Nov 07, 2024 Version 3

# A standard pipeline for processing short-read sequencing data from Littorina snails V.3

DOI
https://dx.doi.org/10.17504/protocols.io.dm6gp3m21vzp/v3

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], Basile Pajot[11]

[1]Tjärnö Marine Laboratory, University of Gothenburg, 452 96 Strömstad, Sweden;
[2]Department of Botany and Biodiversity Research, University of Vienna;
[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;
[11]Sorbonne Université, CNRS, Adaptation et Diversité en Milieu Marin, AD2M, F-29680 Roscoff, France

James Reeve: Corresponding author;

James Reeve
Danish Technical University



OPEN ACCESS

**DOI:** https://dx.doi.org/10.17504/protocols.io.dm6gp3m21vzp/v3

**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, Basile Pajot 2024. A standard pipeline for processing short-read sequencing data from Littorina snails. **protocols.io** https://dx.doi.org/10.17504/protocols.io.dm6gp3m21vzp/v3Version created by James Reeve



**Protocol status:** Working
**We use this protocol and it's working**

**Created:** November 07, 2024

**Last Modified:** November 07, 2024

**Protocol Integer ID:** 111737

**Keywords:** bioinformatics, biology, genetics, sequencing, Littorina, data from littorina snails short, processing sequence data in bioinformatic pipeline, read data from marine snail, sequencing data, genome sequencing, bioinformatic pipeline, littorina snails short, pipelines for other organism, marine snail, discovery pipeline, raw reads to variant, genome, processing sequence data, specific pipeline, genus littorina, pipeline, raw read, default pipeline, standard pipeline, underpinning evolution, other organism, read data, read sequencing, sequencing, organism

**Funders Acknowledgements:**
**Swedish Research Council**
Grant ID: 2018-03695_VR

## Abstract

Short-read whole-genome sequencing helps us to understand the genetic mechanisms underpinning evolution. To go from raw reads to variants requires a bioinformatic pipeline. Processing sequence data in bioinformatic pipelines is intricate, with several intermediate steps that have a plethora of parameters and options. A default variant-discovery pipeline saves time and effort and makes diverse studies more comparable.

This protocol explains the default pipeline we recommend for analysing short-read data from marine snails in the genus Littorina. 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, which can be used as a template for building more specific pipelines. For more general information about filtering next generation sequencing data see Hemstrom et al. (2024). 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.

## Attachments


LSRP_v3_2024_11_07.p...

735KB

## 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 template, 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.

A tutorial of this protocol working with a single L. saxatilis from Sweden (NCBI SRA: SAMN37254815) can be found on GitHub repository for this project: https://github.com/ja-Reeve/Littorina_SeqDat_pipeline
The protocol, sans variant filtering, has also been implemented into a snakemake command found at: https://github.com/PAJOT-Basile/Snakemake_processing_of_short-read_data

**Speeding things up [optional]:**
Our recommendations are not the most optimised pipeline. We have purposely decided to go into extra detail for each step to explain the process behind making a pipeline. 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. Finally, for some of the larger jobs, like read mapping and variant calling, commands can be run in parallel over multiple computer cores. An example of parallelizing variant calling is shown in step 9.

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

## Read quality inspection and trimming

2 Short-read 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 was 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; $base_i \neq base_{i+1}$). 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. 2024). 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 & 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 & 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, based on the developers advice in the manual. 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 run with **samtools**. Reads are sorted in two ways, first alphabetically by read name using the collate command, then by genome position with the sort command. In between these sorting steps, we run the fixmate command 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. The -b option tells **samtools** that the input is in **BAM** format. 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% | 93.15% | 82.82% | 137 |
| | *L. arcana* | 98.69% | 93.00% | 82.15% | 8 |
| | *L. compressa* | 98.60% | 92.39% | 80.02% | 8 |

| | Species | Mapped | Primary | Paired | N |
|---|---|---|---|---|---|
| | *L. fabalis* | 97.91% | 80.38% | 71.91% | 432 |
| | *L. obtusata* | 97.61% | 81.39% | 73.72% | 14 |
| | *L. littorea* | 80.73% | 75.35% | 53.18% | 4 |

**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., 2024). 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. Similar conclusions were also reached by an intedpendent study using simulated data (Lefouili & Nam 2022). 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, which can be speed up by paralleling the job (see example below). 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 calculate many population genetic statistics (e.g. π and $d_{xy}$).

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

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, with the understanding that this limits the potential for some downstream analysis.

9.3 **Speeding up bcftools [optional]:**

Making a **mpileup** and calling the **VCF** are time consuming steps as there are a lot of calculations to make. Splitting the genome into regions with the -r option speeds up the process by parallelising the job. **bcftools** does not parallelise jobs well. The --threads option only works for compressing the output. We recommend using GNU **parallel**, a utility which can split a **bcftools** job into multiple smaller jobs. This is done in a few steps:

1. A chromosome or genome regions file is created. For *Littorina* we split each chromosome, smaller genomes could be split just by chromosome. As a rule of

thumb, smaller regions will run quicker, but will use more threads to run jobs in parallel.

2. **bcftools** is called within the **parallel** command. This will produce separate files, one for every row in the regions files.
3. The separate files need to be stitched together with **bcftool concat**. This step usually is very time consuming. As the output is an output **VCF**, it is also helpful to pipe the output into **bcftool sort** to recompress and reorder the data.

On the next page is an example of a parallel call of LG1 split into 112 jobs running across 20 threads. Trialling this on a server with 6.4 GB memory per thread, parallel calls of 1Mb regions of the genome were completed in an average of 16 hours.

```
### Calling in parallel for chromosome 1
## Step 1: create regions files for each chromosome
RegSize=1000000 # set region size in bp
chr="chr1"
# List chromosome lengths using reference genome index
samtools faidx /path/reference.fa
cut -f1-2 /path/reference.fa.fai > /path/chrSize.txt
# Length of target chromosome
chrL=$(grep -w "$chr" /path/chrSize.txt | cut -f2)
# List regions (e.g., “1000001-2000000”)
paste -d '-' \
          <(seq 1 $RegSize $chrL) \
          <(printf "$(seq $RegSize $RegSize $chrL)\n$chrL") \
          > /path/$chr_regions.txt
# Add chromosome prefix (e.g., “chr1:1000001-2000000”)
sed -i -e 's/^/'"$chr"':/' /path/$chr_regions.txt

## Step 2: variant call each region
cat path/$chr_regions.txt | parallel -j20 '  # -j sets number of
threads
          bcftools mpileup -Ou -a
FORMAT/AD,FORMAT/DP,FORMAT/SP,INFO/AD \
                    -b /path/list.of.bams.txt \
                    -f /path/reference.fa \
                    -r {} |\
           bcftools call -mOz -f GQ,GP -o /path/{}.vcf.gz'
# Note: output VCFs will be named by region (e.g., chr1:1-
1000000.vcf.gz)

## Step 3: concatenate VCFs of each region
ls /path/$chr*.vcf.gz > /path/list.of.regions.vcfs.txt
bcftools concat -f /path/list.of.regions.vcfs.txt |\
            bcftools sort -Oz - > /output/$chr.raw.vcf.gz

## Clean up
rm /path/$chr:*.vcf.gz /path/list.of.regions.vcf.txt
/path/$chr_regions.txt
```

## 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 may need tweaking (Hemstrom et al. 2024). 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 the filtering thresholds for your own data.

*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
bcftools view -H /path/output.vcf.gz | wc -l
# Don’t mix up the -h and -H options
```

11 **4.1) Hard filters:**
Hard filtering deletes entire positions from the **VCF** based on criteria that suggest the site was poorly sequenced or problematic. It 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 indel,other,mnps removes the indels and multi nucleotide polymorphisms, keeping only SNPs and invariant sites in the output. We also remove multiallelic sites by setting the maximum allele count with the -M 2 option.

```
bcftools filter -Ou -g 5:indel,other /path/output.vcf.gz |\
bcftools view -Oz -M 2 -V indels,other,mnps >
/path/SNP_only.vcf.gz
# Do not mix up -V which excludes indels, with -v which only keeps
indels
```

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. The average coverage can be extracted from the unfiltered **VCF** following the instruction in section 14.

```
bcftools filter -Oz -e 'INFO/DP>[VALUE]' /path/SNP_only.vcf.gz >
/path/depth_filt.vcf.gz
# Replace [VALUE] with 2.5 x average coverage
```

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

For high coverage calls there is a risk of falsely calling an invariant site by mistake. This risk can be reduced by setting a lower total depth limit. 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., 2023).

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 out 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
```

Finally, 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 are of adequate quality, we will 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, and GQ of 30 is a 0.1% chance. GQ ≥ 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 -Ou -S . -e 'FMT/GQ<20' /path/SP_filt.vcf.gz |\
bcftools filter -Oz -S . -e 'FMT/DP<5' - > /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 a lot of 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 (Table 2). Our filters 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. 2023.

```
# 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 any 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\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 based on allele frequency (as this is typically specific to only certain types of downstream analysis; Hemstrom et al., 2024), 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()
```

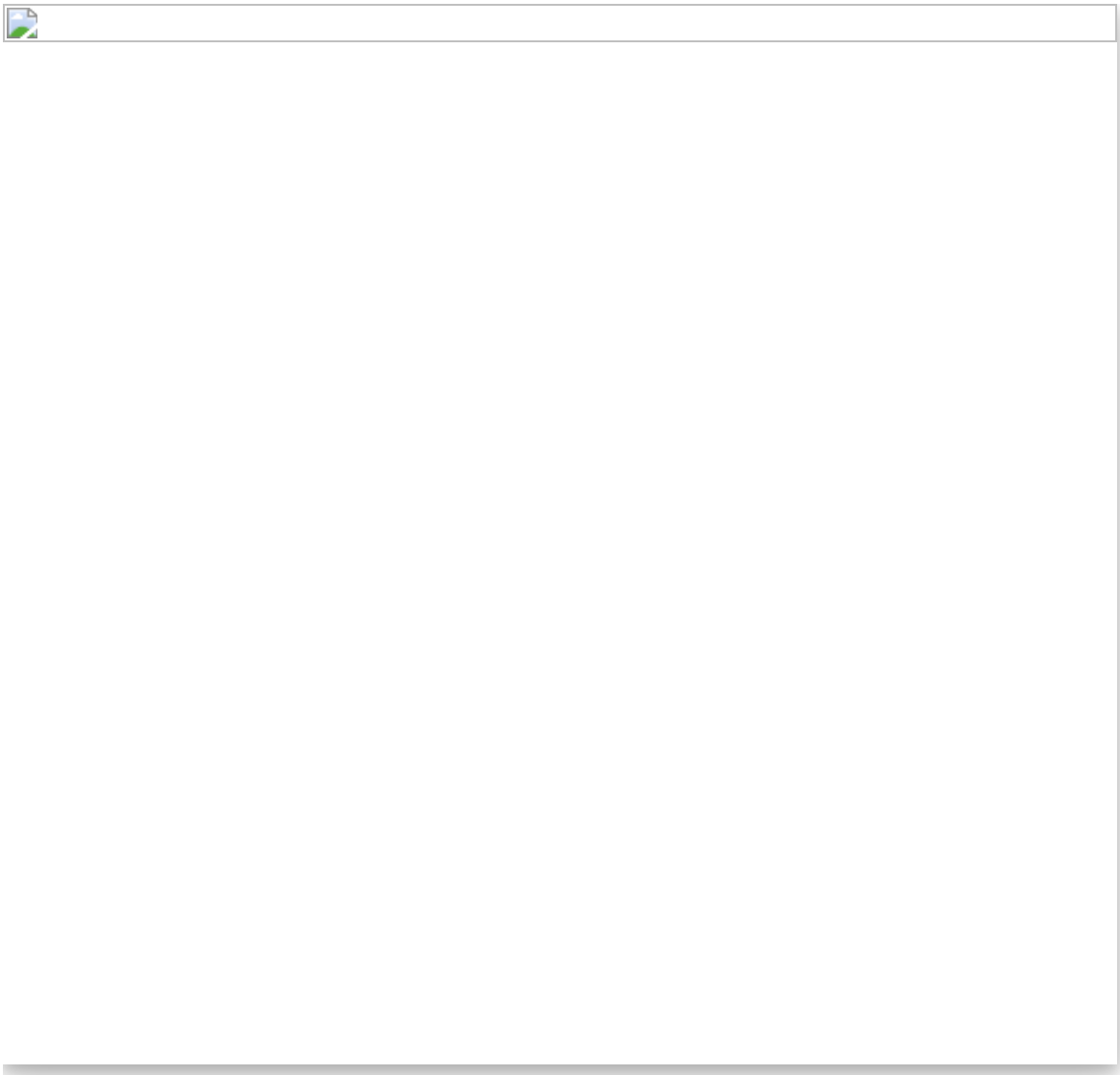

**Figure 2:** density curves for VCF statistics before (light red) and after (dark red) filtering. **DP** = total read depth per site. **AF** = frequency of the alternative allele. **QUAL** = Phred scaled variant call quality score. **MQ** = average Phred-scaled mapping quality. **SP** = Phred-scaled 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 | 0.12.1 | https://github.com/s-andrews/FastQC |
| MultiQC | Merging fastqc reports across the whole dataset | 1.22.2 | https://multiqc.info/docs/ |
| fastp | Trimming fastq files | 0.23.4 | https://github.com/OpenGene/fastp |
| BWA | Read mapping | 0.7.18 | https://bio-bwa.sourceforge.net/bwa.shtml |
| samtools | Processing and summarising BAM files | 1.20 | http://www.htslib.org/doc/samtools.html |
| bcftools | Merging BAM files and variant calling | 1.20 | https://samtools.github.io/bcftools/bcftools.html#mpileup |
| parallel | Parallelising job across multiple cores | 20230422 | https://www.gnu.org/software/parallel/man.html |
| vcfrandomsample | Randomly sample a VCF | 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.

## Protocol references

Chen, S., Zhou, Y., Chen, Y., & Gu, J. (2018). fastp: an ultra-fast all-in-one FASTQ preprocessor. Bioinformatics, 34(17), i884-i890. https://doi.org/10.1093/bioinformatics/bty560

De Jode, A., Faria, R., Formenti, G., Sims, Y., Smith, T.P., Tracey, A., Wood, J.M.D., Zagrodzka, Z.B., Johannesson, K., Butlin, R.K., & Leder, E.H. (2024). Chromosome-scale genome assembly of the rough periwinkle Littorina saxatilis. Genome Biology and Evolution, 16(4), evae076. https://doi.org/10.1093/gbe/evae076

Del Fabbro, C., Scalabrin, S., Morgante, M., & Giorgi, F.M. (2013). An extensive evaluation of read trimming effects on Illumina NGS data analysis. PLoS One, 8(12), e85024. https://doi.org/**10.1371/journal.pone.008502**4

Evolgenomics (2022). Haplotagging. GitHub repository. https://github.com/evolgenomics/haplotagging

Garrison E., Kronenberg, Z.N., Dawson, E.T., Pedersen, B.S., & Prins, P. (2022). A spectrum of free software tools for processing the VCF variant call format: vcflib, bio-vcf, cyvcf2, hts-nim and silvar. PLoS Computational Biology. 18(5): e1009123. https://doi.org/10.1371/journal.pcbi.1009123

Harvard Chan Bioinformatics Core training. (n.d.) Introduction to RNA-Seq using high performance computing. hpctraining. https://hbctraining.github.io/Intro-to-rnaseq-hpc-salmon/lessons/qc_fastqc_assessment.html

Hemstrom W., Grummer, J.A., Luikart, G., & Christie, M.R. (2024). Next-generation data filtering in the genomics era. Nature Reviews Genetics. https://doi.org/10.1038/s41576-024-00738-6

Illumina. (2023). Advantages of paired-end and single-read sequencing. Illumina Experiments & Protocols. https://emea.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html#:~:text=Unlike%20single%2Dread%20seqeuncing%2C%20paired,gene%20fusions%20and%20novel%20transcripts.

Kofler, R., Orozco-terWengel, P., De Maio, N., Pandey, R.V., Nolte, V., Futschik, A., Kosiol, C., & Schlötterer, C. (2011). PoPoolation: a toolbox for population genetic analysis of next generation sequencing data from pooled individuals. PLoS One. 6(1), e15925. https://doi.org/10.1371/journal.pone.0015925

Lefouili, M., & Nam, K. (2022). The evaluation of Bcftools mpileup and GATK HaplotypeCaller for variant calling in non-human species. Scientific Reports, 12, 11331. https://doi.org/10.1038/s41598-022-15563-2

Meier, J.I., Salazar, P.A., Kučka, M., & Chan, Y.F. (2021). Haplotype tapping reveals parallel formation of hybrid races in two butterfly species. PNAS. 118(25), e2015005118. https://doi.org/10.1073/pnas.2015005118

Musich, R., Cadle-Davidson, L., & Osier, M.V. (2021). Comparison of short-read sequence aligners indicates strengths and weaknesses for biologists to consider. Frontiers in Plant Science. 12.

https://doi.org/10.3389/fpls.2021.657240

Perini, S. (2021). Reproductive isolation at contact zones [Doctoral thesis, University of Gothenburg]. GUPEA. https://gupea.ub.gu.se/handle/2077/67013

Predeus, A. [predeus]. (2017). Duplicates on Illumina. [online forum]. Biostars. **https://www.biostars.org/p/229842/**

Puritz, J.B., Matz, M.V., Toonen, R.J., Weber, J.N., Bolnick, D.I., & Bird, C.E. (2014). Demystifying the RAD fad. Molecular Ecology. 23(24), 5937-5942. h**ttps://doi.org/10.1111/mec.12965**

Rochette, N.C., Rivera-Colón, A.G., & Catchen, J.M. (2019). Stacks 2: analytical methods for paired-end sequencing improve RADseq-based population genomics. Molecular Ecology. 28(21), 4737-4754. https://doi.org/10.1111/mec.15253

Schilbert,H.M., Rempel, A., & Pucker, B. (2020). Comparison of read mapping and variant calling tools for the analysis of plant NGS data. Plants. 9(4): 439. https://doi.org/10.3390/plants9040439

Schlötterer, C., Tobler, R., Kofler, R., & Nolte, V. (2014). Sequencing pools of individuals - mining genome-wide polymorphism data without big funding. Nature Reviews Genetics. 15, 749-763. https://doi.org/10.1038/nrg3803

Stankowski, S., Zagrodzka, Z.B., Garlovsky, M., Pal, A., Shipilina, D., Garcia Castilo, D., Lifchitz, H., Le Moan, A., Leder E., Reeve, J., Johannesson, K., Westram, A.M., & Butlin, R.K. (2023). The genetic basis of a recent transition to live-bearing in marine snails. Science. 383(6678), 114-119. https://doi.org/10.1126/science.adi2982

Shaji, A., Numanagić, I., & Berger, B. (2018). Latent variable model for aligning barcoded short-reads improves downstream analyses. Research in Computational Molecular Biology. Apr;10812:280-282. https://doi.org/10.1101/220236

Trapnell, C., & Salzburg, S.L. (2010). How to map billions of short reads onto genomes. Nature Biotechnology. 27(5), 455-457. https//doi.org/**10.1038/nbt0509-455**

Zanti, M., Michailidou, K., Loizidou, M.A., Machattou, C., Pirpa, P., Christodoulou, K., Spyrou, G.M., Kyriacou, K., & Hadjisavvas, A. (2021). Performance evaluation of pipelines for mapping variant calling and interval padding, for the analysis of NGS germline panels. BMC Bioinformatics. 22: 218. https://doi.org/10.1186/s12859-021-04144-1