Calling genetic variants with breseq

Daniel Padfield

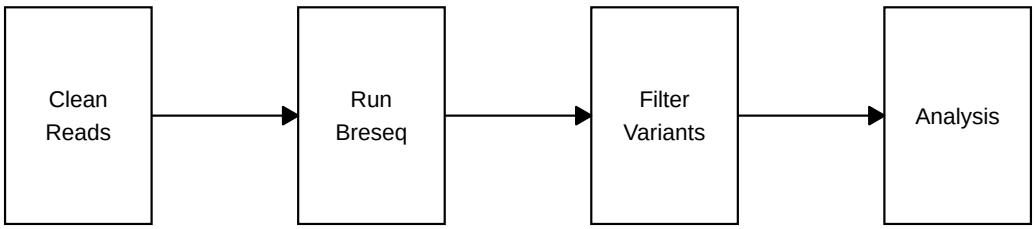
July 10, 2025

Table of contents

Outline

This protocol describes a general workflow for calling genetic variant using **breseq**, which is developed by the Barrick Lab. I like **breseq** because it is quite conservative, gives lots of intuitive output - as well as the traditional VCF file - and has really good support.

The general workflow is as follows.



Resources

There are many resources that go into more depth than this walkthrough does here. You should go and read them! Especially if you are new to *breseq* and want to understand how it works and its output.

- breseq documentation.

Pipeline

Where to run this?

If you have only a couple of files and you are not running it in polymorphism mode, you can probably run this on your local computer. However, if you have a lot of files with a lot of sequencing data each (e.g. you want to call variants from an experimental evolution experiment) then I would recommend trying to run this on either the RStudio Servers (which I have done before) or a HPC at Exeter (which I should have done before).

Install miniforge and breseq

This can be one of the most laborious, and possibly anger inducing, steps. Bioinformatics software can be notoriously difficult to install. However, there is a code chunk that can helpfully help with this. It first installs miniforge that installs **conda** and **mamba** package managers. It then sets a one time configuration for which repositories to use when installing packages, prioritising **conda-forge** and **bioconda** based on current recommendations.

Firstly we install **tmux** which allows us to run multiple terminal sessions at once, and is particularly useful for running long-running commands. A cheatsheet for tmux is here. The think I found most awkward was the `Ctrl-b`, then `d` for detach, but is easier when you realise `Ctrl-b` is the prefix action to all **tmux** commands.

We then create a new environment (which is a bit like a virtual machine) for all of the tools in the pipeline (actually just breseq and fastp). This is useful to avoid conflicts between different projects. You can learn more about conda environments here.

```
# install new mamba/conda installation if needed
curl -L -O
 ↪  "https://github.com/conda-forge/miniforge/releases/latest/download/M
 ↪  -m).sh"

bash Miniforge3-$(uname)-$(uname -m).sh

# update conda if needed
conda update conda

# set up channel configuration based on current best
 ↪  practice
conda config --add channels bioconda
conda config --add channels conda-forge
```

```
conda config --set channel_priority strict

# install tmux
conda install -c conda-forge tmux

# create conda environment for breseq
conda create -n breseq_env -c bioconda breseq fastp seqkit
```

Filter reads

Next we need to do some additional filtering of the short reads. Sequencing services do some standard filtering and normally give you a folder of the raw files called **Trimmed**, but we will do our own just to ensure the quality of the reads in the pipeline.

Our filtering removes: - Adapters - Median Q score, all bases > 30 - Expected read length > 95% of expected read length (e.g. for 2x 150bp reads, this would be 143bp, for 2x 300bp reads, this would be 285bp) - We check the average read length of a single file using `seqkit stats` to see what the expected read length is.

```
# activate conda environment
conda activate breseq_env

# set working directory to where the short reads are
wd=short_reads

# make fastp reports folder
mkdir -p "$wd/fastp_reports"

# look at one file to look for read length
file=$wd/307504_NalR2_1_trimmed.fastq.gz
seqkit stats $file
```

```
# run fastp on all the short reads
for file in $wd/*1_trimmed.fastq.gz; do

    fwd=$file
    # replace 1_trimmed.fastq.gz with 2_trimmed.fastq.gz to
     ↪  get the reverse read
    rev=${fwd%1_trimmed.fastq.gz}2_trimmed.fastq.gz

    #echo $fwd
    #echo $rev

    # run fastp on the file
    fastp -i $fwd -I $rev -o  "$wd/trimmed/$(basename $fwd)"
     ↪  -O "$wd/trimmed/$(basename $rev)" -w 4
     ↪  --detect_adapter_for_pe -l 237 -q 30 -j
     ↪  "$wd/fastp_reports/$(basename
     ↪  ${fwd%1_trimmed.fastq.gz}fastp.json)" -h
     ↪  "$wd/fastp_reports/$(basename
     ↪  ${fwd%1_trimmed.fastq.gz}fastp.html)"
done
```

You can then look at the output in **fastp_reports**. Or run **seqkit stats** on the filtered files.

```
# run seqkit on filtered files
seqkit stats $wd/trimmed/*.fastq.gz
```

File Name	Format	Type	Num Seqs	Sum Len	Min Len	Avg Len	Max Len
307504_NalR2_1_trimmed.fastq.gz	FASTQ	DNA	650,074	163,155,237	237	251	251
307504_NalR2_2_trimmed.fastq.gz	FASTQ	DNA	650,074	163,100,867	237	250.9	251

File Name	Format	Type	Num Seqs	Sum Len	Min Len	Avg Len	Max Len
307505_NalR3_1_trimmed.fastq.gz	FASTQ	DNA	444,294	111,504,387	27	251	251
307505_NalR3_2_trimmed.fastq.gz	FASTQ	DNA	444,294	111,465,523	27	250.9	251
307506_NalR4_1_trimmed.fastq.gz	FASTQ	DNA	656,664	164,809,429	27	251	251
307506_NalR4_2_trimmed.fastq.gz	FASTQ	DNA	656,664	164,741,483	27	250.9	251
307507_NalR5_1_trimmed.fastq.gz	FASTQ	DNA	655,940	164,628,343	27	251	251
307507_NalR5_2_trimmed.fastq.gz	FASTQ	DNA	655,940	164,573,108	27	250.9	251

Run breseq

Now we can run **breseq**. The command runs a for loop through all of the different samples, and runs **breseq** against a reference. This command is written as a task which allows us to run it in parallel. When **N=1**, it is run iteratively like a simple for loop, but when **N>1**, multiple instances of the task are run in parallel.

This example maps reads from individual samples back to the same reference genome, and is done on clonal sequencing data, so we are not expecting polymorphisms.

Things you will want to watch out for are: - If your reference genome is not a single contig, you might want to use the `--contig-reference` option. - If you have population level data, you will want to use the `-p` option to run in polymorphism mode. - This code assumes your forward and reverse reads are in the same folder, and their identifier is **1_trimmed.fastq.gz** and **2_trimmed.fastq.gz**. If this is not the case for your forward and reverse reads, you will need to change this! - **breseq** likes references in .gbk format. - **-j** is the number of threads. Set this to 8 or so if running on the RStudio server, might need to run it lower if on your local machine. - **breseq** by default does not provide annotation of variants in the VCF file, so we create those using **gdtools ANNOTATE** for each file. You can then use **R** code below to create a combined **csv** file with all the variants and their annotations.

```
# set working directory
wd=BASEFOLDER_OF_CHOICE

# set reference genome
ref=$wd/reference/reference.gbk

# set output folder
output_folder=$wd/breseq
# make folder if it does not exist
mkdir -p $output_folder

# write up the breseq command in a task
task(){
        # assign fwd and rev files
        file_fwd="$1"

        # assign fwd and rev files
        file_fwd="$1"

         ↪  file_rev=${file%_1_trimmed.fastq.gz}_2_trimmed.fastq.gz

        #stub
        stub=$(basename ${file%_1_trimmed.fastq.gz})

        echo $stub

        mkdir -p $wd/breseq/$stub

        # run breseq - there are two contigs so run them in
         ↪  -c contig mode
        breseq $file_fwd $file_rev -j 4 -o  $stub -r $ref

        # run gdtools to annotate the variants
        gdtools ANNOTATE $stub/output/output.gd -r $ref -f
         ↪  CSV --output $stub/output/output.csv
```

```
}

# run one instance of this
N=1

# try and run it on 6 instances at once
(
for file in $trimmed_files/*_1_trimmed.fastq.gz; do
   ((i=i%N)); ((i++==0)) && wait
   task "$file" &
done
)
```

This will create a folder called **breseq** in the working directory, and within that folder will be a folder for each sample containing intermediate and output files for that sample.

You can then go into R and create a combined CSV file with all the variants and their annotations for each sample.

```
# load in libraries
librarian::shelf(tidyverse, vcfR)

# list folders in the breseq output folder
folders <- list.dirs("breseq", full.names = TRUE, recursive
 ↪  = FALSE)

# for loop to create combined CSV file
for (i in 1:length(folders)) {
  temp_folder <- folders[i]

  # read in the output CSV file
  output_file <- file.path(temp_folder, "output",
 ↪  "output.csv") %>%
```

```
    read.csv() %>%
    select(
      ref_seq,
      new_seq,
      type,
      snp_type,
      gene_product,
      gene_position,
      gene_name
    )

  # read in vcf file
  vcf_file <- file.path(temp_folder, "output", "output.vcf")
  ↪ %>%
    read.vcfR(
      breseq_files[str_detect(breseq_files, ids[i])],
      verbose = FALSE
    ) %>%
    vcfR2tidy(., info_only = TRUE) %>%
    .$fix %>%
    janitor::clean_names()

  # combine and save out
  bind_cols(d_vcf, d_gd) %>%
    write.csv(
      file = file.path(
        temp_folder,
        "output",
        paste(base_name, "annotated_output.csv", sep = '_')
      ),
      row.names = FALSE
    )
}
```

This should result in a combined CSV file with all the variants and their annotations for each sample. They should be named by the folder/sample name **annotated_output.csv**. These can then be used in downstream analysis, although the other **breseq** output is also very useful.