# Pipeline GBS relatedness (Thyroptera)

# Information

### SOFTWARE USED ON LINUX VERSION CENTOS 7 (CORE)

- stacks version 2.4
- plink version 1.90b6.21
- plink2 version 2.00a3.7
- cutadapt version 1.9.1
- gcc version 7.2.0
- perl version 5.34.0
- gcta version 1.94.1
- R version 4.1.2:
  - sequoia version 2.5.6

### KEY REFERENCES

- Stacks manual: http://catchenlab.life.illinois.edu/stacks/manual/
- Rochette et al. 2019: https://onlinelibrary.wiley.com/doi/10.1111/mec.15253
- Rochette et al. 2017: https://www.nature.com/articles/nprot.2017.123
- Paris et al. 2017: https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.12775
- Chang 2020: https://core.ac.uk/download/pdf/286682225.pdf
- Plink manual: https://plink.readthedocs.io/en/latest/
- Sequoia vignette: https://cran.r-project.org/web/packages/sequoia/vignettes/vignette-main.html
- Huisman et al. 2017: https://onlinelibrary.wiley.com/doi/full/10.1111/1755-0998.12665

---

# Pipeline

## Step 1: File Organization

- Download and unzip the tar file from the sequencing facility:

```
tar -xf Chaverri_Project_003_analysis.tar
```

- Add both the pilot and large batch sequences to folder `raw`, and check that there are 114 samples total.

```
cp /work/jstynoski/batGBSpilot/raw/*.fastq.gz /work/jstynoski/batGBS/raw
cp /work/jstynoski/batGBS/Chaverri_Project_003_analysis/fastq/*.fastq.gz
/work/jstynoski/batGBS/raw
cd /work/jstynoski/batGBS/raw
ls -1 | wc -l
```

- Housekeeping: place all other files from UMCG in a folder "Archive". Files in `.fastq` format from unzipped original folder are now duplicates and can be deleted. The main folder `batGBS` should have two folders `raw` and `Archive`.

```
cd /work/jstynoski/batGBS
mkdir Archive
rm -r /work/jstynoski/batGBS/Chaverri_Project_003_analysis
```

## Step 2: Data Cleaning

The UMGC sequencing facility includes variable length "padding sequences" from 0 to 10 bases at the start of each read which helps with read accuracy and makes sequencing cheaper. The perl script called `gbstrim.pl` (repository: https://bitbucket.org/jgarbe/gbstrim/src/master/) will trim the padding sequences. Including the argument `--croplength 90` will trim all reads to the same length, which is a requirement for Stacks.

- Note: the version of `cutadapt` on our HPC (1.9.1) is too old to run multiple threads. To avoid an error about not detecting `--cores`, perl script line 184 ( `$cutthreads = ($threads) ? "--cores=$threads" : "";` ) was deleted along with the variable `$cutthreads` from line 185. This does extend run time slightly (10-30 min per sample).
- Run `gbstrim.slurm` in the `/batGBS/raw/` folder:

```
#!/bin/bash
#SBATCH --job-name=thyro_gbstrim
#SBATCH --output=gbstrim.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=2-00:00:00

module load cutadapt/Python3.5-1.9.1
module load perl/5.34.0

for SAMPLE in 1_S24 2_S34 3_S43 4_S53 5_S64 6_S75 7_S86 8_S96 9_S106 10_S10
11_S15 12_S16 13_S17 14_S18 15_S19 16_S20 17_S21 18_S22 19_S23 20_S25 21_S26
22_S27 23_S28 24_S29 25_S30 26_S31 27_S56 28_S32 29_S33 30_S57 31_S58 32_S35
33_S36 34_S37 35_S38 36_S39 37_S40 38_S41 39_S42 40_S44 41_S45 42_S46 43_S47
44_S59 45_S48 46_S49 47_S50 48_S51 49_S52 50_S54 51_S55 52_S56 53_S57 54_S58
55_S59 56_S60 57_S61 58_S62 59_S63 60_S65 61_S66 62_S67 63_S68 64_S69 65_S70
66_S71 67_S72 68_S73 69_S74 70_S76 71_S77 72_S78 73_S79 74_S80 75_S81 76_S82
77_S83 78_S84 79_S85 80_S87 81_S88 82_S89 83_S90 84_S91 85_S92 86_S93 87_S60
```

```
88_S94 89_S95 90_S61 91_S97 92_S98 93_S99 94_S100 95_S101 96_S102 97_S103
98_S104 99_S105 100_S1 101_S2 102_S3 103_S4 104_S5 105_S6 106_S7 107_S8
108_S9 109_S62 110_S11 111_S12 112_S63 113_S13 114_S14
do
perl gbstrim.pl --enzyme1 BamHI --enzyme2 nsiI --fastqfile
${SAMPLE}_R1_001.fastq.gz --read R1 --outputfile ${SAMPLE}.trim.fastq --
verbose --threads 24 --minlength 90 --croplength 90
done

date
time
```

- Verify in the `gbstrim.txt` output file that about 10% of reads were discarded, hopefully not more than 20%.
- Move all new `.trim.fastq` files to a new folder `/work/jstynoski/batGBS/raw/trim/`.

```
mkdir trim
cp /work/jstynoski/batGBS/raw/*.trim.fastq /work/jstynoski/batGBS/raw/trim/
cd /work/jstynoski/batGBS/raw/trim
ls -1 | wc -l
cd ..
rm /work/jstynoski/batGBS/raw/*.trim.fastq
```

- Because files arrived already demultiplexed from the sequencing facility, a "barcode file" does not need to be included in the cleaning step below, as mentioned in the Stacks manual. Also, the second digestion enzyme (`--renz_2 nsiI`) is not mentioned in the execution command because these GBS data are single-end (1x100).
- Clean trimmed raw files with `thyro_process_radtags.slurm` in the `batGBS` folder:

```
#!/bin/bash
#SBATCH --job-name=thyro_process_radtags
#SBATCH --output=thyro_process_radtags.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=3-00:00:00
#SBATCH --mail-user=stynoski@gmail.com
#SBATCH --mail-type=END,FAIL

mkdir out_process_radtags
module load gcc/7.2.0
module load stacks/2.4

process_radtags -p ./raw/trim/ -o ./out_process_radtags/ -c -q -r -e BamHI

date
time
```

- Housekeeping: Remove "trim." from file names in the folder `out_process_radtags`:

```
for f in *.gz; do mv "$f" "${f/trim./}"; done
```

## Step 3: Run a Pilot Analysis to Optimize Parameters

Uses the "r80" method (see Paris et al. 2017) which simulates a range of values for the parameters M (number of mismatches in a heterozygote), n (number of mismatches between population alleles), and m (number of reads required to initiate a new allele) and monitors the number of polymorphic RAD sites in at least 80% of the samples.
- It is recommended to set m to 3 because that works well for a lot of datasets. Also, start with n equal to M, and vary M and n with the same values from 1 to 9. An M too low will not allow alleles to collapse into loci, whereas with an M too high the paralogous loci will merge erroneously.

- Create a "population map" file of 8 pilot samples called `popmap1` (replace `<tab>` with an actual tab in nano):

```
27_S56<tab>pilot
30_S57<tab>pilot
31_S58<tab>pilot
44_S59<tab>pilot
87_S60<tab>pilot
90_S61<tab>pilot
109_S62<tab>pilot
112_S63<tab>pilot
```

- In `batGBSpilot`, create a folder `OptimizeDenovo` with subfolders from `DenovoM1` to `DenovoM9`.
- `OptDenovo.slurm` will execute Stacks with values of M (and n) from 1 to 9. FYI, this is a slow and computationally intensive process.

```
#!/bin/bash
#SBATCH --job-name=OptDenovo
#SBATCH --output=OptimizeDenovo.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=2-00:00:00
#SBATCH --mail-user=stynoski@gmail.com
#SBATCH --mail-type=END,FAIL

module load gcc/7.2.0
module load stacks/2.4

for Mn in 1 2 3 4 5 6 7 8 9
do
denovo_map.pl -T 8 -M ${Mn} -n ${Mn} -m 3 -o ./OptimizeDenovo/DenovoM${Mn} -
-samples ./out_process_radtags --popmap ./popmap1 --min-samples-per-pop 0.80
done
```

```
date
time
```

- Make a graph of the number of new SNPs identified in each round of increasing the M parameter. The highest value of M before the "new" SNP count becomes negative should be the chosen M.
  - M=4 was the best option, so repeat with M=4, n=3 and M=4, n=5.
  - M=4, n=5, m=3 gave the highest r80 value, so the pipeline uses those parameter values.

## Step 4: Build Loci and Catalog, and Match to Find Variants

For de novo alignment, `ustacks`, `cstacks`, and `sstacks` are run sequentially by a wrapper program called `denovo_map.pl`. It will also run `tsv2bam` to convert sample data to locus data, and run `gstacks` to assemble contigs and recall SNPs.

- Create a "population map" file with one line per sample that serves as a list of samples for Stacks called `popmap1` (replace `<tab>` with an actual tab in nano):

```
1_S24<tab>Baru
2_S34<tab>Baru
3_S43<tab>Baru
...
112_S63<tab>Baru
113_S13<tab>Baru
114_S14<tab>Baru
```

- Run `batDenovo.slurm` in the `batGBS` folder, which will execute the `denovo_map.pl` wrapper for all samples with the desired parameters.

```
#!/bin/bash
#SBATCH --job-name=batDenovo
#SBATCH --output=batDenovo.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=3-00:00:00
#SBATCH --mail-user=stynoski@gmail.com
#SBATCH --mail-type=END,FAIL

module load gcc/7.2.0
module load stacks/2.4

mkdir Denovo
denovo_map.pl -T 8 -M 4 -n 5 -m 3 -o ./Denovo --samples
./out_process_radtags --popmap ./popmap1
```

```
date
time
```

## Step 5: Filter and Export Variant Data

- In `batGBS`, create a VCF file with only a single SNP per RAD site, to avoid linked SNPs.

```
#!/bin/bash
#SBATCH --job-name=mkVCF
#SBATCH --output=mkVCF.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=03:00:00

module load gcc/7.2.0
module load stacks/2.4

populations -P ./Denovo --write-single-snp --vcf

date
time
```

## Step 6: Reformat Data for Pedigree Analysis

The goal is to filter and thereby reduce the number of SNPs in the VCF file (from 33k to 300-700) and convert the VCF to plink format for use in sequoia package in R.

- To "invent" chromosomes based on Stacks contigs (because plink is built for working with human genome data rather than non-model de novo RAD site data), change the name of the contigs in the VCF file:

```
awk '{if($0 !~ /^#/) print "contig"$0; else print $0}' populations.snps.vcf
> populations.snps.chr.vcf
```

- Create `.bed`, `.bim`, and `.fam` files using plink2 with special arguments to allow non-model species chromosomes and assign variant ID values:

```
#!/bin/bash
#SBATCH --job-name=cutPlink
#SBATCH --output=cutPlink.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=05:00:00

module load miniconda/3
source activate plink2-2.00a3.7
```

```
plink2 --vcf populations.snps.chr.vcf --make-bed --allow-extra-chr --out
batbed
plink2 --bfile batbed --set-all-var-ids @:# --allow-extra-chr --make-bed --
out batbed2

date
time
```

- Filter plink files by missingness and minor allele frequency. Typically filtering at this step also includes linkage disequilibrium, but it isn't possible with a de novo contig variant call because we would need a reference genome to identify loci on different chromosomes. To reduce SNPs to the required number (<700), more stringent missingness and MAF filters were used (0.05 and 0.45 instead of 0.1 and 0.3, respectively).

```
#!/bin/bash
#SBATCH --job-name=cutPlink2
#SBATCH --output=cutPlink2.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=03:00:00

module load miniconda/3
source activate plink-1.90b6.21

plink --bfile batbed2 --geno 0.05 --maf 0.45 --allow-extra-chr --make-bed --
out batbed3
plink --bfile batbed3 --allow-extra-chr --recode A --out for_sequoia

date
time
```

- This step created a list of 454 SNPs in plink format. The process was repeated using MAF of 0.4 instead of 0.45, which created a list of 870 SNPs called `for_sequoiaB.raw`. The R package sequoia (see below) was better able to assign relatedness pairs with the list of 454 SNPs than with 870 SNPs, so the list with 870 SNPs was disregarded.

## Step 7: Calculate Relatedness

- Download `for_sequoia.raw` for use in R on local machine.
  - In text editor, remove rows from `for_sequoia.raw` for individuals that are known duplicates: 2, 32, 33, 39, 45, 50, 54, 71, 78, 79, 80, 81, 82, 87, 92, 114
  - Housekeeping: For aesthetics, remove `_S##` from each sample name (indicates sample number from the sequencing facility, not relevant).
- Create a Life History `.csv` file with the columns ID, Sex, BirthYear, BY.min, and BY.max with each individual in a row. This file must use the exact same sample names in the edited `.raw` file.
- In R:

```
library(sequoia)
setwd("~/Google Drive/Projects/Bat RADseq")

##Import genotypes of 98 individuals and their life history data
Geno.sub<-GenoConvert(InFile = "subset.for_sequoia.raw", InFormat="raw")
LH <- read.csv("LHThyro.sub.csv", header=T)
CheckGeno(Geno.sub)

##Run Sequoia to create pedigree data
PedOUT.sub <- sequoia(GenoM = Geno.sub, LifeHistData = LH,Err=0.1,quiet =
FALSE, Plot = TRUE)
SummarySeq(PedOUT.sub)

##Create pedigree-based relatedness matrix
MatrixPedigree<-CalcRped(PedOUT.sub$Pedigree,OUT="M")
write.csv(MatrixPedigree,file="MatrixPedigreeAll.csv")

##Run GetMaybeRel to find parent-offspring pairs with missing birthyear data
PedOUT.sub.maybe<-GetMaybeRel(GenoM = Geno.sub, LifeHistData = LH)
write.csv(PedOUT.sub.maybe$MaybePar,file="MaybeParentPup.csv")
```

- Use the information in `MaybeParentPup.csv` to edit `MatrixPedigreeAll.csv` and improve relatedness within groups. Manually calculate group relatedness in Excel (one tab per group).

## Step 8: Verify if Genomic Relatedness Can Reduce Pedigree Error

The goal is to create a database of genomic-level relatedness and consider incorporating it into the Sequoia output (https://cran.r-project.org/web/packages/sequoia/vignettes/vignette-main.html#genomic-relatedness).

- Using GCTA (https://yanglab.westlake.edu.cn/software/gcta/#Inputandoutput), create a GRM (genomic relatedness matrix) to determine relatedness of pairs of individuals based on all 12,034 filtered SNPs.

```
#!/bin/bash
#SBATCH --job-name=GRMbat
#SBATCH --output=GRMbat.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=05:00:00

module load miniconda/3
source activate gcta

gcta64 --bfile batbed2 --autosome-num 33713 --maf 0.01 --geno 0.1 --make-grm
--out Thyro --thread-num 10

date
```

```
time
```

- Alternative method of creating GRM (may be less accurate):

```
#!/bin/bash
#SBATCH --job-name=PlinkGRM
#SBATCH --output=PlinkGRM.txt
#SBATCH --partition=nu
#SBATCH --ntasks=1
#SBATCH --time=03:00:00

module load miniconda/3
source activate plink-1.90b6.21

plink --bfile batbed2 --allow-extra-chr --geno 0.1 --maf 0.01 --make-grm-gz
--out Thyro

date
time
```

- Download Thyro.grm.gz and Thyro.grm.id to use in R on local machine:

```
# Read in output from GCTA
Rel.snp<-read.table("Thyro.grm.gz")
Rel.id<-read.table("Thyro.grm.id",stringsAsFactors=F)
Rel.snp[,1] <- as.character(factor(Rel.snp[,1], labels=Rel.id[,2]))
Rel.snp[,2] <- as.character(factor(Rel.snp[,2], labels=Rel.id[,2]))
names(Rel.snp) <- c("IID1", "IID2", "nSNPS", "R.GRM")
Rel.snp <- Rel.snp[Rel.snp$IID1 != Rel.snp$IID2,]  # between-indiv only

# Remove "_S#" from ID names in genomic relatedness matrix so that the two
matrices will match up
Rel.snp.noS<-data.frame(lapply(Rel.snp, function(x) gsub("_.*","", x)))

# Read in output from Kinship2
MatrixPedigree<-CalcRped(PedOUT.sub$Pedigree,OUT="DF")

# Combine with pedigree relatedness
library(data.table)
Rel.both <- merge(data.table(Rel.snp.noS[,c(1,2,4)], key=c("IID1", "IID2")),
                  data.table(MatrixPedigree, key=c("IID1", "IID2")),
all.x=TRUE)
Rel.both <- as.data.frame(Rel.both)  # turn back into regular dataframe

Rel.both$R.GRM<-as.numeric(Rel.both$R.GRM)
write.csv(Rel.both,file="Matrix of Genomic vs. Pedigree Relatedness.csv")
```

```
round(cor(Rel.both[, c("R.GRM","R.ped")],
          use="pairwise.complete"), 3)
#
# scatterplot doesn't work well with many thousand points
# >> use heatmap-like alternative, e.g. hexbinplot
hexbin::hexbinplot(Rel.both$R.GRM ~ Rel.both$R.ped,
                   xbins=100, aspect=1,
                   xlim=c(-.05,1.06), ylim=c(-.2, 1.06),
                   xlab="Pedigree relatedness", ylab="Genomic relatedness",
                   trans=log10, inv=function(x) 10^x,
                   colorcut=seq(0,1,length=14), maxcnt=10^6.5,
                   colramp = function(n) {grDevices::hcl.colors(n,
palette='Berlin')})

#Create database of possible relationships missed in the kinship data
PossibleMissing<-subset(Rel.both,Rel.both$R.GRM>0.2&Rel.both$R.ped<0.1)
write.csv(subset(Rel.both,Rel.both$R.GRM>0.2&Rel.both$R.ped<0.1),file="Possi
bleMissingRelationships.csv")
```

- The information generated by the GRM regarding potential missed relationships did not significantly improve pedigree errors based on associations within groups, so these data were not incorporated in final kinship matrices.