VERSION 2

JAN 11, 2024

OPEN ACCESS

DOI:
dx.doi.org/10.17504/protocols.io.dm6gp3m21vzp/v2

Protocol Citation: James Reeve, Amin Ghane, Pierre Barry, Alfonso Balmori-de la Puente, Roger K. Butlin, Le Qin Choo, Alan Le Moan, Diego Garica Castillo, Ana-Maria Peris Tamayo, Sarah Kingston, Erica Leder, Sean Stankowski 2024. A standard pipeline for processing short-read sequencing data from Littorina snails. **protocols.io** https://dx.doi.org/10.17504/protocols.io.dm6gp3m21vzp/v2Version created by James Reeve

A standard pipeline for processing short-read sequencing data from Littorina snails V.2

James Reeve[1], Amin Ghane[2], Pierre Barry[3], Alfonso Balmori-de la Puente[3,4], Roger K. Butlin[5,1], Le Qin Choo[5], Alan Le Moan[6], Diego Garica Castillo[7], Ana-Maria Peris Tamayo[8], Sarah Kingston[9], Erica Leder[1], Sean Stankowski[7,10]

[1]Tjärnö Marine Laboratory, University of Gothenburg, 452 96 Strömstad, Sweden;
[2]Department of Botany and Biodiversity Research, University of Vienna, Vienna, Austria;
[3]BIPOLIS, CIBIO, University of Porto, 4485-661 Vairão, Portugal;
[4]Universitat de Barcelona, 08028 Barcelona, Spain;
[5]School of Biological Sciences, University of Sheffield, Sheffield S10 2TN, UK;
[6]Roscoff Marine Station, Sorbonne University, 29680 Roscoff, France;
[7]ISTA, 3400 Klosterneuberg, Austria;
[8]Faculty of Biosciences and Aquaculture, Nord University, 8049 Bodø, Norway;
[9]Sea Education Association, Woods Hole, MA 02543, USA;
[10]Department of Ecology and Evolution, University of Sussex, Brighton BN1 9RH, UK

James Reeve: Corresponding author;

James Reeve
Tjärnö Marine Laboratory, University of Gothenburg, 452 96 S...

ABSTRACT

This protocol explains the default pipeline we recommend for analysing short-read data for Littorina marine snails. The pipeline was designed based on our experience processing WGS data, and careful consideration of the available tools. We are presenting this protocol as a suggestion of how Littorina sequencing data can be processed, and as a template for building more specific pipelines. Our hope is that through writing this protocol we can have more consistency among projects, leading to greater transparency for collaborators and providing a template which could be used to build pipelines for other organisms.

Created: Jan 03, 2024

Last Modified: Jan 11, 2024

PROTOCOL integer ID: 92904

Keywords: bioinformatics, biology, genetics, sequencing, Littorina

Funders Acknowledgement:
Swedish Research Council
Grant ID: 2018-03695_VR

ATTACHMENTS

LitSD_pipeline_protocol_v1.pdf

GUIDELINES

This protocol was created as a guide for new users. We strongly recommend exploring your data and reading the manuals for the software we present, before designing your own sequence data pipeline.

Overview

1 Processing of short read sequencing data can be broken down into four major steps (Fig. 1):

1) read quality inspection and trimming
2) read mapping
3) variant calling
4) variant filtering

There are numerous sub-steps in between, which together have a plethora of different adjustments one could make. Without a reference, two users could create vastly different pipelines for the same data. Below, we outline our recommended pipeline steps, briefly describing the process, showing example code, and explaining the different options available.

Speeding things up [optional]:

Our recommendations are not the most optimised pipeline. We purposely go into detail for each step to explain the process. Once you are familiar with this pipeline, you can pipe commands together to skip generating intermediate files. Large files (e.g. **fastq**, **BAM** and **VCF**) can also be compressed to reduce storage space, which will speed up the analysis.

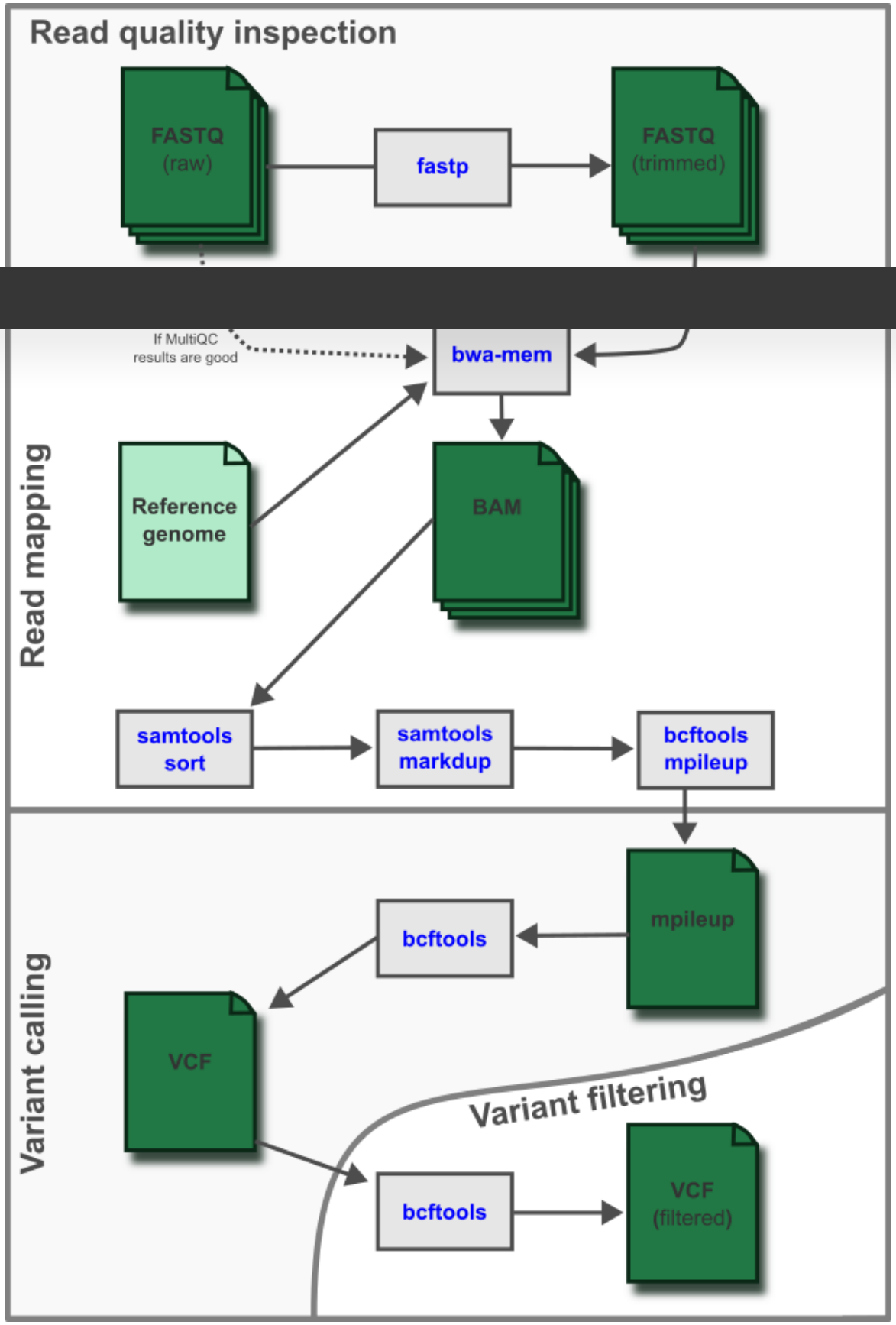

Figure 1: flowchart of steps in the *Littorina* variant calling pipeline. See Table S1 for definitions.

sequencers output raw data in **fastq** format. This contains nucleotides along with a Phred-likelihood quality score for each base (the probability that the called base is wrong). Bases at the end of reads often have lower quality than those at the start. Trimming these poor quality bases is recommended using a tool like **fastp**. (Del Fabbro 2013, Chen et al. 2018). Base quality is not the only metric that can be used to evaluate reads; other important statistics include read length, GC content, duplications and adaptor content (HBCtraining, n.d.). We recommend using **fastqc** to generate summary reports of read quality. For datasets of many individuals, these reports can be merged with **MultiQC**.

3 **1.1) Read quality:**

Before trimming and mapping, our reads are evaluated with **fastqc** per individual and **MultiQC** for the whole dataset. We run **fastqc** with default settings, apart from -t which specifies the number of threads (-t will depend on your computer specifications). The resulting quality reports are composed of several sections (e.g. read length, nucleotide content, adaptors, duplication). Detailed example reports and a tutorial video are on the **fastqc** website. We recommend looking at these before deciding how to trim reads.

```
# fastqc
fastqc --dir /path/temp_files -t 20 \
 /path/file.fastq.gz \
 --outdir /path/output/

# MultiQC
multiqc /path/fastqc/dir \
 -o /path/output/dir \
 -n output.name.prefix

# Note: change '/path/' to your own file directory path.
```

4 **1.2) Read trimming:**

We recommend the program **fastp** for trimming adaptors and removing low quality ends. This program can identify adaptor sequences (assuming standard adaptors were used by the sequencing centre), and other known errors like polyG tails, and automatically trim them from the data. There are also options to specify adaptor sequences manually and the number of base pairs to clip, if the default settings do not work with the data. **fastp** also generates a quality report which can be compared with the **fastqc** reports to see if the trimming is successful.

For paired-end sequencing (Illumina, 2023) you must specify both a forward (-i) and reverse (-I) read files. By default **fastp** overwrites existing files, we set -o and -O to specify the output. Optional filtering steps we use are -g to trim polyG tails, -c to perform base correction in overlapping paired data, and -y as a complexity filter that removes reads with < 30% complexity (shifts between neighbouring base pairs; basei ≠ basei+1). Finally, --html, --json, and --report_title are set to specify where the output reports are placed.

```
fastp \
      -i /path/input_R1.fastq.gz \ # input files
      -I /path/input_R2.fastq.gz \
 -o /path/output_R1.fastq.gz \ # output files
      -O /path/output_R2.fastq.gz \
 --thread 20 -g -c -y 30 \
 --html /path/output_report.html \ # quality reports
 --json /path/output_report.json \
 --report_title report.title.prefix
```

It is a good idea to inspect the reports at this point to determine if read trimming has improved the quality (see fastqc tutorials). If so, move onto read mapping. If not, the **fastp** options need tweaking until the reads are ready to map.

Read mapping

5 Once we have good quality reads, we next map them onto the Littorina saxatilis reference genome (De Jode et al., in review). At the simplest level, read mapping is just aligning DNA sequences. The complexity comes in when trying to optimise this process for next generation sequencing data (Trapnell and Salzberg 2010). We recommend using the Burrows-Wheeler Aligner (**BWA**). Other read callers can do a similar job, but **BWA** has the highest accuracy in benchmarking papers (Musich et al. 2021, Zanti et al. 2021, Schilbert et al. 2020). **BWA** takes two **fastq** files as input per sample, the forward and reverse reads. If sequencing data are paired-end, then **BWA** outputs a single **SAM** file. This output adds additional information to the **fastq** which shows where each read is placed on the reference genome and if there are any mismatches. **SAM** files tend to be massive, so as a final step we compress them to **BAM** format.

6 **2.1) Index reference genome:**
Before running **BWA** we need to index the reference genome. This helps the algorithm to find shorter sequences throughout the genome, speeding up the whole process (Trapnell and Salzberg 2010). We index the reference genome with the simple one-liner below. This only has to be done once, but remember to keep the index files in the same directory as the reference genome.

```
bwa index /path/reference.fa
```

7 **2.2) Mapping reads with BWA:**
Reads are mapped to the reference genome using **BWA**. We use the maximal exact matches (mem) algorithm. Three input files are required: the reference genome, the forward reads **fastq** and the reverse reads **fastq**. We optionally set -M to mark shorter splits as secondary alignments (i.e. other parts of the genome with lower quality mapping), -R to specify the read group information (flags that indicate how a

read was sequenced and the sample it came from), and -t to set the number of threads to run in parallel. Lastly, the standard output is piped into **samtools** to write the file in the compressed **BAM** format.

```
bwa mem \
     -M -t 20\
     -R '@RG\tID:1\tSM:{sample.name}\tPL:ILLUMINA\tLB:lin\tPU:
{platform.name}' \
     /path/reference.fa \ # reference genome
     /path/input_R1.fastq.qz \ # trimmed forward reads
     /path/input_R2.fastq.gz |\ # trimmed reverse reads
  samtools view -b > /path/output.bam
```

8 2.3) Post mapping read data sorting:

After mapping reads, some additional intermediate steps are needed before we can start variant calling. These steps vary depending on the type of analysis conducted and which software are used. For this default pipeline, we follow the guidelines from the **samtools** website (http://www.htslib.org/algorithms/duplicate.html). Our intermediate steps are i) reordering reads in the **BAM**, ii) identifying possible duplicated reads, and iii) creating a summary report.

8.1 2.3.1) Sort BAM:

Sorting steps are all conducted using **samtools**. Reads are sorted in two ways, first alphabetically by read name using the -n option, then by genome position with the default sort command. In between these sorting steps, we run the fixmate to fill in missing details among paired reads, with the -m option adding a mate-score tag to the **BAM**. While this does not seem very important, this step is necessary when we identify duplicates. Each of these **samtools** commands is piped into a single line of code, to avoid creating a massive number of intermediate files.

```
samtools collate -@ 20 -Ou /path/input.bam |\ # sort by read names
samtools fixmate -@ 20 -m - - |\ # fix mate-pair information
samtools sort -@ 20 - -o /path/output.bam - # sort by positions
```

The **samtools** -@ option sets the number of threads to run, and the - are placeholders for the input and output files. These placeholders are necessary to pipe different **samtools** commands together.

8.2 2.3.2) Mark duplicates:

Duplicated reads are identified in the sorted **BAM** with **samtools markdup**. PCR duplicates can occur during the amplification steps of library preparation. Another type of duplication, optical duplication, originates from technical artefacts in the clustering on the Illumina flowcell and these are not identified by default. **samtools markdup** uses coordinate and tile information from the read name to mark optical duplicates, using the -d option to set the distance among coordinates. Both types of

duplicates can be removed with the -r option, but this is often not necessary as reads marked as duplicates are not considered by most variant callers.

```
samtools markdup -@ 20 -d 100 /path/input.bam /path/output.bam
```

8.3 **2.3.3) Index and generate flagstat report:**

Duplicate marked **BAM** files are indexed with **samtools index** and a summary table of **BAM** flags is generated with **samtools flagstat**. Neither step is necessary for our variant calling, but index files are required for several programs that take **BAM** inputs, and the flagstats reports are a convenient way to assess overall mapping quality (e.g. the percentage of mapped reads, the number of duplicates, mappings to more than one location). See table 1 for a comparison of mapping quality among different Littorina species.

```
samtools index -b /path/input.bam
samtools flagstat /path/input.bam > /path/output.flagstat
```

Species	Mapped	Primary	Paired	N
L. saxatilis	98.77%	75.35%	53.18%	4
L. obtusata	97.61%	81.39%	73.72%	14
L. fabilis	97.91%	80.38%	71.91%	432
L. compressa	98.60%	92.39%	80.02%	8
L. arcana	98.69%	93.00%	82.15%	8
L. saxatilis	98.77%	93.15%	82.82%	137

Table 1: Example mapping statistics for different species in the genus Littorina, mapped to the L. saxatilis v.2 reference genome (De Jode, et al., in review). Values are the average percentages across **N** samples. **Mapped** indicates the percentage of reads aligned to the reference genome. **Primary** excludes lower quality reads mapped to multiple genomic positions. **Paired** represents both the forward and reverse reads mapping to the same genomic region.

Variant calling

9 Variant calling sifts through multiple BAM files to identify genomic positions that vary among individuals. These variants can include indels, SNPs, SNVs, or repeated elements. The most common variants used in evolutionary research are SNPs. SNP calling identifies positions where two (or more) versions of a nucleotide are present in a population. Each individual is also assigned a genotype, often scored as ‘0/0’

if homozygous for the reference allele (i.e., the individual carries two alleles that match the allele in the reference genome), '1/1' if homozygous for the alternative allele, and '0/1' if heterozygous. Individuals with no called genotype at this position are scored as './.'.

Genotypes can be called with a range of different models (one of the key distinctions among different calling software packages) that give each individual three probability scores (one for each possible genotype) at each biallelic site. In addition, other metrics, such as depth of coverage (i.e. how many reads are used), mapping quality and strand biases, are provided. All of this information is stored in a variant call format (**VCF**) file.

We recommend calling variants using the program **bcftools**. It performed quicker and as accurately (evaluated with precision and recall) as a more popular caller (GATK; https://gatk.broadinstitute.org/hc/en-us), when tested on sequence data from a set of Littorina parent-offspring trios. There are two steps to calling with **bcftools**; first, sequences from each **BAM** file are concatenated together into an **mpileup** file.

9.1 3.1) mpileup:

This step estimates a genotype likelihood for each variant. **mpileup** files can take some time to generate. Splitting the genome into regions with -r CHROM:POS, speeds up the process by parallelising the job. Splitting is done by chromosome, splitting some of the larger chromosomes in half. The -a option tells **bcftools** to annotate the **mpileup** with additional quality scores. We add allele depth (AD), total depth (DP) and strand bias (SP) per site to the FORMAT field, and the allelic depth (AD) per individual to the INFO field.

```
bcftools mpileup \
 --threads 20 \
      -a FORMAT/AD,FORMAT/DP,FORMAT/SP,INFO/AD \ # specify tags
 -f /path/reference.fa \ # reference genome
 -b /path/list.of.bams \
 -r CHROM:POS |\ # optional for parallelising
```

9.2 3.2) Variant call:

The next step is to call variant sites with **bcftools call**. The -m option activates the multiallelic calling mode, -Oz compresses the output, and -f GQ,GP includes genotype quality (GQ) and genotype probability (GP) in the output. We recommend calling both variant and invariant sites (of sufficient quality), as invariant positions are needed to correctly calculate many population genetic statistics (e.g. π and dxy).

```
bcftools call \
 --threads 20 \
 -f GQ,GP \
 -m -Oz \ # add -v to drop invariant sites
 -o /path/output.vcf.gz
```

There are a few options to consider when calling variants. Downstream processes can be sped up by changing the output to be in binary call format (**BCF**) with -Ob. However, this will increase the file size. If space is limited, set -v to remove invariant sites from the output, but this limits the potential for some downstream analysis.

Variant filtering

10 After calling, variants must be filtered to remove possible poorly sequenced, low quality sites and undesired types of variant (e.g. indels). This step will vary most between studies, as filtering thresholds depend heavily on the type of data, sequencing approach and end results of the analysis. The recommendations we give below are intended to provide some advice for new users, as a benchmark for some of the most common filters. Depending on the downstream analysis and other factors like sample number and sequencing depth, some criteria might need tweaking. For instance, for low coverage data a lot of information may be lost with the filters proposed here. On the other hand, for very precise analysis, even more stringent filters are required. Ultimately, judgement is left up to the user; we recommend extracting variant quality scores (e.g. DP, SP, MQ, or QUAL) from the **VCF** and plotting their distributions before choosing filtering thresholds.

Main steps in filtering:
It is best to think about filtering as three steps.

1. Hard filtering: the removal of entire variants, purging their record from the **VCF** file
2. Soft filtering: the masking of individual genotypes from the file by setting these as missing data
3. Filter by the proportion of missing data: removing entire sites where fewer than x% of individuals have a genotype

A note on applying filters:
It is tempting to apply filters in a single command which removes sites or masks genotypes based on multiple criteria simultaneously. Although this may seem more efficient, it is difficult to check that filters are applied as expected. It is much easier to understand the effect of each filter if they are run separately, or in small sets. The number of sites left after each filter (see code below) should be recorded to keep track of data loss over the entire filtering process . Be sure not to overwrite your original file so you can always go back and filter the dataset based on a different set of criteria if needed.

```
# Counting SNPs in VCF
zcat /path/output.vcf.gz | awk ´!/\#/' | wc -l
# OsX terminal uses gzcat
```

11 **4.1) Hard filters:**

The goal is to delete entire positions from the **VCF** based on criteria that suggest the site was poorly sequenced or problematic. Hard filters can also be used to remove types of variants that are not of interest, such as indels or multiallelic sites. Below are some commands for filters that are commonly applied, but this list is not exhaustive. Different variant callers also compute different quality statistics, so be aware that the names and possible options vary if you are using a different caller (e.g., **GATK**). Short descriptions of the quality statistics should be listed in the **VCF** header.

11.1 **4.1.1) Remove indels and multallelic sites:**

Firstly, we remove types of variants which are not interesting for our research. Typically for WGS analysis, indels and multiallelic variants are removed, but there is some evidence that they may contribute to evolutionary processes (Perini 2021). We opted for not just removing indels, but also SNPs within 5bp of them as these sites tend to be problematic due to errors in the realignment process. The -g 5:indel,other option removes sites around indels, while the -v snps keeps only SNPs. We also remove multiallelic sites by setting -M 2. Specifically, this option caps the number of alleles per site to two. Optionally, a minimum can be set with -m 2 to remove invariant sites.

```
bcftools filter -Ou -g 5:indel,other /path/output.vcf.gz |\
bcftools view -Oz -M 2 -v snps > /path/SNP_only.vcf.gz
# Add -m 2 to last line to remove invariant sites
```

11.2 **4.1.2) Total read depth per site:**

Next we filter by total read depth per site (INFO/DP). This is the sum of reads across all samples. Only an upper limit is generally needed, as low read depth will be filtered by other steps (see soft filters). When depth is too high we recommend removing the whole site, as these can represent repeated sequences and misalignments of the reference genome. A common criterion is to drop sites with more than 2.5x the average coverage.

```
bcftools filter -Oz -e 'INFO/DP>2.5*AVG(INFO/DP)' /path/SNP_only.vcf.gz
> /path/DP_filt.vcf.gz
```

11.3 **4.1.2b) Setting a minimum total depth filter [conditional]**

For high coverage calls including invariant sites you can set a lower limit to remove sites where there is a risk of calling an invariant site by mistake. We recommend dropping sites with fewer reads than 14 x the number of samples. The parameter 14 is based on the distribution of variant and invariant sites in L. saxatilis data (Stankowski et al. 2024).

Again, do not set the lower INFO/DP filter for low coverage data. This will throw away too many good quality sites. We do not want to toss the baby with the bath water.

11.4 **4.1.3) mapping quality, genotype quality and strand-bias:**

Three more steps are involved in hard filtering, mapping quality (MQ), site call quality (QUAL) and strand bias (SP) filters. All three use Phred quality scores, which correspond to the probability of an error. The Phred scale is logarithmic, so larger scores indicate a lower probability of error. For example, a Phred-score of 20 indicates a 1% change of an error and a Phred-score of 30 indicates a 0.1% chance.

The first Phred-score is MQ, the average mapping quality at a site. Typically this should be set high to drop poorly mapped positions. Values between 30 and 40 are common.

```
bcftools filter -Oz -e 'MQ<30' /path/DP_filt.vcf.gz >
/path/MQ_filt.vcf.gz
```

QUAL is the probability that the alternative allele call was wrong. A score of 30 is common.

```
bcftools filter -Oz -e 'QUAL<30' /path/MQ_filt.vcf.gz >
/path/QUAL_filt.vcf.gz
```

Lastly, SP is the probability that one DNA strand is sequenced at a higher frequency than the other. Dropping SP > 3 is recommended as a light filter. Your threshold should be based on the distribution of scores (Fig. 2).

```
bcftools filter -Oz -e 'SP>3' /path/QUAL_filt.vcf.gz >
/path/SP_filt.vcf.gz
```

12 **4.2) Soft filters:**

After we have arrived at a set of variants that we feel are of adequate quality, we will now soft filter individual genotypes to mask those that we do not trust. Soft filters will set genotypes that do not pass our filters to missing (./.) while genotypes that pass the filters remain unaffected.

For **bcftools** the two soft filtered statistics are the genotype quality score (GQ) and depth of coverage (FMT/DP) per sample. GQ is Phred scored, thus GQ of 20 indicates a 1% chance that the call is incorrect for that sample. A GQ score of 30 is considered the gold standard, but might be overly conservative with low coverage data.

FMT/DP is the depth from the format field of the **VCF** (i.e. the coverage for each sample). At least 2 reads need to be present to call a heterozygote in diploid data. However, at FMT/DP = 2 there is only a 50% chance that we have seen both alleles. 5 reads gives us a 93.8% chance. With 10 reads this goes up to 99.8%, but if the threshold is above the targeted coverage most sites will be dropped. In short, the

higher thresholds are more accurate, but the appropriate choice depends on sequencing depth.

```
bcftools filter -Oz -S . -e 'FMT/GQ<20 | FMT/DP<5' /path/SP_filt.vcf.gz >
/path/soft_filt.vcf.gz
```

13 **4.3) Filter out missing data:**

After setting low quality genotypes to missing, we need to remove sites with high missing data. The percentage of missing data at which we remove sites is up to the user to decide. A simple rule is that with more samples more missing data can be tolerated. Filters that remove sites with more than 10-20% missing genotypes are common.

```
bcftools filter -Oz -e 'F_MISSING>0.1' /path/soft_filt.vcf.gz >
/path/filtered.vcf.gz
```

14 **4.4) How do I choose filtering thresholds for my data?**

Before and after applying a filter it is good practice to plot the distributions of key **VCF** statistics. There are no strict rules for choosing filters, only guidelines. Examples for some Littorina datasets are provided in Table 2. Do not blindly follow our example filters (Table 2). They are a starting point, to get a feel for how scores are distributed. We strongly recommend looking at the quality scores of each **VCF** before filtering. One must use common sense, logic and the shapes of the distributions to decide the best thresholds. After filtering, it is useful to replot these distributions to ensure that the filter worked as expected.

Type	INFO/DP	MQ	QUAL	SP	GQ	FMT/DP	FMISS
Low coverage WGS (3X)	-	< 20	< 30	> 3	< 20	< 1 | >18	> 20%
High coverage WGS (20X)	> 2xMDP	< 40	< 30	> 3	< 20	< 5	>10%

Table 2: Example thresholds for Littorina data with low and high coverage. Hard filters: **INFO/DP** = total read coverage across all samples. With MDP = average depth score. **MQ** = phred-scaled mapping quality. **QUAL** = phred-scaled variant call quality. **SP** = phred-scaled probability of strand bias. Soft filters: **FMT/DP** = read depth per genotype. **GQ** = phred-scaled genotype quality. **FMISS** = proportion of samples missing per site.

Quality scores can be extracted from a **VCF** with **bcftools query**. The results can then be plotted in **R**, or any other data analysis program. Trying this for the full data set is very time consuming as there is a lot of data to view. To speed things up, we recommend randomly downsampling a small proportion of SNPs using **vcfrandomsample** from the **vcflib** package (Garrison et al. 2022). For information on filtering invariant positions, see the supplementary information in Stankowski et al. 2024.

```
# Randomly sample 1% of SNPs
bcftools view filtered.vcf.gz | vcfrandomsample -r 0.01 > subset.vcf
```

bcftools query uses the -f option to query. This is followed by a string expression in quotation marks. See the full **bcftools** manual for details. We are extracting the chromosome name (CHROM), site position (POS) and the focal score. This last value can be set to score in the **VCF**. By default the score for each site (often site average) is extracted. Place square brackets around the value to get the score for each genotype.

```
# Total read depth per site [INFO/DP]
bcftools query -f '%CHROM\t%POS\t%DP\t]\n' subset.vcf >
output/vcfstats.tDP.txt
```

```
# Average map quality per site [MQ]
bcftools query -f '%CHROM\t%POS\t%MQ\n' subset.vcf >
output/vcfstats.MQ.txt
```

```
# Call quality per site [QUAL]
bcftools query -f '%CHROM\t%POS\t%QUAL\n' subset.vcf  >
output/vcfstats.QUAL.txt
```

```
# Read depth per sample [FMT/DP]
bcftools query -f '%CHROM\t%POS\t[%DP\t]\n' subset.vcf >
output/vcfstats.sDP.txt
```

```
# Genotype quality per sample (Phred-score) [GQ]
bcftools query -f '%CHROM\t%POS\t[%GQ\t]\n' subset.vcf >
output/vcfstats.GQ.txt
```

For strand-bias we are looking at the average score per site. While we could download all scores then calculate the average, it is much faster to calculate the average beforehand. Below, we use the **awk** command to directly calculate the row averages from the **bcftools query** output.

```
# Strand-bias P-value (Phred-score) [SP]
bcftools query -f '%CHROM\t%POS\t[%SP\t]\n' subset.vcf |\
       awk 'BEGIN{OFS = "\t"}{sum = 0; for (i = 3; i <= NF; i++) sum +=
$i; sum /= NF; print $1,$2,sum}' > output/vcfstats.SP.txt
```

Our **VCF** does not automatically include the fraction of missing genotypes per site (FMISS). FMISS is added using the +fill-tags plugin. This plugin was introduced in version 1.2 of **bcftools**.

```
# Missing data [FMISS]
bcftools +fill-tags subset.vcf -- -t 'FMISS=F_MISSING' | bcftools query -
f '%CHROM\t%POS\t%FMISS\n' > output/vcfstats.MISS.txt
```

Since we are extracting scores, it is a good idea to also look at the distribution of allele frequencies. Although we will not filter these scores, the distribution can indicate possible problems with the data.

```
# Allele frequency [AF]
bcftools +fill-tags subset.vcf -- -t AF | bcftools query -f
'%CHROM\t%POS\t%AF\n' > vcfstats.AF.txt
```

Once quality scores are extracted you can plot them to determine your filtering thresholds. These can then be compared to the values after filtering (Fig. 2). We show how to do this for total site depth in **R** script.

```
<<< R script >>>
library("ggplot2")
# Load data into R
vcfstat <- read.table("filepath/vcfstats.tDP.txt",
 col.names = c("CHROM", "POS", "STAT"))
# Plot distribution curve
ggplot(vcfstat, aes(STAT))+
 geom_density(colour = "red")+
 labs(x = "Depth of coverage for each site", y = "Density")+
 theme_classic()
```

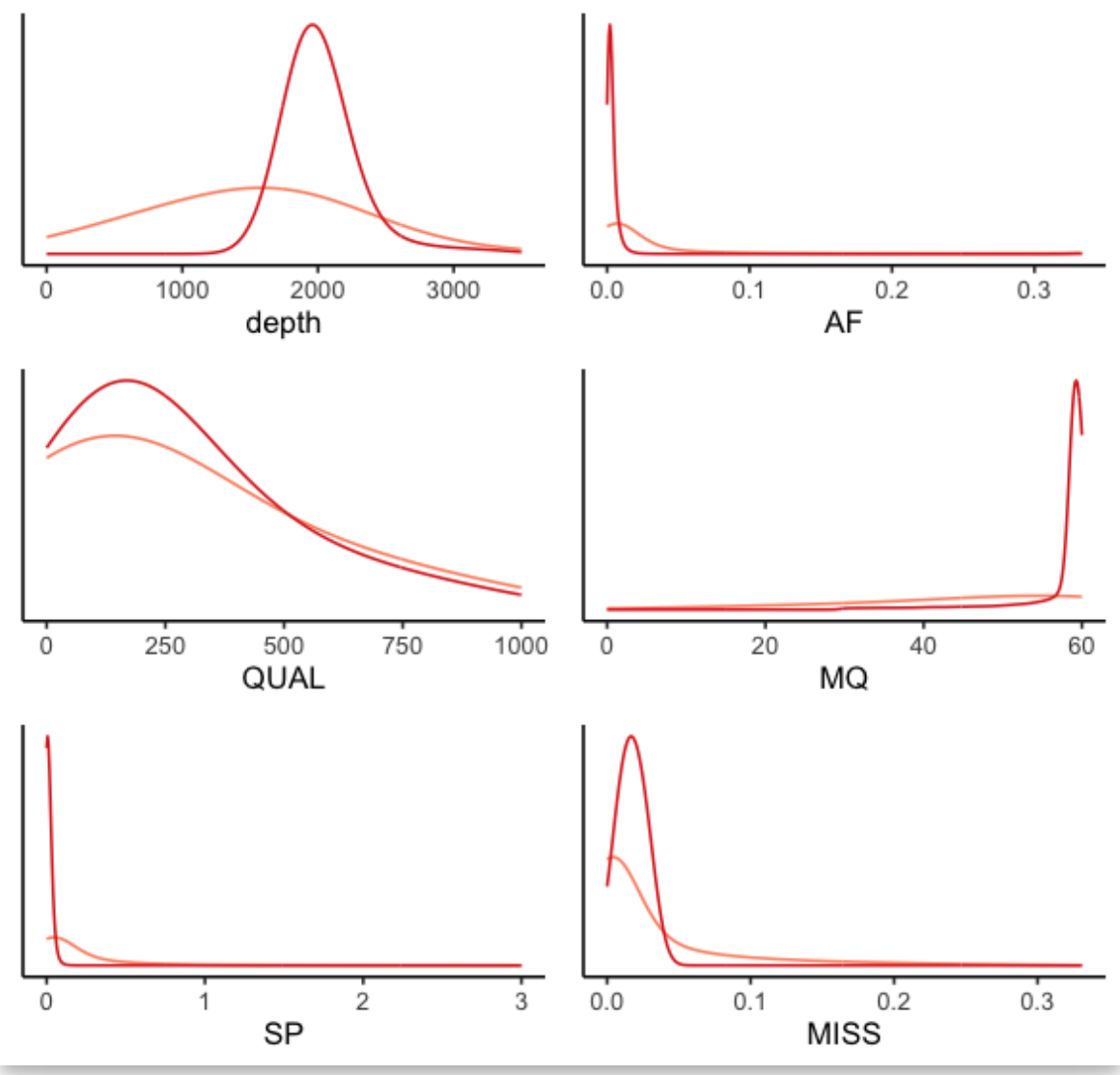

Figure 2: density curves for VCF statistics before (light red) and after (dark red) filtering. **DP** = total read depth per site. **AF** = frequency of the alternative allele. **QUAL** = Phred scaled variant call quality score. **MQ** = average Phred-scaled mapping quality. **SP** = Phred-scaled probability of strand bias. **MISS** = fraction of samples with missing alleles per SNP. Filtering thresholds are from our high coverage example (Table 2).

Appendix 1: adjustments for different datasets

15 Our default pipeline is developed for whole genome shotgun reads, with the goal of producing a set of SNP genotypes for analysis, as this is the most common approach used in evolutionary genomics research. However, other types of sequencing data require modifications to our pipeline. Here we give general advice about how our pipeline can be modified for three common types of data: RAD-seq, PoolSeq, and Haplotagging.

16 **RAD-seq:**

Restriction-site-associated DNA (RAD) sequencing is an older approach still widely used in population genetic studies (Puritz et al. 2014). It breaks DNA at specific restriction sites resulting in several thousand short reads per run. It is commonly used as a cheaper alternative to WGS. RAD-seq fragments the genome at restriction sites using specific adaptor sequences. Trimming RAD-reads requires special handling of these adaptors. We recommend ignoring adaptors in **fastp** (option '-A') and processing RAD-seq data with an established pipeline (e.g. stacks, Rochette et al., 2019).

17 **PoolSeq:**

Pooled sequencing is a common approach for analysing allele frequencies among populations on a budget (Schlötterer et al. 2014). DNA samples are pooled together in even proportions across a population to create a single pooled sample, which is then sequenced at high coverage. A single pair of **fastq** files is generated for each population.

PoolSeq and WGS-seq pipelines are similar. The major difference is that variant calling requires a program which can handle high "ploidy" (e.g. N ≥ 50). **PoPoolation** (Kofler et al. 2011) was specially designed to handle PoolSeq data. Filtering options are more limited with **PoPoolation**, as it only retains the statistics added to the **mpileup**. However, many of the filtering steps we have mentioned for **VCFs** can be applied to the **mpileup** and **BAM** files (see example in Morales et al. 2019).

18 **Haplotagging:**

Haplotagging is a new method of sequencing that keeps linkage information among reads (Meier et al. 2021). DNA libraries are prepared using specialised beads marking long DNA fragments with barcodes. Barcoded molecules are then sheared and washed off the beads and amplified with PCR, before standard Illumina short-read sequencing.

SamHaplotag is a pipeline specifically made to process haplotagged data (evolgenomics, 2022). In demultiplexing, the barcodes are encoded within the comments fields, using the 10xspoof.pl script.The reads are then trimmed (if necessary) and, the haplotag barcodes are converted using the 16BaseBCGen programme into generic 16-base barcodes with 7-base joins which can then be used as input for read mapping. We recommend using the **EMA** read mapper (Shajii et al. 2017) instead of **BWA**, to keep barcode information, which improves downstream genotyping accuracy, phasing, and structural variant detection. Variant calling and filtering can proceed with the recommendations of our pipeline.

Appendix 2: glossary

19

Software	Function	Version	Link to manual
fastqc	Summary report of fastq quality	0.11.9	https://github.com/s-andrews/FastQC
MultiQC	Merging fastqc reports across the whole dataset	1.12	https://multiqc.info/docs/
fastp	Trimming fastq files	0.23.4	https://github.com/OpenGene/fastp
BWA	Read mapping	0.7.17	https://bio-bwa.sourceforge.net/bwa.shtml
samtools	Processing and summarising BAM files	1.17	http://www.htslib.org/doc/samtools.html
bcftools	Merging BAM files and variant calling	1.17	https://samtools.github.io/bcftools/bcftools.html#mpileup
vcftools	Variant filtering	0.1.16	https://vcftools.sourceforge.net/man_latest.html
vcflib	Parsing and manipulating VCFs	1.0.9	https://github.com/vcflib/vcflib/tree/master

File formats:

Name	Description	More details
fastq	Sequence data file with sequencing quality scores	https://en.wikipedia.org/wiki/FASTQ_format
BAM	[B]inary [A]lignment [M]ap file	https://samtools.github.io/hts-specs/SAMv1.pdf
mpileup	Table of "piled up" reads at the same genetic position. Samples are written in separate columns	https://en.wikipedia.org/wiki/Pileup_format
VCF	[V]ariant [C]all [F]ormat file	https://samtools.github.io/hts-specs/VCFv4.2.pdf

Table S1: glossary of software and file formats. Check the version of your software before running this pipeline. Newer versions may have different options specified in their manual.