BIOL550-Lab1_Report

February 5, 2026

1 BIOL550 Lab 1 — Trapnell Data QC + Alignment (FastQC, FastX, STAR)

Date: 2026-02-05
Goal: QC raw reads, attempt cleanup using FastX based on QC findings, and run one STAR alignment example.

This notebook is organized as **Step → Implementation → Notes/Results** blocks for readability.

[1]:
```
!ssh -X pzg8794@sequoia.rit.edu "hostname; whoami; pwd"
```

```
RIT information technology resources are for the use of the RIT community only.
By using RIT information technology resources you acknowledge that you have read
and comply with RIT's Code of Conduct for Computer and Network Use and RIT's
Information Security Policy and Standards. Use of RIT information technology
resources may be monitored and unauthorized use is strictly prohibited.
sequoia
pzg8794
/home/pzg8794
```

1.1 Data Locations

These paths are where artifacts live **on your Mac** (downloaded from `sequoia`): - `qc_bundle/fastqc_out/` (raw FastQC) - `qc_bundle/fastqc_trimmed_out/` (trimmed FastQC) - `qc_bundle/star_align/` (STAR logs for one sample) - `multiqc_report/multiqc_report.html`

[2]:
```
from pathlib import Path

LAB_DIR = Path('/Users/pitergarcia/DataScience/Semester5/BIOL550/BIOL550-Lab')
BUNDLE = LAB_DIR / 'qc_bundle'
RAW_QC = BUNDLE / 'fastqc_out'
TRIM_QC = BUNDLE / 'fastqc_trimmed_out'
STAR_DIR = BUNDLE / 'star_align' / 'GSM794486_C2_R1'
MULTIQC_OUT = LAB_DIR / 'multiqc_report'

print('LAB_DIR:', LAB_DIR)
print('BUNDLE exists:', BUNDLE.exists())
print('RAW_QC exists:', RAW_QC.exists())
print('TRIM_QC exists:', TRIM_QC.exists())
print('STAR_DIR exists:', STAR_DIR.exists())
print('MULTIQC_OUT exists:', MULTIQC_OUT.exists())
```

```
LAB_DIR: /Users/pitergarcia/DataScience/Semester5/BIOL550/BIOL550-Lab
BUNDLE exists: True
RAW_QC exists: True
TRIM_QC exists: True
STAR_DIR exists: True
MULTIQC_OUT exists: True
```

[3]:
```
%%bash
set -euo pipefail

ssh pzg8794@sequoia.rit.edu '
  set -euo pipefail
  EXPORT=~/BIOL550/Lab1/exports/$(date +%F)
  mkdir -p "$EXPORT"

  tar -czf "$EXPORT/fastqc_raw.tgz" -C ~/BIOL550/Lab1 fastqc_out
  tar -czf "$EXPORT/fastqc_trimmed.tgz" -C ~/BIOL550/Lab1 fastqc_trimmed_out

  ls -lh "$EXPORT"/*.tgz
'
```

```
RIT information technology resources are for the use of the RIT community only.
By using RIT information technology resources you acknowledge that you have read
and comply with RIT's Code of Conduct for Computer and Network Use and RIT's
Information Security Policy and Standards. Use of RIT information technology
resources may be monitored and unauthorized use is strictly prohibited.
```

```
-rw-rw-r-- 1 pzg8794 pzg8794 4.6M Feb  5 23:28
/home/pzg8794/BIOL550/Lab1/exports/2026-02-05/fastqc_raw.tgz
-rw-rw-r-- 1 pzg8794 pzg8794 4.6M Feb  5 23:28
/home/pzg8794/BIOL550/Lab1/exports/2026-02-05/fastqc_trimmed.tgz
-rw-rw-r-- 1 pzg8794 pzg8794 400M Feb  5 10:56
/home/pzg8794/BIOL550/Lab1/exports/2026-02-05/star_align_bams.tgz
-rw-rw-r-- 1 pzg8794 pzg8794 400M Feb  5 10:54
/home/pzg8794/BIOL550/Lab1/exports/2026-02-05/star_align_logs.tgz
```

1.2 Step 1 — QC With FastQC (Raw Reads)

I ran FastQC on all 12 raw FASTQ files (`6 samples × 2 mates`).

Then, I summarize PASS/WARN/FAIL calls by reading each `*_fastqc.zip` → `summary.txt`.

[4]:
```
import sys
import zipfile
from collections import Counter
```

```
def summarize_fastqc(zip_paths):
    fail = Counter()
    warn = Counter()
    total = 0

    for z in zip_paths:
        total += 1
        with zipfile.ZipFile(z) as zf:
            name = next((n for n in zf.namelist() if n.endswith('summary.
 ↪txt')), None)
            if not name:
                continue
            for line in zf.read(name).decode('utf-8', errors='replace').
 ↪splitlines():
                status, module, *_rest = line.split('        ')
                if status == 'FAIL':
                    fail[module] += 1
                elif status == 'WARN':
                    warn[module] += 1

    return total, fail, warn

raw_zips = sorted(RAW_QC.glob('*_fastqc.zip'))
raw_total, raw_fail, raw_warn = summarize_fastqc(raw_zips)

print('Raw FastQC zip count:', raw_total)
for mod, n in raw_fail.most_common():
    print(f'RAW FAIL: {mod}: {n}/{raw_total}')
for mod, n in raw_warn.most_common():
    print(f'RAW WARN: {mod}: {n}/{raw_total}')
```

```
Raw FastQC zip count: 12
RAW FAIL: Per base sequence quality: 12/12
RAW FAIL: Per sequence quality scores: 12/12
RAW FAIL: Per sequence GC content: 12/12
```

Notes/Results (Raw FastQC):

- Expectation: this dataset commonly shows quality drop-offs and GC-distribution deviations in FastQC.
- In my run, the FAIL modules were consistent across all raw mates (see printed output above).

1.3 Step 1b — Cleanup Attempt With FastX (Quality Trimming)

On `sequoia`, we used `fastq_quality_trimmer` (FastX 0.0.13) to trim low-quality tails.

This produced 12 uncompressed trimmed FASTQs (`*.trim.fq`) and used ~22 GB of disk because the outputs are not gzipped.

```
[5]: %%bash
set -euo pipefail

# Implementation (runs on sequoia). Safe guard: if outputs exist, skip␣
↪recomputing.
ssh pzg8794@sequoia.rit.edu bash -lc '
  set -euo pipefail
  READS="/home/pzg8794/BIOL550/Lab1/Trapnell_Data/Trapnell Data/Raw reads"
  TRIM="$HOME/BIOL550/Lab1/fastx_trimmed"
  mkdir -p "$TRIM"

  if ls "$TRIM"/*.trim.fq >/dev/null 2>&1; then
    echo "Found existing trimmed FASTQs in $TRIM; skipping FastX trimming."
    exit 0
  fi

  for f in "$READS"/*.fq.gz; do
    base="$(basename "$f" .fq.gz)"
    zcat "$f" | /usr/local/bin/FastX/0.0.13/fastq_quality_trimmer -Q33 -t 20 -l␣
↪0       -o "$TRIM/${base}.trim.fq"
  done

  echo "FastX trimming complete."
'
```

```
RIT information technology resources are for the use of the RIT community only.
By using RIT information technology resources you acknowledge that you have read
and comply with RIT's Code of Conduct for Computer and Network Use and RIT's
Information Security Policy and Standards. Use of RIT information technology
resources may be monitored and unauthorized use is strictly prohibited.
```

```
Found existing trimmed FASTQs in /home/pzg8794/BIOL550/Lab1/fastx_trimmed;
skipping FastX trimming.
```

```
bash: -c: option requires an argument
```

Notes/Results (FastX):

- Trimming at `-t 20` is a moderate cleanup step.
- It may reduce low-quality tails but FastQC PASS/FAIL calls can still remain FAIL depending on FastQC thresholds and dataset characteristics.

1.4 Step 1c — QC With FastQC (Trimmed Reads)

I re-ran FastQC on the trimmed reads and summarized PASS/WARN/FAIL.

```
[6]: trim_zips = sorted(TRIM_QC.glob('*.trim_fastqc.zip'))
trim_total, trim_fail, trim_warn = summarize_fastqc(trim_zips)

print('Trimmed FastQC zip count:', trim_total)
```

```
for mod, n in trim_fail.most_common():
    print(f'TRIM FAIL: {mod}: {n}/{trim_total}')
for mod, n in trim_warn.most_common():
    print(f'TRIM WARN: {mod}: {n}/{trim_total}')
```

```
Trimmed FastQC zip count: 12
TRIM FAIL: Per base sequence quality: 12/12
TRIM FAIL: Per sequence quality scores: 12/12
TRIM FAIL: Per sequence GC content: 12/12
```

Notes/Results (Trimmed FastQC):

- The same three modules remained FAIL across all trimmed mates.
- Next iteration would try: more aggressive trimming, read filtering (`fastq_quality_filter`), and/or adapter clipping.

1.5 Step 2 — STAR Alignment (One Worked Example)

I built a STAR index using a matching NCBI reference genome + GTF (assembly `GCF_000001215.4`), then aligned one sample: - `GSM794486_C2_R1` (paired-end, gzipped)

Then, I read key alignment metrics from `Log.final.out`.

[7]:
```
log_final = STAR_DIR / 'Log.final.out'
print('Log.final.out exists:', log_final.exists())

if log_final.exists():
    text = log_final.read_text(encoding='utf-8', errors='replace')
    keys = [
        'Number of input reads',
        'Average input read length',
        'Uniquely mapped reads number',
        'Uniquely mapped reads %',
        '% of reads mapped to multiple loci',
        '% of reads mapped to too many loci',
        'Mapping speed',
    ]
    for line in text.splitlines():
        if any(k in line for k in keys):
            print(line)
```

```
Log.final.out exists: True
       Mapping speed, Million of reads per hour |       248.73
                          Number of input reads |       11607325
                      Average input read length |       150
                   Uniquely mapped reads number |       11516696
                        Uniquely mapped reads % |       99.22%
             % of reads mapped to multiple loci |       0.69%
             % of reads mapped to too many loci |       0.09%
```

Notes/Results (STAR):

- This notebook documents **one** successful alignment example.
- For full differential expression, one would align all samples and generate a gene count matrix.

1.6 Step 3 — MultiQC Summary Report

MultiQC aggregates FastQC + STAR outputs into one HTML report.

Note: MultiQC detects the raw FastQC `*_fastqc.zip` by default. Trimmed reports are named `*.trim_fastqc.zip`, so I rely on the summary tables above for trimmed-vs-raw comparisons.

[8]:
```python
import shutil
import subprocess
from pathlib import Path

MULTIQC_OUT.mkdir(parents=True, exist_ok=True)
report = MULTIQC_OUT / 'multiqc_report.html'

# Avoid modifying/breaking the active environment. If the report already␣
↪exists, reuse it.
if report.exists():
    print('MultiQC report already exists:', report)
else:
    # Prefer a multiqc executable already on PATH.
    mqc = shutil.which('multiqc')
    if mqc:
        subprocess.check_call([mqc, '-o', str(MULTIQC_OUT), str(BUNDLE),␣
↪'--force'])
        print('MultiQC report:', report)
    else:
        # Fallback: use the known-working .quantum environment to run MultiQC.
        quantum_py = Path('/Users/pitergarcia/DataScience/Semester4/GA-Work/.
↪quantum/bin/python')
        if quantum_py.exists():
            subprocess.check_call([str(quantum_py), '-m', 'multiqc', '-o',␣
↪str(MULTIQC_OUT), str(BUNDLE), '--force'])
            print('MultiQC report:', report)
        else:
            raise RuntimeError('multiqc not found on PATH, and .quantum python␣
↪was not found. Install MultiQC in your active env or set quantum_py.')
```

```
MultiQC report already exists: /Users/pitergarcia/DataScience/Semester5/BIOL550/
BIOL550-Lab/multiqc_report/multiqc_report.html
```

1.6.1 Local

[2]:
```python
import webbrowser
webbrowser.open("file:///Users/pitergarcia/DataScience/Semester5/BIOL550/
↪BIOL550-Lab/multiqc_report/multiqc_report.html")
```

```
[2]: True
```

1.6.2 Drive

```
[5]: webbrowser.open("https://drive.google.com/file/d/
      ↪144BQHnsAVKfSaR5Yz_sWyGh4NjIMNL8v/view?usp=sharing")
```

```
[5]: True
```

Report artifact:

- `multiqc_report/multiqc_report.html`

Recommendation: open MultiQC in your browser (inline embedding can cause CSS to cover the notebook UI).