

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

Samtools - mpileup

Information about parameters can be found: http://www.htslib.org/doc/samtools.html

The mpileup is a file type that contains base-pair information at each chromosomal position.

In the below code you can add an arguement of -f and the path to the reference genome. This will give you the reference genome info in the mpileup file. I didn’t use this because I don’t really care what my reference is. In this step all of the .bam files will be combined into one mpileup (in my case site 1 through 8). For PoPoolation 1, you need to form a mpileup for each site which will be outlined later in this pipeline.

-B: stops BAC re-alignment. Necessary for PoPoolation2.

-o: write to this outfile

This example has three sites (XX, YY and ZZ) being combined into one mpileup

```
samtools mpileup -B \
XX_rm1mate_Q20_nodup.bam YY_rm1mate_Q20_nodup.bam ZZ_rm1mate_Q20_nodup.bam \
-o ref_allpools.mpileup
```

```
#!/bin/bash
# -----------------------------------------------------------------------
# Samtools mpileup
# -----------------------------------------------------------------------

#SBATCH --job-name=tut_allpools_mpileup

#SBATCH --account=def-srogers

#SBATCH --cpus-per-task=16
#SBATCH --time=0-06:00
#SBATCH --mem=1G

# -----------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# -----------------------------------------------------------------------

module load nixpkgs/16.09
module load gcc/5.4.0
module load intel/2016.4

module load samtools/1.5

/path/to/code/mpileup_code

# -----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# -----------------------------------------------------------------------
```

PoPoolation2 - convert mpileup to msync

The msync format is the file type that PoPoolation2 needs to do all of its analysis. It is also one of the formats that Poolfstat accepts as its input. You will need to download PoPoolation2 into your home directory. (https://sourceforge.net/p/popoolation2/wiki/Main/).

–**min-qual 20**: filter anything that has a base quality of less than 20

–**threads 8**: The number of threads you will be using. I have this faint memory (why didn't I type these notes as I went?) that this step goes a little squirrely if threaded higher than 8

```
java -jar /home/youraccount/popoolation2_1201/mpileup2sync.jar \
--input path/to/ref_allpools.mpileup --output ref_allpools.sync \
--fastq-type sanger --min-qual 20 --threads 8
```

```
#!/bin/bash
# -----------------------------------------------------------------------
# mpileup2sync
# -----------------------------------------------------------------------

#SBATCH --job-name=mpileup2sync

#SBATCH --account=def-srogers
```

```
#SBATCH --cpus-per-task=8
#SBATCH --time=0-2:00
#SBATCH --mem=15G

# -----------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# -----------------------------------------------------------------------

module load nixpkgs/16.09
module load java/1.8.0_121

/path/to/code/mpileup2sync_code

# -----------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# -----------------------------------------------------------------------
```

In the slurm.out it will give you exit code 1. Check that the last chromosome in the sync file matches the mpileup file and if it does, you're ok.

Poolfstat - pairwise *Fst*

I originally used PoPoolation2 to calculate pairwise FST but for reasons in my thesis, I think it is biased. So, I went with Poolfstat. It is an R package and it's so fast. Took about 4 hours to bring the data into R and then under ten minutes to run the calculation.

```
library(poolfstat)

#From their paper, which was re-analyzing Dennenmoser et al. 2017
#- where he had four populations of n=44 of prickly scuplin

pool.data = popsync2pooldata((sync.file="./file"), poolsizes=c(44,44,44,44),
poolnames=c(FE,CR,PI,HZ), min.rc=1,
min.cov.per.pool=10, max.cov.per.pool=300, min.maf=0.01, noindel=TRUE)

#I used:
min.rc=1
min.cov.per.pool=20
max.cov.per.pool=200
min.maf=0.05
noindel=TRUE
```

Once the data is in, you are set to run pairwise *Fst*

```
PW_fst <- computePairwiseFSTmatrix(pool.data, method = "Anova",
min.cov.per.pool=20, max.cov.per.pool=200, min.maf=0.05,
output.snp.values = TRUE)
```

I saved the $PairwiseFSTmatrix * and* NbOfSNPs$ components of the outfile as their own files because I wanted to keep them. The $PairwiseFSTmatrix file is necessary to do the below.

Visualizing pairwise *Fst*

To visual the Pairwise *FST* distance between each population, I used a Principal Coordinate Analysis. There is a really good blog post describing the different between PCoA and PCA here: http://occamstypewriter.org/boboh/2012/01/17/pca_and_pcoa_explained/.

```r
#read in the pairwise FST generated by Poolfstat
pcoa <- read.csv("PW_fst_matrix.csv", header=FALSE)

#make it a matrix
pcoa.matrix <- data.matrix(pcoa)

#calculate the euclidean distance between the pairwise FST
euc.matrix <- dist(pcoa.matrix,'euclidean')

library(labdsv) #package used for PCoA

pco <- pco(euc.matrix, k =2) #calculate the pco for euc distance

#sum the eigen vectors (8 populations)
sumeigen=pco$eig[1]+pco$eig[2]+...+pco$eig[8])

eig1=pco$eig[1]
eig2=pco$eig[2]
perc_eig1=eig1/sumeigen #percent explained by eigen vector 1
perc_eig2=eig2/sumeigen #percent explained by eigen vector 2

plot(pco) #will give you a not pretty figure

#the data is stored as character so this and below deals with that.
pco.ggplot<-data.frame(cbind(c("Pop1", "Pop2",..."Pop8"), as.numeric(pco$points[,1]),
as.numeric(pco$points[,2])))

pco.ggplot$X2<-as.numeric(as.character(pco.ggplot$X2))
pco.ggplot$X3<-as.numeric(as.character(pco.ggplot$X3))

colnames(pco.ggplot) <- c("site", "PCoA1", "PCoA2")

library(ggrepel)
library(RColorBrewer)
library(ggplot2)

tiff('PCoA', units="in", width=10, height=5, res=300)
ggplot(pco.ggplot, aes(x=PCoA1, y=PCoA2)) + geom_point(colour="chartreuse4")
+ geom_point(data=pco.ggplot[c(3, 4, 7), ], aes(x=PCoA1, y=PCoA2), colour="purple4")
+ geom_label_repel(aes(label = site), size = 3, hjust = 0, nudge_x = 0.003,
                   nudge_y = - 0.00, colour="chartreuse4", show.legend = FALSE)
+ theme(legend.position="none")
+ geom_label_repel(data=pco.ggplot[c(3, 4, 7), ],
                   aes(label = site, x=PCoA1, y=PCoA2), colour="purple4",
                   size = 3, hjust = 0, nudge_x = 0.003, nudge_y = - 0.00, show.legend = FALSE)
+ theme_bw() + theme(axis.text=element_text(size=13), axis.title=element_text(size=15,face="bold"),
                     panel.border = element_blank(), panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank(), axis.line = element_line(colour = "black"))
```

```
+ labs(x = "PCoA 1 (XX.XX%)", y = "PCoA 2 (XX.XX%)")
+ scale_x_continuous(breaks=c(-0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1.0),
                     labels=waiver()) + coord_fixed()
dev.off()

#I am going to be honest here with this ggplot code. It works.
#It ain't pretty and I just kept adding to it until it did a thing and now I am afraid to touch it.
```

PoPoolation1 - Nucleotide Diversity

The first step here is to make mpileups for each of your sites, instead of one with all of them. For example, in the above, you would just specify site XX.

–**fastq-type sanger**: even though our data is Illumina, we had to set it as sanger. This is because Phred 33, not 64

–**pool-size**: this should equal the number of individuals x 2 (for diploids) in the site. I have also read that if it is ok to put the number of individuals and that it makes so little difference they are considering taking that parameter out.

–**min-count**: minor allele count for that site

–**min-coverage**: the minimum coverage of a SNP for it to be used in analysis

max-coverage: the maximum coverage of a SNP for it to be used in analysis

–**window-size**: size of window

```
perl /home/youraccount/popoolation_1.2.2/Variance-sliding.pl --measure pi
--input /path/to/XX_rm1mate_Q20_nodup.mpileup --output XX_pi.file
--snp-output XX_pi.snps --fastq-type sanger --pool-size 40 --min-count 4
--min-coverage 20 --max-coverage 200 --window-size 250 --step-size 250
```

This will give you a .file and .snp file as the outfiles. The .file can be loaded into R and used to calculate mean nucleotide diversity and standard deviation.

```
XX_w250_pi <- read.delim("./XX_w250_pi.file", header = FALSE, sep = "\t", dec = ".")

colnames(XX_w250_pi) <- c("chr", "position", "Num.of.SNPs", "frac.of.cov", "pi")

XX_w250_pi$pi<-as.numeric(as.character(XX_w250_pi))

mean(XX_w250_pi$pi, na.rm=TRUE)

pi_matrix <- matrix(c("Pop1", "Pop2",..."Pop8", pi_1, pi_2, ...pi_8), nrow = 8, ncol = 2)

colnames(pi_matrix) <- c("Site", "NucleotideDiversity")

pi_DF <- as.data.frame(pi_matrix)

pi_DF$NucleotideDiversity <- as.numeric(as.character(pi_DF$NucleotideDiversity))

library(ggplot2)

tiff('pi_figure.tiff', units="in", width=5, height=5, res=300)
ggplot(data=pi_DF, aes(x=Site, y=NucleotideDiversity))
```

```r
+ geom_point(colour="purple4", size = 3) + labs(y= "Nucleotide Diversity")
+ geom_point(data=pi_DF[c(1, 2, 3, 4, 5), ], aes(x=Site, y=NucleotideDiversity),
             colour="chartreuse4", size = 3) + theme_bw() + ylim(0, 0.006)
+ theme(axis.text=element_text(size=13), axis.title=element_text(size=15,face="bold"),
        panel.border = element_blank(), panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(), axis.line = element_line(colour = "black"))
dev.off()

#Same as above regarding the ggplot.
```