Supplementary Data S1

**S1. System requirement**

OS: Linux or UNIX for S2, no specific requirement for S3

Software packages: Fastqc (v 0.11.4), Stacks (v .1.44), FastX ( v.0.0.14), Flash (v.1.2.11), bowtie2 (v.2.2.9), samtools (v.1.3.1), picard (v.2.4.1), GATK (v3.4.0), perl (v.5.20.1), Python (v.3.5.1 or higher)

Make sure the Python version is equal to or higher than v 3.0; codes are not compatible with any Python ver 2.x.x. Scripts work under Linux/UNIX/Windows/MacOS.

To test the pipeline (using demo files), the following software is needed:

**Operation system:** Ubuntu or other Linux system, Ubuntu 16.04 LTS is used here.

**Software installation:**

Install python:

*$ sudo apt-get update*

*$ sudo apt-get install python3.6*

Install perl

*$ sudo apt-get update*

*$ sudo apt-get upgrade*

*$ sudo apt-get install -y perl*

Install fastqc package:

Download fastqc from <http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>.

Switch to fastqc directory, do

*$ chmod +x fastqc*

*$ sudo ln -s PWD\_to\_FastQC/fastqc /usr/local/bin/fastqc*

Install fastx package:

Download fastx from <https://www.starnet.com/fastx/>

*$ sudo apt-get install alien*

*$ sudo alien --install --scripts StarNetFastX2-\*.rpm*

*$ sudo /usr/lib/fastx2/install.sh*

Install stacks:

Download stacks from <http://catchenlab.life.illinois.edu/stacks/>

*$ tar xfvz stacks-2.xx.tar.gz*

*$ cd stacks-2.xx*

*$ ./configure*

*$ make*

*$ sudo make install*

*$ ./configure --prefix=/home/local/bin*

*$ make -j 8*

Install flash

Download FLASH from <http://ccb.jhu.edu/software/FLASH/FLASH-1.2.11-Linux-x86_64.tar.gz>, unzip it and add flash to PATH.

Install Bowtie2

*$ sudo apt-get update*

*$ sudo apt-get install bowtie2*

Install samtools

*$ wget https://github.com/samtools/samtools/releases/download/1.3.1/samtools-1.3.1.tar.bz2 -O samtools.tar.bz2*

*$ tar -xjvf samtools.tar.bz2*

*$ cd samtools-{version}*

*$ make*

*$ sudo make prefix=/usr/local/bin install*

Install picard

*$ sudo apt-get install picard*

Install GATK

No installation require for GATK, download version 3.4 and version 3.6 GATK from <https://software.broadinstitute.org/gatk/download/>.

**S2. Generate reference from raw reads**

To generate a reference from raw reads:

1. Check sequencing quality of raw reads with ‘*fastqc*’ [1], and output results to a new directory “Fastqc\_result”

The script:

*fastqc/0.11.4/fastqc raw\_reads\_lane1\_read1.fastq -o Fastqc\_result*

1. Split the raw reads according to their barcode using ‘*process\_radtags*’ from Stacks [2].

For reads from each sequencing lane, make a separate barcode file for each length of barcode, run scripts below for each barcode file, and put the outputs in separate folders for each lane (like “lane1”). Check Stacks documentation for details and parameters.

The script:

*stacks/1.44/bin/process\_radtags -1 raw\_reads\_lane1\_read1.fastq -2 raw\_reads\_lane1\_read2.fastq –b barcode.txt –o lane1 --renz\_1 pstI –r --adapter\_1 CACGACGCTCTTCCGATCT --adapter\_2 CGAGATCGGAAGAGCGG --adapter\_mm 1*

1. If applicable, merge reads from the same sample sequenced on multiple sequencing lanes, and output the result to a new directory “BCraw”. Do this separately for read 1 and read 2. Make sure the files are merged in the same order for read 1 and read 2.

The script:

*cat lane1/sample.1.fq lane2/sample.1.fq > BCraw/sample1.1.fq*

*cat lane1/sample.2.fq lane2/sample.2.fq > BCraw/sample1.2.fq*

1. Using step 3 output as input, trim reads to remove restriction sites and low quality bases using ‘*fastx\_trimmer*’ (http://hannonlab.cshl.edu/fastx\_toolkit/). Put output in a new directory “BCpc”

The script:

*fastx/0.0.14/bin/fastx\_trimmer –Q 33 –f 6 -l 135 –i sample1.1.fq -o BCpc/sample1.1.fq*

*fastx/0.0.14/bin/fastx\_trimmer –Q 33 –f 4 -l 145 –i sample1.2.fq -o BCpc/sample1.2.fq*

Use –f 6 for *Pst*I, -f 4 for *Msp*I, -f 5 for *Nde*I, -l 135 for the forward reads, -l 145 for the reverse reads (these values are for a raw read length of 150 bp; -l is the read length remaining after removal of barcodes and five lower quality bases at the 3’ end of reads).

To test the pipeline (using demo files):

*$ mkdir BCpc*

*$ fastx\_trimmer –Q 33 –f 6 -l 135 –i 1M\_PM-KNE10.1.fq -o BCpc/ 1M\_PM-KNE10.1.fq*

*$ fastx\_trimmer –Q 33 –f 4 -l 145 –i 1M\_PM-KNE10.2.fq -o BCpc/ 1M\_PM-KNE10.2.fq*

Do this for each of the three demo samples.

1. In the “BCpc” directory, merge overlapping paired reads using ‘*flash*’ [3]. Do this for each sample.

The script:

*flash/1.2.11/flash sample1.1.fq sample1.2.fq*

Flash produces three files of interest: *out.extendedFrags.fastq* which contains the merged overlapping paired-end fragments, *out.notCombined\_1.fastq* which contains the non-overlapping forward reads and *out.notCombined\_2.fastq* which contains the corresponding non-overlapping reverse reads.

Rename these three files as follows:

 *mv out.extendedFrags.fastq sample1.Comb.fq*

 *mv out.notCombined\_1.fastq sample1.Solo.1.fq*

 *mv out.notCombined\_2.fastq sample1.Solo.2.fq*

To test the pipeline (using demo files):

*$ flash 1M\_PM-KNE10.1.fq 1M\_PM-KNE10.2.fq*

*$ mv out.extendedFrags.fastq 1M\_PM-KNE10.Comb.fq*

*$ mv out.notCombined\_1.fastq 1M\_PM-KNE10.Solo.1.fq*

*$ mv out.notCombined\_2.fastq 1M\_PM-KNE10.Solo.2.fq*

Do this for each of the three demo samples.

1. Put python *script EL.1.1.py* (EL for ‘equal length’) in the same folder as the flash output files. Python version must be 3.0 or higher. Run *EL.1.1.py* by double clicking *EL.1.1.py* in Windows/MacOS or typing “*python3 EL.1.1.py*” in Linux.

This script will (1) remove reads from the *Solo.1.fq* and *Solo.2.fq* files that are shorter than the expected (trimmed) size (if a forward read is removed, the corresponding reverse read is also removed, and *vice versa*), (2) reverse complement the reverse reads in the *Solo.2.fq* file and artificially join them to the 3’ end of forward reads in the *Solo.1.fq* file, and (3) extend merged overlapping reads at the 3’ end with ‘As’ to make them the same length as the joined non-overlapping reads.

The script will ask for the following inputs: Comb file full name (*sample1.Comb.fq*), Solo 1 file name (*sample1.Solo.1.fq*), Solo 2 file name(*sample1.Solo.2.fq*), read length for Solo 1 reads (any solo 1 reads shorter than this number will be removed), read length for Solo 2 reads (any solo 2 reads shorter than this number will be removed), and output file name (for example: *sample.fq*). The *EL1.1.py* script will generate two output files, *sample.fq* for ustacks input, and *dump\_sample.fq* which contains all removed reads.

To test the pipeline (using demo files):

*$ python EL.1.1.py*

When asked to 'Enter comb reads file full name:', enter: *1M\_PM-KNE10.Comb.fq*

When asked to 'Enter Solo 1 reads file full name:', enter: *1M\_PM-KNE10.Solo.1.fq*

When asked to 'Enter Solo 2 reads file full name:', enter: *1M\_PM-KNE10.Solo.2.fq*

When asked to 'enter the read length standard for Solo 1:', enter: 104

When asked to 'enter the read length standard for Solo 2:', enter: 84

When asked to 'enter output file full name:', enter: *1M\_PM-KNE10.fq*

Do this for each of the three demo samples.

1. Use ‘*ustacks*’ to cluster reads within each sample and generate a consensus sequence for each cluster; place the output file into a new directory “ASU”. Check Stacks documentation for details and explanation of parameters.

The script:

*stacks/1.44/bin/ustacks -t fastq -f BCfin/sample1.fq -o ASU/ -I 1 -m 2 -M 1 -N 1 -H --max\_locus\_stacks 2*

The parameters used in the stack formation will vary with the breeding system and polymorphism level of the species under investigation.

To test the pipeline (using demo files):

*$ mkdir temp*

*$ mv \*.Comb. \* \*.Solo.\* temp*

*$ mkdir BCfin*

*$ mv \*.fq ../BCfin*

*$ cd ../BCfin*

*$ mkdir ASU*

*$ ustacks -t fastq -f 1M\_PM-KNE10.fq -o ../ASU -I 1 -m 2 -M 1 -N 1 -H --max\_locus\_stacks 2*

Do this for each of the three demo samples.

1. Use ‘*cstacks*’ to cluster consensus sequences (generated by ‘*ustacks*’) across samples and output to a new directory “Cstacks”. Check Stacks documentation for details and explanation of parameters.

The script:

*Stacks/1.44/bin/cstacks -b 1 -s sample1 -s sample2 -s sample3 -o Cstacks/ -n 1 -p 8 --report\_mmatches.*

For large number of samples (more than 100), we suggest to set -p as high as possible (maximum to CPU threads of computer) to decrease calculation duration. ‘cstacks’ on a large number of samples is computationally very intensive. ‘ASustacks’ (see below) can be run as an alternative (recommended). As for *‘ustacks*’, parameters will need to be adjusted based on polymorphism level of the species under investigation.

1. Filter cstacks output and generate reference sequence.

In ‘Cstacks’ folder, use *FCT.pl* to filter the tags generated by ‘*cstacks*’; the minimum (will determine the number of missing data allowed) and maximum number of accession (typically all accessions) that have to be included in a consensus stack is given by the first and second numbers; only consensus tags will be selected that consist of maximum one ‘*ustack*’ tag per accession.

The script:

*perl FCT.pl batch\_1.catalog.tags.tsv 2 3*

Then rename the output to reference.

*mv cstacks\_output\_unique\_consensus\_multi\_fasta ref.fa*

To use the ‘*ASustacks*’ module instead of ‘*cstsacks*’ for the generation of stacks across accessions (recommended), replace steps 8 and 9 with the steps listed below:

1. Copy the python script ‘*ASustacks.py*’ to the “ASU” folder, and run ‘*ASustacks.py*’ for each sample:

*python3 ASustacks.py*

The script will ask for the file input; enter the full name of the *tags.tsv* file for a sample (for example, sample1.tags.tsv). The script will output an artificial .fastq file *sample1\_u.fq*.

To test the pipeline (using demo files):

*$ cd ../ASU*

*$ python3 ASustacks.py*

When asked to 'Enter the Ustacks output tags.tsv file:', enter: 1M\_PM-KNE10.tags.tsv

When asked to 'Enter the output fastq filename:', enter: 1M\_PM-KNE10.fq

Do this for each of the three demo samples.

1. Merge the artificial .fastq files from all samples into a single file. Use the script below:

*cat sample1\_u.fq sample2\_u.fq sample3\_u.fq > all\_u.fastq*

Run ‘*ustacks’* on the file *all\_u.fq* using the script:

*stacks/1.44/bin/ustacks -t fastq -f all\_u.fq -I 1 -m 1 -M 1 -N 1 -H --max\_locus\_stacks 2*

To test the pipeline (using demo files):

*$ cat 1M\_PM-KNE10.fq 1M\_PM-MD20.fq 1M\_PM-OK.fq > all\_u.fastq*

*$ ustacks -t fastq -f all\_u.fq -I 1 -m 1 -M 1 -N 1 -H --max\_locus\_stacks 2*

1. Generate a reference from file *all\_u.tags.tsv*. Copy the python script ‘*ASustacks\_ref.py*’ to the same folder as *all\_u.tags.tsv*. Run the script:

*python3 ASustacks\_ref.py*

The script will ask the following questions upon running: 1) full name of *all\_u.tags.tsv*; 2) the number of samples required for using a tag as reference; 3) the output file name. For example, if the output of step c has the file name *all\_u.tags.tsv*, the total sample number is 100, and a tag must exist in at least 20 samples to be included in the reference, then enter ‘*all\_u.tags.tsv*’ for question 1, and ‘*20*’ for question 2.

The script will output a fasta file; continue to step 10.

To test the pipeline (using demo files):

*$ python3 ASustacks\_ref.py*

When asked to 'Enter Ustacks output tags.tsv file name:', enter: *all\_u.tags.tsv*

When asked to 'Enter output reference fasta file name:', enter: ref.fa

When asked to ‘Enter the threshold number for reference’, enter: *1*

When asked to 'Enter output reference fasta file name:', enter: *ref.fa*

1. Filter reference to remove tags with 98% similarity.

In the same folder that contains *ref.fa*, run the scripts below:

*ncbiblast+/2.2.29/bin/makeblastdb -in ref.fa -dbtype nucl -title refdb -parse\_seqids -out refdb refdb.log*

*ncbiblast+/2.2.29/bin/blastn -db refdb -query ref.fa -evalue 1e-5 -outfmt 6 -max\_target\_seqs 2 -out ref.blast*

To test the pipeline (using demo files):

*ncbiblast+/2.2.29/bin/makeblastdb -in ref.fa -dbtype nucl -title refdb -parse\_seqids -out refdb refdb.log*

*ncbiblast+/2.2.29/bin/blastn -db refdb -query ref.fa -evalue 1e-5 -outfmt 6 -max\_target\_seqs 2 -out ref.blast*

Then, copy Ref\_98.py into the same folder and run the script:

*python3 Ref\_98.py*

The script asks for the reference sequence and reference blast result as inputs; enter the full name of *ref.fa* and *ref.blast*. The script’s output is the filtered reference *ref\_98.fa*. This is the reference to which the trimmed reads will be aligned.

To test the pipeline (using demo files):

*$ python3 ASustacks\_ref.py*

When asked to ‘Enter the reference fasta file:’, enter: *ref.fa*

When asked to ‘Enter the reference blast file:’, enter: *ref.blast*

**S3. Align GBS reads to reference and SNP calling**

1. Prepare a reference index for bowtie2 [4] and GATK [5]

Copy *ref\_98.fa* to the “BCpc” folder, and run the scripts below:

*bowtie2/2.2.9/bin/bowtie2-build ref\_98.fa ref\_98*

*java -jar picard/2.4.1/CreateSequenceDictionary.jar -R=ref\_98.fa O=ref\_98.dict*

*samtools faidx ref\_98.fa*

To test the pipeline (using demo files):

*$ bowtie2-build ref\_98.fa ref\_98*

*$ java -jar picard CreateSequenceDictionary.jar -R=ref\_98.fa O=ref\_98.dict*

*$ samtools faidx ref\_98.fa*

1. Align reads to the reference and process output for GATK

In “BCpc” folder, run the scripts below for each sample.

*bowtie2/bin/bowtie2 -x ref\_98 -1 sample1.1.fq -2 sample1.2.fq -S sample1.sam*

*samtools/1.3.1/samtools view -bS sample1.sam > sample1.bam*

*java -jar picard/2.4.1/ValidateSamFile.jar INPUT= sample1.bam*

*java -jar picard/2.4.1/AddOrReplaceReadGroups.jar I=sample1.bam O=sample1.Gr.bam LB=Whatever PL=Illumina PU=Whatever SM=sample1*

*samtools/1.3.1/samtools sort sample1.Gr.bam sample1.Gr.sorted*

*samtools/1.3.1/samtools index sample1.Gr.sorted.bam*

To test the pipeline (using demo files):

*$ mkdir SNP*

*$ cd ..*

*$ bowtie2 -x ASU/ref\_98 -1 BCpc/1M\_PM-KNE10.1.fq -2 BCpc/1M\_PM-KNE10.2.fq -S SNP/1M\_PM-KNE10.sam*

*$ cd SNP*

*$ samtools view -bS 1M\_PM-KNE10.sam > 1M\_PM-KNE10.bam*

*$ java -jar picard ValidateSamFile.jar INPUT= 1M\_PM-KNE10.bam*

*$ java -jar picard AddOrReplaceReadGroups.jar I=1M\_PM-KNE10.bam O=1M\_PM-KNE10.Gr.bam LB=Whatever PL=Illumina PU=Whatever SM=1M\_PM-KNE10*

*$ samtools sort 1M\_PM-KNE10.Gr.bam 1M\_PM-KNE10.Gr.sorted*

*$ samtools index 1M\_PM-KNE10.Gr.sorted.bam*

Do this for each of the three demo samples.

1. SNP calling with GATK

For the “Unified Genotyper” module, run the scripts below in the “BCpc” folder:

*java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T UnifiedGenotyper -R ref\_98.fa -I sample1. Gr.sorted.bam -I sample2.Gr.sorted.bam -I sample3.Gr.sorted.bam -dcov 1000 -glm BOTH -o raw\_SNPs\_unified.vcf*

To test the pipeline (using demo files):

*$ java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T UnifiedGenotyper -R ../ASU/ref\_98.fa -I 1M\_PM-KNE10. Gr.sorted.bam -I 1M\_PM-MD20.Gr.sorted.bam -I 1M\_PM-OK.Gr.sorted.bam -dcov 1000 -glm BOTH -o raw\_SNPs\_unified.vcf*

For the “Haplotype Caller” module:

1. Run the script below for each sample:

*java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T HaplotypeCaller -R ref\_98.fa -I sample1.Gr.sorted.bam --genotyping\_mode DISCOVERY -stand\_emit\_conf 10 -stand\_call\_conf 30 --minPruning 1 --emitRefConfidence GVCF -o sample1.g.vcf*

1. Then run the script below to get SNP data for all samples:

*java -jar gatk/3.4.0/GenomeAnalysisTk.jar -T GenotyperGVCFs -R ref\_98.fa --variant sampel1.g.vcf --variant sample2.g.vcf --variant sample3.g.vcf -o raw\_SNPs\_haplo.vcf*

1. Filter the GATK output

Select only biallelic SNPs and filter on allele frequency (AF). Run the scripts below in the “BCpc” folder.

*java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T SelectVariants -R ref\_98.fa -o biallelic\_raw\_SNPs\_unified.vcf --variant raw\_SNPs\_unified.vcf -restrictAllelesTo BIALLELIC*

*java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T SelectVariants -R ref\_98.fa -V biallelic\_raw\_SNPs\_unified.vcf --filterExpression “AF <= 0.100 || AF >= 0.900” --filterName “AF\_0.100\_0.900” -o AF\_biallelic\_raw\_SNPs\_unified.vcf*

To test the pipeline (using demo files):

*$ java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T SelectVariants -R ref\_98.fa -o biallelic\_raw\_SNPs\_unified.vcf --variant raw\_SNPs\_unified.vcf -restrictAllelesTo BIALLELIC*

*$ java -jar gatk/3.4.0/GenomeAnalysisTK.jar -T SelectVariants -R ref\_98.fa -V biallelic\_raw\_SNPs\_unified.vcf --filterExpression “AF <= 0.100 || AF >= 0.900” --filterName “AF\_0.100\_0.900” -o AF\_biallelic\_raw\_SNPs\_unified.vcf*

**S4. Further processing of SNP markers: SNP filtering, SNP consolidation and removal of cosegregating markers.**

1. Remove adjacent SNPs

Copy scripts “SNPs\_ISL.pl” and “Rm\_adj\_SNPs.pl” into the “BCpc” folder, and run the script below.

*perl SNPs\_ISL.pl input\_file\_vcf*

The following output file is generated: “more\_than\_one\_SNPs\_for\_same\_chromosome\_number\_in\_same\_line.vcf”

Run the second script on the output file:

*perl Rm\_adj\_SNPs.pl more\_than\_one\_SNPs\_for\_same\_chromosome\_number\_in\_same\_line.vcf*

1. Consolidate SNPs

Place the GATK output SNP .vcf file and the python script ‘*SNP\_genotyper.py*’ in the same folder.

To consolidate SNPs, run *‘SNP\_genotyper.py*’ by double clicking it (Windows/MacOS) or by typing ‘*python3 SNP\_genotyper.py*’ (Linux/UNIX). ‘*SNP\_genotyper.py*’ will translate the GATK results from read numbers to genotype (A, B, H, C, D or -), consolidate all SNPs within a GBS tag (if a GBS reference is used) or within a physical distance of 500 bp (if a genome assembly is used as reference), and give the consolidated SNP a unique name ‘Tag\_’ plus number. The output is a tab delimited text file (\*.txt) that can be opened with excel.

The ‘*SNP\_genotyper.py*’script will ask for the following questions upon running.

1. ‘Enter the GATK .vcf file name: ‘

Enter the full name (including extension name) of the GATK output vcf file

1. ‘Did you apply any GATK filters (AF/QD/..) on your results? If yes, enter ‘y’, else, enter ‘n’ (use lower case only): ‘

Enter ‘y’ or ‘n’ (in lower case)

1. ‘Enter the percentage of missing data you want to remove, integer only: ‘

For example, if the total accession number is 100, and the input number is 20, all SNP loci with missing data in 20 or more accessions will be removed.

1. ‘Enter the threshold for H (2/3/4) check instruction for detail):’

Enter an integer (suggested 2 to 4) to determine heterozygosity. The script takes allele depth (AD) from the GATK output file at each SNP locus for each accession as input to provide a genotypic score (A, B, H, C, D, -). The answer to this question only affects the definition of “H”, “C”, and “D”.

If 4 is used, AD is converted to A, B, H, C and D scores as follows:

ADref(erence allele)/ADalt(ernate allele) ratio ≥10: A (homozygous for the Parent 1 allele)

ADref/ADalt ≤0.10: B

10>ADref/ADalt>4: D (ambiguous A or heterozygous (H))

0.25>ADref/ADalt>0.1: C (ambiguous B or H).

All other ratios: H

If 3 is used, AD is converted to A, B, H, C and D scores as follows:

ADref(erence allele)/ADalt(ernate allele) ratio ≥10: A (homozygous for the Parent 1 allele)

ADref/ADalt ≤0.10: B

10>ADref/ADalt>3: D (ambiguous A or heterozygous (H))

0.33>ADref/ADalt>0.1: C (ambiguous B or H).

All other ratios: H

If 2 is used, AD is converted to A, B, H, C and D scores as follows:

ADref(erence allele)/ADalt(ernate allele) ratio ≥10: A (homozygous for the Parent 1 allele)

ADref/ADalt ≤0.10: B

10>ADref/ADalt>2: D (ambiguous A or heterozygous (H))

0.50>ADref/ADalt>0.1: C (ambiguous B or H).

All other ratios: H

Increasing the parameter from 2 to 4 increases the number of loci scored as “H” (definition range increases from ½-2 to ¼-4) and decreases the number of loci scored as “C” (definition range decreases from 1/10 - 1/2 to 1/10 -1/4) and “D” (definition range decreases from 2-10 to 4-10).

We recommend 4 as input.

1. 'Enter the threshold for sequence depth for robust SNP scoring (*e.g.* 8), integer only: '

If the number is set to 8, only SNPs with a read depth ≥8 will be scored. If read depth is <8, SNP scores will be listed as missing data (“-“). For populations containing heterozygotes, we recommend setting this number to 8.

1. Select SNPs based on parental scores (only applies to some mapping populations)

To select SNPs that are ‘A’ in parent 1 and ‘B’ in parent 2 (or *vice versa*), run ‘*SNP\_ selectByParents.py’* on the ‘*SNP\_genotyper.py*’ output file. The script will ask for the input file name, output filename, parents’ name, and parents’ genotype for selection.

1. Remove cosegregating SNP markers

Place the GATK output SNP .vcf file and the python script ‘*SNP\_cosegregation.py*’ in the same folder.

To remove cosegregating markers, run the script ‘*SNP\_cosegregation.py*’ in the same way as ‘*SNP\_genotyper.py*’. This code generates two output files; file 1 comprises the SNPs for downstream analyses), and the second file contains information on all cosegregating markers (file name is ‘info\_’ plus file 1 name). In the second output file, the cosegregating markers group together and groups are separated by a blank line. For each cosegregating group, the first marker is the best marker, and this marker will be represented in file 1.

This script can be applied after ‘*SNP\_genotyper.py*’, or it can be run independently from ‘*SNP\_genotyper.py*’. If run independently, the file input format must be:

1. Tab delimited file.
2. First row must be the sample name and contain the string “CHROM” anywhere you like.
3. Only A/B/C/D/H/- should be used in the data area, all other strings are not acceptable.

The script will ask the following questions upon running.

1. 'Enter the SNP file name (Tab delimited file \*.txt): '

Enter the input file name; usually this is the output file from ‘*SNP\_genotyper.py*’

1. ‘Enter the output file name: ‘

Enter a name for the output file; the script will generate two files, one will be the name you enter, the other will be ‘info\_’ plus this name.

1. ‘Enter the sample start column number (count start from 1, input 11 if applied on the output of ‘*SNP\_genotyper.py*’): ‘

If the input is the ‘*SNP\_genotyper.py*’ output, enter 11; else, count the column in which the genotypic data starts, starting from 1.

References:

1. Andrews S. FastQC: a quallity control tool for high throughput sequence data. 2010.

2. Catchen JM, Amores A, Hohenlohe P, Cresko W, Postlethwait JH. Stacks: building and genotyping loci *de novo* from short-read sequences. G3 (Bethesda, Md). 2011;1(3): 171-182. Epub 2012/03/03. doi: 10.1534/g3.111.000240. PubMed PMID: 22384329; PubMed Central PMCID: PMCPMC3276136.

3. Magoc T, Salzberg S. FLASH: Fast length adjustment of short reads to improve genome assemblies. Bioinformatics. 2011;27. Epub 2963.

4. Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nat Meth. 2012;9: 357-359.

5. McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, et al. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res. 2010;20(9): 1297-1303. doi: 10.1101/gr.107524.110. PubMed PMID: PMC2928508.