Streamlined GBS pipeline

Natalie J Forsdick

05 July, 2021

The following pipeline was produced as part of work carried out for my PhD thesis (Chapter 3, Natalie Jane Forsdick 2020) to conduct analysis of introgression using Genotyping By Sequencing (GBS) data as input. This research has since been published as Natalie J. Forsdick et al. (2021). I gratefully acknowledge my co-authors Liz Brown, Denise Martini, Hugh Cross, Richard Maloney, Tammy Steeves, and Michael Knapp.

This pipeline uses command-line bash commands to take raw GBS data for multiple individuals through analysis of introgression resulting from hybridisation, with additional analysis and visualisation of results with 'R v3.5.1. The steps included here should be run independently of one another. Variables should be modified according to the project-specific directories and names of input and output files, and available software versions.

Software requirements

Updated versions of software can be used, as long as this is done so consistently. Versions listed below are those used to process stilt data and are specified here for reproducibility.

- `Sabre v1.0`
- `Cutadapt v1.17`
- `SAMtools v1.9`
- `BWA v0.7.17`
- `STACKS v2.2`
- `VCFtools v0.1.15`
- `PLINK v1.9`
- `ADMIXTURE v1.3`

R package requirements

1. Demultiplexing, filtering and trimming raw data

Firstly we must demultiplex the GBS data containing sequences from multiple samples.

Requirements:

- Input FASTQ file of raw multiplexed GBS data (datafile.fq);
- FASTA file containing adapter sequences (adapters.fa);
- Tab-delimited TXT file containing two columns, the firstcontaining the barcode sequence and the second containing the respective sample file name to output (barcodes.txt), e.g., ACCATCG SQ063_CCHB8ANXX_s_1_1fq.fastq

Demultiplexing

Script name: GBS_preprocessing.sh

To run:

```
bash GBS_preprocessing.sh /path/to/datafile.fq /path/to/barcodes.txt \
/path/to/adapters.fa
```

GBS_preprocessing.sh script:

```
#!/bin/bash -e


# Check for arguments/help
if [ "$1" == "help" ]; then
      printf "\n This is a script to demultiplex, filter, and trim raw GBS data using
      sabre and cutadapt \n\n"
      printf "Three arguments are needed: the path to the FASTQ file containing raw
      GBS data,\n a the path to the tab-delimited text file containing sample
      barcodes and output file names,\n and the path to a FASTA file containing
```

```
      adapter sequences for removal."
      printf "For example:\n\t ./GBS_preprocessing.sh /path/to/datafile.fq
      /path/to/barcodes.txt /path/to/adapters.fa \n "
      exit 1

elif [ "$1" != "" ]; then
      printf "\t The raw data file is:  ${1}\n "
  else
  printf "\t You have not specified a file with raw GBS data for processing \n\n "
              exit 1
  fi

if [ "$2" != "" ]; then
      printf "\t The barcodes file is:  ${2}\n "
  else
    printf "\t You have not specified a barcodes file \n \n "
              exit 1
fi

if [ "$3" != "" ]; then
      printf " \t The adapter file is:  ${3} \n "
  else
    printf " \t You have not specified an adapter file \n\n "
              exit 1
fi

# Setting up a logfile to collect output metrics
start=`date`
echo "Logfile for GBS pipeline run on $start" > logfile1.txt
logfile=./logfile1.txt

echo "Demultiplexing with sabre"
```

```
echo "Demultiplexing with sabre" >> $logfile

# Make output directory
mkdir ./demultiplexed
cd ./demultiplexed/

# Command to run sabre v1.0 for single end data, uncomment the option needed:
#  -m 1 allows for 1 mismatch in barcode.
# -b = input file of sample barcodes,
# -u = output file of sequences with unknown barcodes.
sabre se -f $1 -b $2 -u unknown_barcode.fq > sabre_summary.txt
# sabre se -f ${datadir}${datafile} -m 1 -b ${datadir}SQ0683_barcodes.txt
#  -u unknown_barcode.fq > ${datadir}/sabre_summary.txt

echo "Demultiplexing complete"
echo "Demultiplexing complete" >> $logfile
echo "Cleaning and defining sample list"

# Calling a sample list for the next step from the demultiplexed files here
samplist=`ls -1 *.fastq | sed 's/.fastq//'`

cd ..
```

Filtering and adapter removal

Next we filter out data that does not contain the correct enzyme restriction site.

Script name: GBS_filtering.sh

To run:

```
bash GBS_filtering.sh
```

GBS_filtering.sh script:

```
echo "Filtering and trimming with cutadapt"
echo "Filtering and trimming with cutadapt" >> $logfile

mkdir ./filtered
echo "Filtering summary" > ./filtering_summary.txt
mkdir ./trimmed50
echo "Trimming summary" > ./trimming_summary.txt

for samp in $samplist
do

now=`date`
echo "Processing $samp $now" >> $logfile
echo "Filtering $samp"

cd ./filtered/

echo "$samp" >> ../filtering_summary.txt

# First we count the number of reads in the sequence file.
grep '^@' ../demultiplexed/${samp}.fastq | wc -l >> ../filtering_summary.txt

## Next we select only the reads that begin with the correct enzyme restriction site.
# This should be modified to the appropriate enzyme restriction site for your data.
grep -B1 -A2 '^TGCAG' ../demultiplexed/${samp}.fastq | sed '/^--$/d' > ${samp}.fastq

# We then count the number of reads with the correct enzyme restriction site.
grep '^@' ${samp}.fastq | wc -l >> ../filtering_summary.txt

cd ..
```

```
# Now to begin the adapter removal and trimming step.
echo "Trimming $samp"
cd ./trimmed50/

echo "$samp" >> ../trimming_summary.txt
grep '^@' ../filtered/${samp}.fastq | wc -l >> ../trimming_summary.txt

# Run cutadapt - this requires a FASTA file containing the appropriate adapter sequence(s)
# (adapters.fa as above).
# Reads shorter than (-m) 50 following adapter removal are removed.
cutadapt -a file:$3 -m 50 -o ${samp}.fastq ../filtered/${samp}.fastq >> \
  ../cutadapt_summary1.txt

# Check the number of reads remaining.
grep '^@' ${samp}.fastq | wc -l >> ../trimming_summary.txt

cd ..

echo "$samp processed"
echo "$samp processed" >> $logfile
done

end=`date`
echo "Filtering and trimming done"
echo "Filtering and trimming done $end" >> $logfile
```

This ends Step 1. All output files should be inspected for errors or anomalies before continuing.

2. Sequence mapping

Here we map the processed sequence reads to the reference genome of the focal species, so we can confidently call high-quality SNPs downstream. For this research, we mapped reads to the kakī reference genome (Galla

et al. 2019).

Script name: GBS_mapping.sh

This requires two arguments to run:

- A text file containing a list of FASTQ files to map (fqlist.txt);
- The path to the selected reference genome FASTA file (REFERENCE.fasta).

To run:

```
# Before running the script, you must move into the directory containing the FASTQ files.
cd /path/to/FASTQs/
bash GBS_mapping.sh /path/to/fqlist.txt /path/to/REFERENCE.fasta
```

GBS_mapping.sh script:

```
#!/bin/bash -e

# Set up environment and variables
# These are set up as modules in our SLURM environment
module load samtools/1.9 bwa/0.7.17

# Check for arguments/help
if [ "$1" == "help" ]; then
        printf "\n This is a script to map multiple sequence files to a single reference
        using BWA \n"
        printf "Two arguments are needed: a text file with a list of fq files to map,\n
        and the reference sequence to map against \n"
        printf "For example: \n\t ./simpler_mapping_loop.sh SEQUENCE_LIST
        /path/to/ref/REFERENCE.fasta \n\n"
        exit 1

elif [ "$1" != "" ]; then
        printf "\t The list file is:  ${1} \n"
```

```
  else
    printf "\t You have not specified a file with list of sequences to map \n\n"
                exit 1
  fi

if [ "$2" != "" ]; then
        printf "\t The reference file is:  ${2} \n"
  else
    printf "\t You have not specified a reference file \n\n"
                exit 1
  fi

# Ensure the reference genome is indexed.
if [ ! -f $2\.amb ]
    then
    printf "\n\t The reference has not been indexed. Indexing now \n"
    bwa index -a bwtsw $2
  else
    echo "BWA index file found"
  fi

start=`date`
echo "Beginning sequence mapping at $start"

# Then we run the mapping loop for all samples
# Map reads, convert to BAM format, sort by genome location, index outputs for
# downstream processes.
# -t = number of threads, -b = output in BAM format, -h = output header,
# -o = output filename
cat $1 | while read fq
do
```

```
    echo $fq
    base=$(echo $fq | cut -f 1 -d '.')

    bwa mem -t 4 $2 $fq | samtools view -b -h - | samtools sort - -o sorted_$base\.bam
    samtools index sorted_$base\.bam
done

end=`date`
echo "Variant discovery completed at $now"

##########

# Following mapping, we collect the percentage of mapped/unmapped reads from the
# output BAM files.
for bam in $(ls *.bam)
do

    echo $bam

    map=$(samtools view -F4 -c $bam)
    unmap=$(samtools view -f4 -c $bam)
    total=$(($map + $unmap))
    perc_mapped=`echo "scale=4;($map/$total)*100" | bc`

    echo $bam >> mappingv2.3_logfile.txt
    echo "mapped $map" >> mappingv2.3_logfile.txt
    echo "% mapped $perc_mapped" >> mappingv2.3_logfile.txt
    echo "unmapped $unmap" >> mappingv2.3_logfile.txt
done
```

Step 2 is complete. Logfiles containing mapping reports should be inspected before proceeding to Step 3.

3. Variant discovery

The next phase is variant discovery. Here we use the `STACKS` reference-guided pipeline for variant discovery.

Requirements:

- A tab-delimited text file containing a list of sample files and their associated population of origin (ref_population.txt)
- The reference genome as used in Step 2.

Script name: STACKS_varcalling.sh

To run:

```
bash STACKS_varcalling.sh /path/to/processed/samples/ /path/to/reference/ \
  /path/to/ref_population.txt
```

STACKS_varcalling.sh script:

```
#!/bin/bash -e

if [ "$1" == "help" ]; then
        printf "\n This is a script to call variants for a set of processed sample files
        using STACKS \n"
        printf "Three arguments are needed: the path to the directory containing the
        processed sample files,\n the path to the reference genome, \n and the
        ref_population.txt file \n"
        printf "For example: \n\t
        ./STACKS_varcalling.sh /path/to/processed/samples/
        /path/to/output/directory/ /path/to/ref_population.txt \n\n"
        exit 1

elif [ "$1" != "" ]; then
        printf "\t The directory containing processed sample files is:  ${1} \n"
else
```

```
    printf "\t You have not specified a directory containing processed sample files \n\n"
                exit 1
fi

if [ "$2" != "" ]; then
        printf "\t The reference file is:  ${2} \n"
else
    printf "\t You have not specified a reference file \n\n"
                exit 1
fi

if [ "$3" != "" ]; then
        printf "\t The populations file is:  ${3} \n"
else
    printf "\t You have not specified a populations file \n\n"
                exit 1
fi

# Set up environment variables
module load stacks/2.2

start=`date`
echo "Beginning variant discovery at $start"
# Adding -d will perform a 'dry-run' to test the pipeline without performing
# variant calling.
ref_map.pl -T 10 --samples $1 -o $2 --popmap $3
end=`date`
echo "Variant discovery completed at $now"
```

Step 3 is completed. This should produce a VCF containing variant call data for all individuals specified in the populations file. The VCF should be checked to ensure correct number of individuals, and that general formatting appears correct before proceeding to Step 4.

4. Variant filtering

Next we want to filter variants to produce a robust set of biallelic SNPs for downstream analyses. This step draws on the filtering pipeline described in http://www.ddocent.com/filtering, and the options available with `VCFtools`.

This requires output results to be considered step-by-step so filtering can be appropriately tailored to the data. Thus, this is not set up as a script, but as a series of line-by-line commands. Properties of the data set should be carefully considered when applying various filters to the data.

Throughout, we are specifying our input VCF with the `--vcf` flag, and a name for the output with the `--out` flag. Filtering steps and the resultant outputs should be carefully tracked.

```
# Setting up environment variables
module load vcftools/0.1.15

# May need to uncomment the below to set the path for perl if there are issues with
# VCFTOOLS.

# export PERL5LIB=/usr/local/vcftools_0.1.15/share/perl
```

First let's find out how many variants were produced from the `STACKS` pipeline.

```
grep "^[^#;]" variants.vcf | wc -l
```

Now let's begin filtering. This could be run with each filter separately, where the number of variants remaining is checked at each stage, but it is more efficient to run all steps in one pass. If very few variants (< 10K) are output, it may be best to assess effects of each filtering step independently.

Here the variables to alter each time are the names of the input and output VCFs.

First let's remove indels, and make sure we only have biallelic SNPs. Throughout, we are using the combination of `grep` and `wc -l` to determine the total number of variant sites remaining following filtering.

```
vcftools --vcf variants.vcf --remove-indels --min-alleles 2 --max-alleles 2 \
  --remove-filtered-all --recode --out filt_variants1

grep "^[^#;]" filt_variants1.recode.vcf | wc -l
```

Next we can remove sites with more than 10% missing data. An appropriate level of missingness should be determined for the focal variant set.

```
vcftools --vcf filt_variants1.recode.vcf --max-missing 0.1 \
  --remove-filtered-all --recode --out filt_variants2


grep "^[^#;]" filt_variants2.recode.vcf | wc -l
```

Next we want to check whether any samples have high levels of missing data - likely any individuals with low number of raw sequences or low number of mapped sequences will be an issue. Let's check the numbers and determine an appropriate cut-off.

```
vcftools filt_variants2.recode.vcf --missing-indv


# This makes a file called out.imiss, where individual missingness is listed.


# Examine the file
less out.imiss
```

We can visualise the distribution of missing data per individual.

```
mawk '!/IN/' out.imiss | cut -f5 > totalmissing
gnuplot << \EOF
set terminal dumb size 120, 30
set autoscale
unset label
set title "Histogram of % missing data per individual"
set ylabel "Number of Occurrences"
set xlabel "% of missing data"
#set yr [0:100000]
binwidth=0.01
bin(x,width)=width*floor(x/width) + binwidth/2.0
plot 'totalmissing' using (bin($1,binwidth)):(1.0) smooth freq with boxes
pause -1
EOF
```

We want to tailor individual filtering to suit the distribution of the data, but removing individuals with >

50% missing data is probably a good way to start. Let's make a list of those individuals with >5 0% missing data and remove them from the set.

```
mawk '$5 > 0.5' out.imiss | cut -f1 > lowDP.indv


vcftools --vcf filt_variants2.recode.vcf --remove lowDP.indv --recode --recode-INFO-all \
  --out filt_variants3.recode.vcf
```

At this point further exploratory analysis of the SNP set should be conducted - individual missingness, SNP density, depth, heterozygosity, HWE, population differentiation can all be assessed. Do the patterns make sense given the biological information? Are there any anomalies? What filtering is appropriate for the questions you aim to answer by downstream analyses, and what number of SNPs is required to answer these questions with sufficient power? See VCFtools manual for detailed instructions, and http:www.ddocent.com/filtering for additional suggestions.

The following is just an example, filtering on a minimum minor allele frequency of 0.01, retaining SNPs with depth between 5 - 200, and a minimum site quality of 20.

```
vcftools --vcf filt_variants3.recode.vcf --maf 0.01 \
  --minDP 5 --maxDP 200 --minQ 20 --remove-filtered-all --recode \
  --out filt_variants4


grep "^[^#;]" /path/to/filt_variants4.recode.vcf | wc -l
```

We also want to remove loci on the sex chromosomes, to avoid problems associated with sex-linked or haploid loci. These data could be assessed independently, or with tools that can appropriately incorporate haploid sites, depending on the question at hand.

```
vcftools --vcf filt_variants4.recode.vcf \
  --not-chr CHROM_Z --not-chr CHROM_W --remove-filtered-all \
  --recode --out filt_variants5.recode.vcf


grep "^[^#;]" filt_variants5.recode.vcf | wc -l
```

This completes Step 4. We should now have a good understanding of our SNP set, and be confident that it can produce robust results in downstream analyses.

5. Pre-processing for downstream analysis of introgression

Here we need to convert the VCF file to be passed as input to ADMIXTURE (or other analyses pipelines as required), using STACKS and PLINK. We also run the following code line-by-line on the command line.

Requirements:

We need to create a new populations file specifying the source population of all individuals (negfree_ref_population.txt), ensuring that filtered individuals are removed.

This process using STACKS produces useful estimates for population-level metrics including mean individuals per site, heterozygosity, population-specific variants/polymorphic sites, and private alleles.

The output PLINK format files are then passed to PLINK for conversion to BED, BIM, and FAM files that will be passed as input to ADMIXTURE.

```
# Set up the environmental variables
module load stacks/2.2
module load plink/1.9

# Here we output to PLINK format - output files will include MAP and PED files
# -V = VCF input, -O = output directory, -M = populations text file, -t = threads.
populations -V filt_variants5.recode.vcf -O ./ -M negfree_ref_population.txt -t 8 --plink

# Here we use --aec to specify non-standard chromosome format
plink --file filt_variants5.recode.plink --make-bed --aec --out filt_variants5
```

Following conversion, the BIM file requires modification before passing to ADMIXTURE. Chromosome/scaffold names cannot begin with a numeric, so add 'C' to the start; also the names cannot include underscores, so these need removing from column one. sed commands are extremely useful here.

If in addition to using ADMIXTURE we want to investigate population clustering with ADEGENET in R, we need to convert those same PLINK files to produce a RAW format file.

```
plink --file filt_variants3_lowDPrem_SexChromrem.recode.plink --aec --recode A
```

Remove header of MAP file, and sort RAW file by population before passing as inputs to ADEGENET.

This completes Step 5. Our files are now ready to be analysed for evidence of introgression (or population structure).

6. Analysis of introgression

While all previous steps can be run simply on a standard computer, `ADMIXTURE` analysis is conducted on a high-capacity computing cluster so that the multiple analysis runs can be performed in parallel using the `array` function, maximising efficiency and producing a robust analysis. Here we use a SLURM scheduling system, as in the original analysis. Original analysis used NeSI (New Zealand eScience Infrastructure, http:www.nesi.org.nz), with SLURM job queuing. Completing a full run with 100 iterations of $k = 1$ - 6, comprising ~15K - 140K SNPs took <2 hrs real-time. SLURM scheduling (`SBATCH`) details are included here to facilitate researchers in determining appropriate resource requirements. These should be modified based on the target data, and can be adapted to other scheduling methods. The following script can be modified to run on a standard computer or on other scheduling systems.

Biologically appropriate values of k should be assessed. In the following example, we test for the presence of up to ten clusters. A test subset of the data implementing a single run with the highest value of k may be useful to determine computational resource requirements for the target data set.

Script name: admixture.sl

This requires the output files from conversion (BED, BIM, and FAM) to be present in the directory.

To run:

```
sbatch admixture.sl
```

admixture.sl script:

The `SBATCH --array` flag can be modified to run various numbers simultaneously, e.g., –array 1-20%100 will run batches of 20 iterations simultaneously until all 100 iterations are completed.

```
#!/bin/bash -e


#SBATCH -J admix1 # Job name
#SBATCH --time 02:00:00 # real time requested
```

```
#SBATCH -c 36 # number of cpus per task
#SBATCH --mem=100G # maximum memory required
#SBATCH --array=1-100 # all 100 ADMIXTURE runs will be performed simultaneously
#SBATCH --partition=large # partition type
#SBATCH --mail-user=[user_email] # to receive notifications
#SBATCH --mail-type=ALL
#SBATCH --output admix1.%j.out
#SBATCH --error admix1.%j.err

##########
# Set up environment
module purge # Clear existing environmental variables

admixture=/path/to/admixture_v1.3/
start=`date`

OUTDIR=/path/to/output/

mkdir ${OUTDIR}/admixture_${SLURM_ARRAY_TASK_ID}
##########

cd ${OUTDIR}/admixture_${SLURM_ARRAY_TASK_ID}

echo "Beginning ADMIXTURE analysis at" $start

# -C = termination criterion - set to terminate when the log-likelihood increases by
# less than default of 0.0001 between iterations
# --cv = perform 10-fold cross-validation to assess appropriate number of clusters
# -s = seed (here we base this on the current time)
# -j = number of threads.

for k in {1..10};
```

```
do
    $admixture -C 0.0001 --cv /path/to/filt_variants5.bed $k -s time -j36 | \
    tee log_filt_variants5_${k}.out
done

end=`date`
echo "ADMIXTURE analysis completed at" $end
```

Step 6 is completed. Barring errors or further adjustments, we can now proceed to Step 7.

7. Processing ADMIXTURE outputs

First, we want to ensure that all the outputs are individually named, so that when we move these files, they don't get overwritten. Q files are renamed based on the run number. `ls` is used to check that the renaming is correct before replacing this with `mv` to rename the files. k values should be altered as above.

```
for k in {1..10}
do
    cd admixture_$k
    for i in *.Q
    do
        ls $i ${k}_${i} # once correct naming is confirmed, replace 'ls' with 'mv'
    done
    cd ../
done

# Then we want to move these all into one folder
for k in {1..10}
do
    for i in admixture_${k}/*.Q
    do
        ls $i admixture_Q/ # again, once correct movement is confirmed, replace 'ls'
        # with 'mv'
```

```
    done
done
```

We then want to extract the CV values to determine the appropriate number of clusters. We pull out CV values and sorted by the cluster number.

```
grep "^CV" admixture_*.out | sort -k 3 > admixture_CV_sorted.txt
```

This is the end of Step 7. These files can then be transferred to the local system ready for processing and visualisation with the `R` package `Pophelper`.

References

Forsdick, Natalie J., Denise Martini, Liz Brown, Hugh B. Cross, Richard F. Maloney, Tammy E. Steeves, and Michael Knapp. 2021. "Genomic Sequencing Confirms Absence of Introgression Despite Past Hybridisation Between a Critically Endangered Bird and Its Common Congener." *Global Ecology and Conservation* 28: e01681. https://doi.org/https://doi.org/10.1016/j.gecco.2021.e01681.

Forsdick, Natalie Jane. 2020. "Assessment of the Impacts of Anthropogenic Hybridisation in a Threatened Non-Model Bird Species Through the Development of Genomic Resources with Implications for Conservation." Thesis, University of Otago. https://ourarchive.otago.ac.nz/handle/10523/10268.

Galla, Stephanie J, Natalie J Forsdick, Liz Brown, Marc Hoeppner, Michael Knapp, Richard F Maloney, Roger Moraga, Anna W Santure, and Tammy E Steeves. 2019. "Reference Genomes from Distantly Related Species Can Be Used for Discovery of Single Nucleotide Polymorphisms to Inform Conservation Management." *Genes* 10 (1): 9. https://doi.org/10.3390/genes10010009.