Population Genomics Tutorial 2: Trim and QC the raw sequence reads

SRK

Fall 2025

In Bioinformatics, we create pipelines that represent our analysis workflow, from raw data to intermediate data outputs to final results and figures. For our spruce data, the first part of our pipeline will be doing some QC on our raw data then mapping the reads to the reference genome.

1. Intro to fastq files

Whenever you get a new batch of NGS data, the first step is to look at the data quality of coming off the sequencer and see if we notice any problems with base quality, sequence length, PCR duplicates, or adapter contamination. A lot of this info is stored in the raw data files you get from the core lab after sequencing, which are in **"fastq" format**.

The fastq files for our project are stored in this path:

`/gpfs1/cl/ecogen/data/pbio6800/PopulationGenomics/fastq/red_spruce`

`cd` over there and `ll` to see the files. There should be 190 fastq files: 2 for each of the 95 red spruce.

Note: there are 2 files/sample because these are paired-end reads, and each sample gets a file with the forward reads (R1) and another with the reverse reads (R2).

The naming convention for our data is:

`<PopCode>_<RowID>_<ColumnID>_<ReadDirection>.fast.gz`

Together, `<PopCode>_<RowID>_<ColumnID>` define the unique individual ID for each DNA sample, and there should be 2 files per sample (an R1 and an R2)

You'll also see a folder with black spruce fastq files here:

`/gpfs1/cl/ecogen/data/pbio6800/PopulationGenomics/fastq/black_spruce`

with paired-end sequences for 18 black spruce samples (=36 fastq files)

So...what is a .fastq file anyway?

A fastq file is the standard sequence data format for NGS. It contains the sequence of the read itself, the corresponding quality scores for each base, and some meta-data about the read.

The files are big (typically many Gb compressed), so we can't open them completely. Instead, we can peek inside the file using `head`. But size these files are compressed (note the .gz ending in the filenames), and we want them to stay compressed while we peek. Bash has a solution to that called `zcat`. This lets us look at the .gz file without decompressing it all the way. Let's peek inside a file:

```
zcat 2505_9_C_R2.fastq.gz | head -n 4

@A00354:455:HYG3FDSXY:1:1101:3893:1031 2:N:0:CATCAAGT+TACTCCTT
GTGGAAAATCAAAACCCTAATGCTGAAAGGAATCCAAATCAAATAAATATTTTCACCGACCTGTTTCGATGCCAGAATTGTCTGCGCAGAAC
+
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FF
```

Note: `zcat` lets us open a .gz (gzipped) file; we then "pipe" `|` this output from `zcat` to the `head` command and print just the top 4 lines `-n4`

The fastq file format has 4 lines for each read:

Line	Description
1	Always begins with '@' and then information about the read
2	The actual DNA sequence
3	Always begins with a '+' and sometimes the same info in line 1
4	A string of characters which represent the **quality** scores; always has same number of characters as line 2

Here's a useful reference for understanding Quality (Phred) scores. If P is the probability that a base call is an error, then:

Q = -10*log10(P)

So:

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10,000	99.99%

The Phred Q score is translated to ASCII characters so that a two digit number can be represented by a single character.

```
 Quality encoding: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
                    |         |         |         |
     Quality score: 0........10........20........30........40
```

What kind of characters do you want to see in your quality score?

2. Clean and visualize fastq data quality using Fastp

We'll use the **Fastp** program to clean the reads for each file. See also the paper published by Chen in May 2023. **Fastp** looks at the quality collectively across all reads in a sample. The program is already installed on the VACC.

Open a terminal and at the command-line type:

```
module load gcc fastp

fastp -h
```

You should see a help menu printed out with lots of options to run the program!

3. Set up our directory structure

OK, now we need to set up our directories to hold our work.

First, let's `cd` to our repo `~/projects/eco_genomics2025`, make a new directory to store the work for the Population Genomics module:

`mkdir population_genomics`

and lastly set up some new directories to store our work. We'll make directories to store our data, scripts, results, documents, and log files:

```
mkdir mydata
mkdir myscripts
mkdir myresults
mkdir mydocs
mkdir mylogs
```

Then let's cd into the `myresults/` folder then use `pwd` to prove to yourself that you're in the `myresults/` folder within your home directory. It should look like this (but with your home directory info instead of mine):

`/users/s/r/srkeller/projects/eco_genomics2025/population_genomics/myresults`

Now within `myresults/` let's make another folder called `fastp_reports/` to hold the QC outputs from our **Fastp** analysis. Do that on your own, just like we did above, then cd into the `fastp_reports/` folder and type `pwd` again to prove to yourself you did it right.

Now, we're ready to run **Fastp** to look at the quality of our sequencing.

4. Write a bash script to run Fastp for multiple samples

We *could* do these 1 sample at a time, but that'd be inefficient.

Since we're doing multiple samples from the same population, we want to be clever and process in a batch instead of manually one at a time. We can do this by writing a bash script that contains a loop. This is the first taste of how bash scripting is powerful!

The basic syntax of a bash loop is like this:

```
cd <path to the input data>

for FILE in somelist

do

  command1 -options ${FILE} -moreOptions
  command2 -options ${FILE} -moreOptions

done
```

Note the use of variable assignment using `${}`. We define the word `FILE` in the "for loop" as the variable of interest, and then call the iterations of it using ${FILE}. For example, we could use the wildcard character (*) in a loop to call all files that include the ID code for your population and then pass those filenames in a loop to the commands.

If your pop was "2505", then you could write a loop that would process all the R1 fastq files in that population like this:

```
cd <path to the input data>

MYPOP="2505"

for FILE in ${MYPOP}*R1.fastq.gz

do

  command1 -options ${FILE} -moreOptions
  command2 -options ${FILE} -moreOptions

done
```

That's the basic idea! Let's do it...

We've provided a partially completed example script here:

`/gpfs1/cl/ecogen/pbio6800/PopulationGenomics/scripts/fastp.sh`

1. Make a copy of the script using `cp` and put it into your `myscripts/` directory
2. Open and edit your copied script using the RStudio file editor.
3. Edit the file so that you're trimming the fastq files for the population code assigned to you; you can add annotations as notes to yourself using the hashtag (#)
4. Save the file after you're done making changes.

Note: A quirk with **Fastp** is that it needs both read pairs (i.e., R1 and R2 files) given to it at the *same* time. This makes the use of name variables in the loop a bit tougher since we can't rely just on the population name (example, `2505`) but also need each individual name too (example, `2505_9_C`).

We'll use a couple of tricks to recycle the R1 file name to quickly create a matching R2 file name within the loop. We'll do a similar trick to quickly name the output files adding "_clean" to the end. We'll talk about this in class together, but there are also annotations in the script itself explaining what's happening.

Once you're ready, go ahead and cd into your 'myscripts/' folder and run your bash script:

```
bash fastp.sh
```

Note that the output (the trimmed and cleaned reads) are going to get saved to a folder on the network drive that you've been given write access to:

`/gpfs1/cl/ecogen/data/pbio6800/PopulationGenomics/cleanreads`

5. Visualize pre- and post-trimming read quality with the `fastp` html output

Conveniently, `fastp` is not only fast, but also produces a summary of the change in quality pre- and post-trimming. The output is in an html file that we can look at in a browser. Pretty cool!

How does the quality look?

Also, since we made some changes (added files) to our github repo, we should practice staging, committing, and then pushing to GitHub! Don't forget to pull first to make sure you're up to date with main ;)

6. Be sure to take notes on your workflow so you can remember what you did for the future!

An important final step is taking good notes on your workflow so you can remember what you did down the road (your "future self") and share your process with others (reproducible science!). It's also really important once you start making detailed decisions that will affect the analysis outcome of your data, so you can recreate the results and explore the effect of different assumptions/decisions.

We want you to keep such a notebook for each module in the course (kind of like you would for each experiment, or each thesis chapter you will work on).

You can create this as a plain text file in RStudio on the OnDemand VACC server. Let's name this file:

`popgen_notes.md`

and save it within your `eco_genomics2025/population_genomics/` repo directory.

You can edit this file directly within RStudio, taking notes using either the `Source` interface if you know the markdown language (or want to learn) or you can use RStudio's built-in markdown GUI editor under the `Visual` tab. This works very similar to Word.

Here's a cheatsheet for markdown language if you want to write using `Source` code:

Markdown cheatsheet