SF-seq

Jake VanCampen

Sun Oct 1 21:26:02 2017

SF-seq read quality score distributions

In this assignment I worked with the following read files from `/projects/bgmp/2017_sequencing/demultiplexed`:

```
2_2B_control_S2_L008_R1_001.fastq.gz
2_2B_control_S2_L008_R2_001.fastq.gz
29_4E_fox_S21_L008_R1_001.fastq.gz
29_4E_fox_S21_L008_R2_001.fastq.gz
```

To determine quality score distributions for each read the program FastQC was loaded on talapas along with it's dependencies: `ml easybuild intel/2017a FastQC`

In a working directory called SF-seq, a new directory was created for the output of FastQC. FastQC was then run using the following options on all four files.

```
$ fastqc --noextract -o fastqc_out -t 4 \
2_2B_control_S2_L008_R1_001.fastq.gz 2_2B_control_S2_L008_R2_001.fastq.gz \
29_4E_fox_S21_L008_R1_001.fastq.gz 29_4E_fox_S21_L008_R2_001.fastq.gz
```

Results

Quality score distributions for the forward and reverse reads as well as per-base N content of both treatments were extracted from the FastQC output and compared to result from my quality score distribution plots.

The quality score distributions for the treatment `29_4E_fox_S21` show high average quality scores (approching 40) for read one with lower quality scores at the begining and slowly tapering towards the end of the read (Figure 1). Read two shows a lower quality score across all the bases, not exceeding 38, with low quality at the start and end of the read (Figures 2). The percent N-content across the bases for reads one and two show the highest percent N-content at the first base position for both reads, and undetectable N-content over the remaining base positions (Figures 3, 4). The detectable N-content at the first base position explains the low quality start in the quality score distributuions because an 'N' at that position will decrease the average quality score at that base position. The plots for the treatment `2_2B_control` show a similar pattern where read two shows a lower average quality score than read one, and the low quality score at the begining of the read is reflected in the percent N-content of each read at base position one (Figure 5-8).

I then ran my quality score distribution script `qscore_dist2.py` on the files using the following slurm script:

```
#!/bin/bash
#SBATCH --partition=fat            ### partition
#SBATCH --job-name=QD     ### Job Name
#SBATCH --output=QD.out    ### File in which to store job output
#SBATCH --error=QD.err     ### File in which to store job error messages
#SBATCH --time=0-24:00:00          ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                  ### Node count required for the job
#SBATCH --ntasks-per-node=28       ### Nuber of tasks to be launched per Node
#SBATCH --mail-user=jvancamp@uoregon.edu        ### Notifyme
#SBATCH --mail-type=ALL                         ### All of it

# load the necessary modules
```

```
ml zlib/1.2.11 python3/3.6.1

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/2_2B_control_S2_L008_R1_001.fastq

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/2_2B_control_S2_L008_R2_001.fastq

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/29_4E_fox_S21_L008_R1_001.fastq

# run qscore_dist2.py on all files
time ./qscore_dist2.py -f ~/SF-seq/data/29_4E_fox_S21_L008_R2_001.fastq
```

The error file from the script held the time of each run, showing that each run took less than 20 minutes, while the total runtime was just over one hour. While this is slower than the total runtime of FastQC (2m 46s) this could because FastQC is written in Java which is compiled rather than interpreted, and whose data structures might be more efficient; result in a faster runtime for this type of analysis. The FastQC analysis I ran also included the threading option -t 4 which ran the analysis for each file as a separate task.

```
$ cat QD.err

The following have been reloaded with a version change:
  1) zlib/1.2.8 => zlib/1.2.11


real    18m24.967s
user    18m21.792s
sys 0m1.324s

real    18m28.796s
user    18m26.856s
sys 0m1.334s

real    15m24.851s
user    15m23.165s
sys 0m1.163s

real    15m16.293s
user    15m14.693s
sys 0m1.070s
```

The quality score distribution plots from my script are shown in Figures 9-12. These distributuions show similar patterns as the results from the FastQC output. Read two for both treatments shows a lower average quality score for more base positions than in read one, and the quality score drop-off toward the begining and end of the read is similar to the results found in the FastQC output as expected.

Adaptor trimming comparison

The sequences must be filtered to trim known adapter sequences from each read so that only the insert sequences remain. With the common occurance of PCR primer contamination, PCR primers are also important to remove. There are a variety of programs that accomplish these tasks, some loaded on Talapas include: `process_shortreads` (part of the Stacks pipeline), `Trimmomatic`, and `cutadapt`. The programs are notably different in a couple of ways. First, Trimmomatic is written in Java, cutadapt is a python package available through the Python Package Index, and Process shortreads is written in C++ and Perl. There are numerous differences for dealing with paired end reads using the mentioned programs. Trimmomatic takes forward and reverse reads, and outputs four files: both paired and unpaired forward and reverse reads. Paired forward and reverse reads have to be complementary, while unpaired forward and reverse read files include reads where either the forward or reverse did not meet the quality threshold. Process_shortread by default discards low quality reads unless their retention is specified as an option. Cutadapt by defualt must take pairs of reads, and will throw out the pair if quality is low, outputting adapter-trimmed read pairs to two forward and reverse read files. Demultiplexing is not supported in cutadapt where it is supported in Trimmomatic and Process_shortreads. This analysis was carried out using process_shortreads because it contains all functionality necessary for this analysis with concise runtime options. The following scripts were run to properly trim adapter sequences from sequenced reads:

```
# run process_shortreads on the set of paired end-reads
process_shortreads -1 ~/SF-seq/2_2B_control_S2_L008_R1_001.fastq.gz -2 ~/SF-seq/2_2B_control_S2_L008_R2
--adapter_1 ACACTCTTTCCCTACACGACGCTCTTCCGATCT \
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT


# run process_shortreads on the set of paired end-reads
process_shortreads -1 ~/SF-seq/29_4E_fox_S21_L008_R1_001.fastq.gz -2 ~/SF-seq/29_4E_fox_S21_L008_R2_001
--adapter_1 ACACTCTTTCCCTACACGACGCTCTTCCGATCT \
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT
```

The results for each treatment are shown below:

```
File 29_4E_fox_S21_L008_R1_001.fastq.gz
Total Sequences 9654866
Reads containing adapter sequence   297563
Ambiguous Barcodes  0
Low Quality 0
Trimmed Reads   275693
Orphaned Paired-ends    0
Retained Reads  9632996

File 2_2B_control_S2_L008_R1_001.fastq.gz
Total Sequences 11661330
Reads containing adapter sequence   331159
Ambiguous Barcodes  0
Low Quality 0
Trimmed Reads   272157
Orphaned Paired-ends    0
Retained Reads  11602328
```

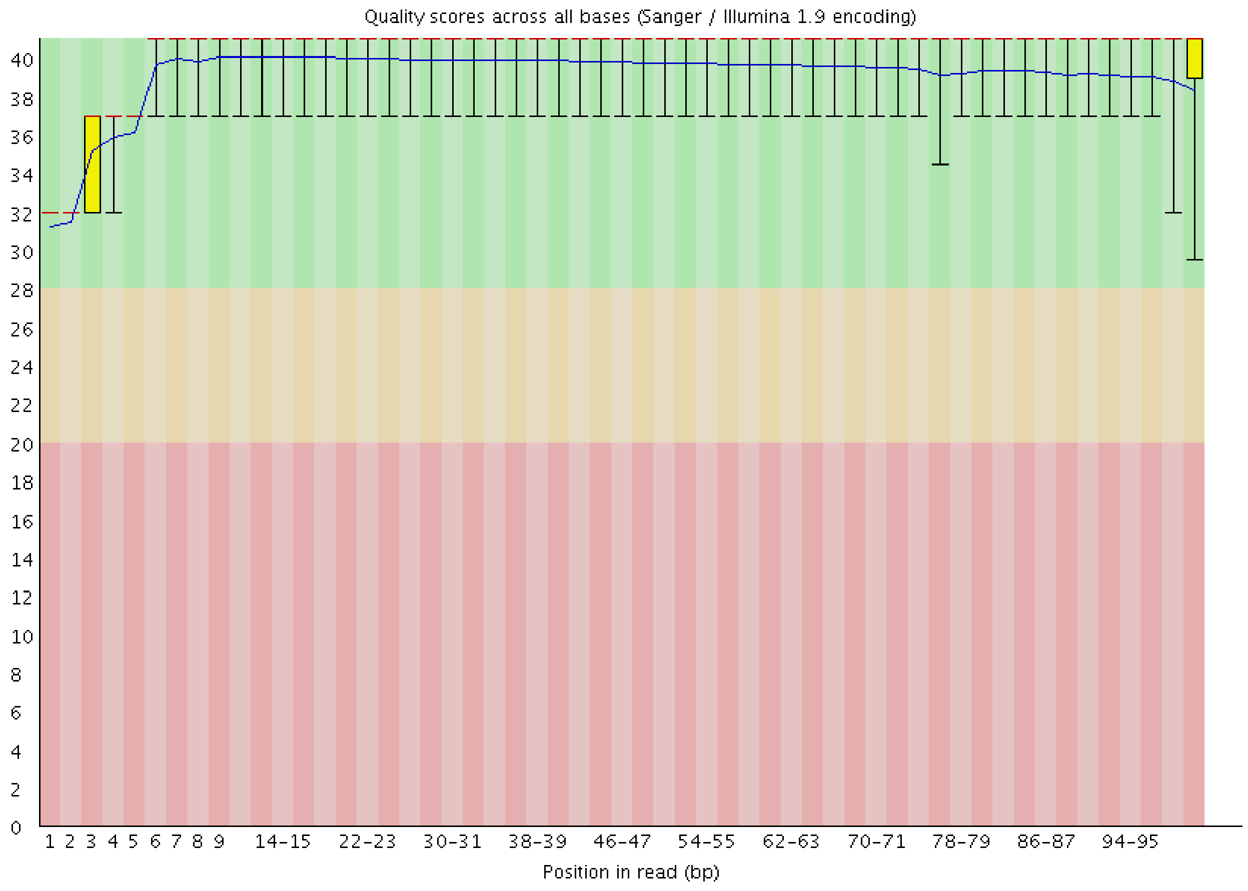

Figure 1: 29_4E_fox_S21 Read1 QScore distribution

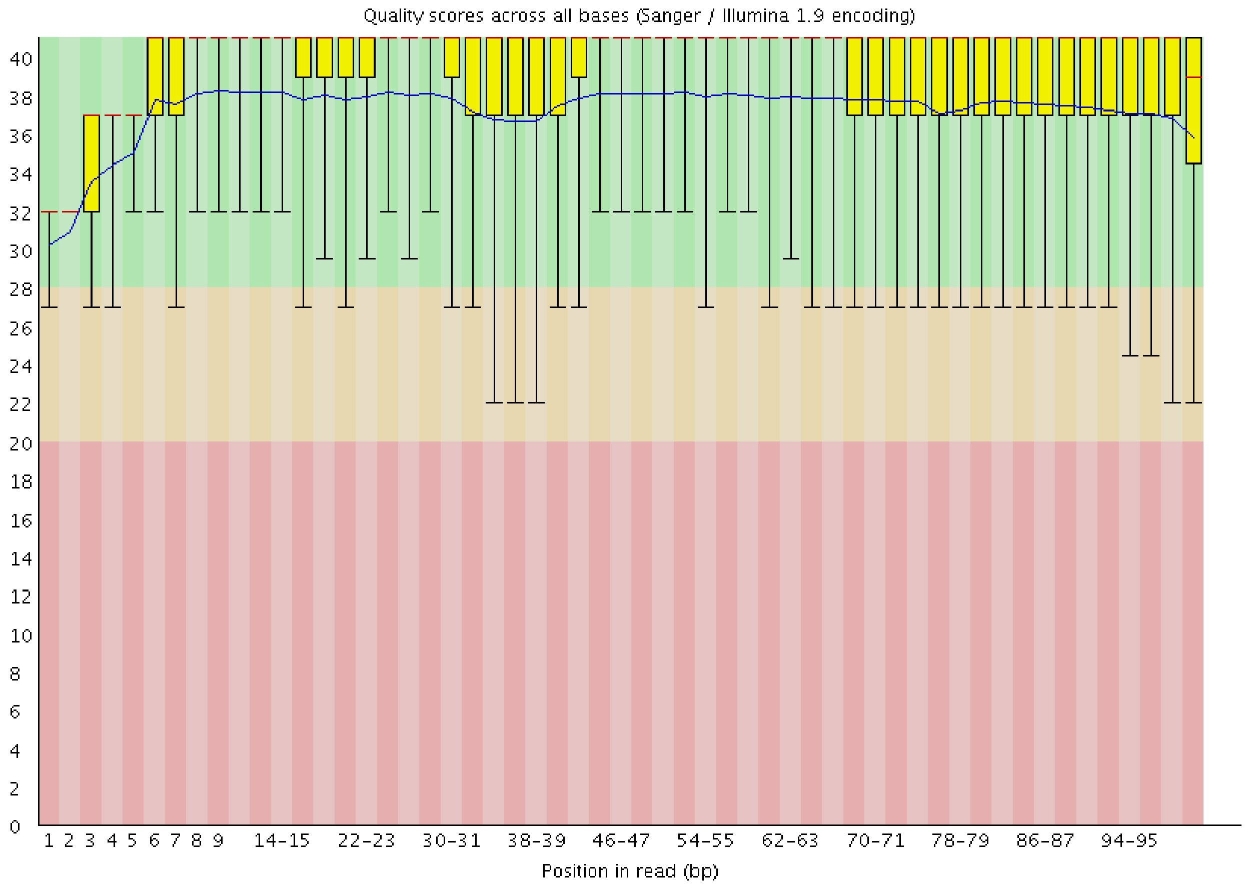

Figure 2: 29_4E_fox_S2 Read2 QScore distribution

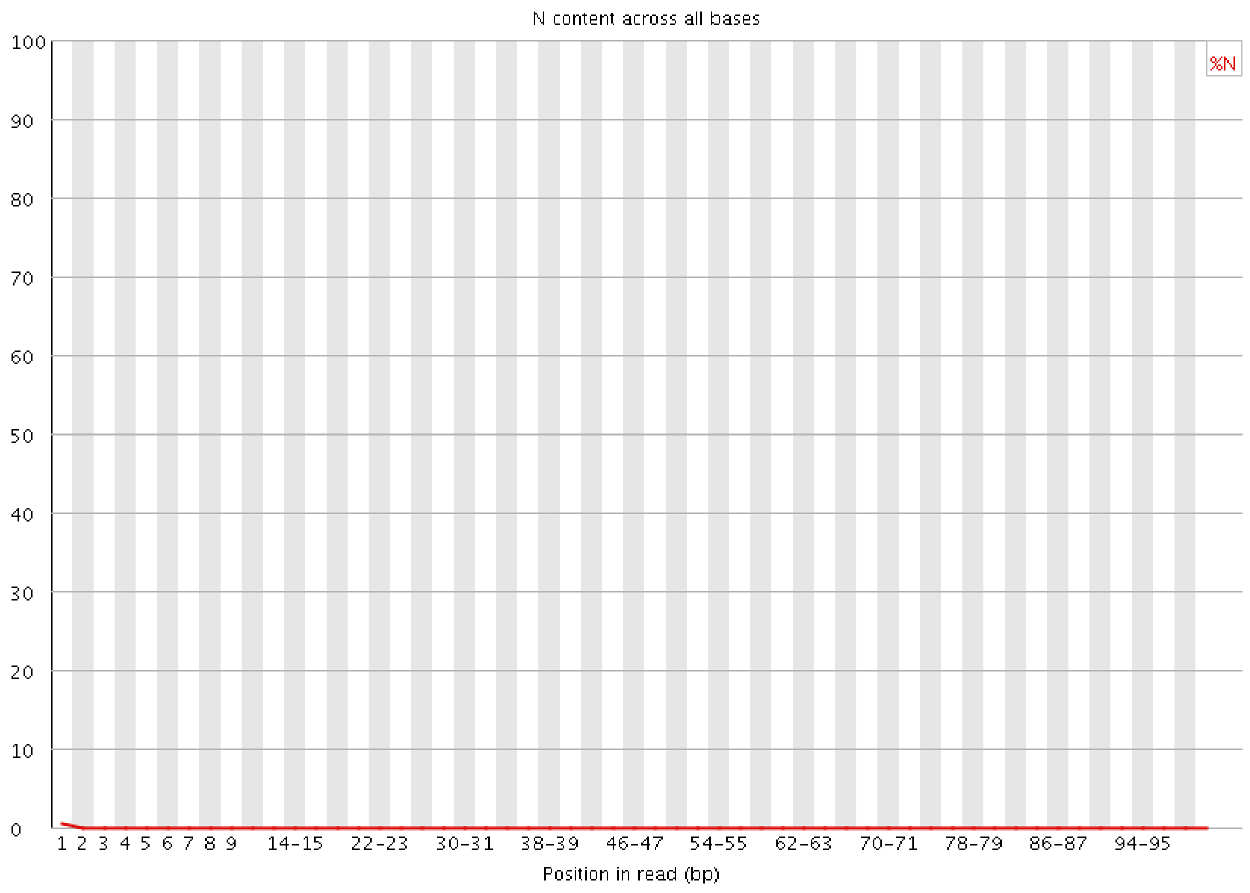

Figure 3: 29_4E_fox_S2 Read 1 Per-base N-content

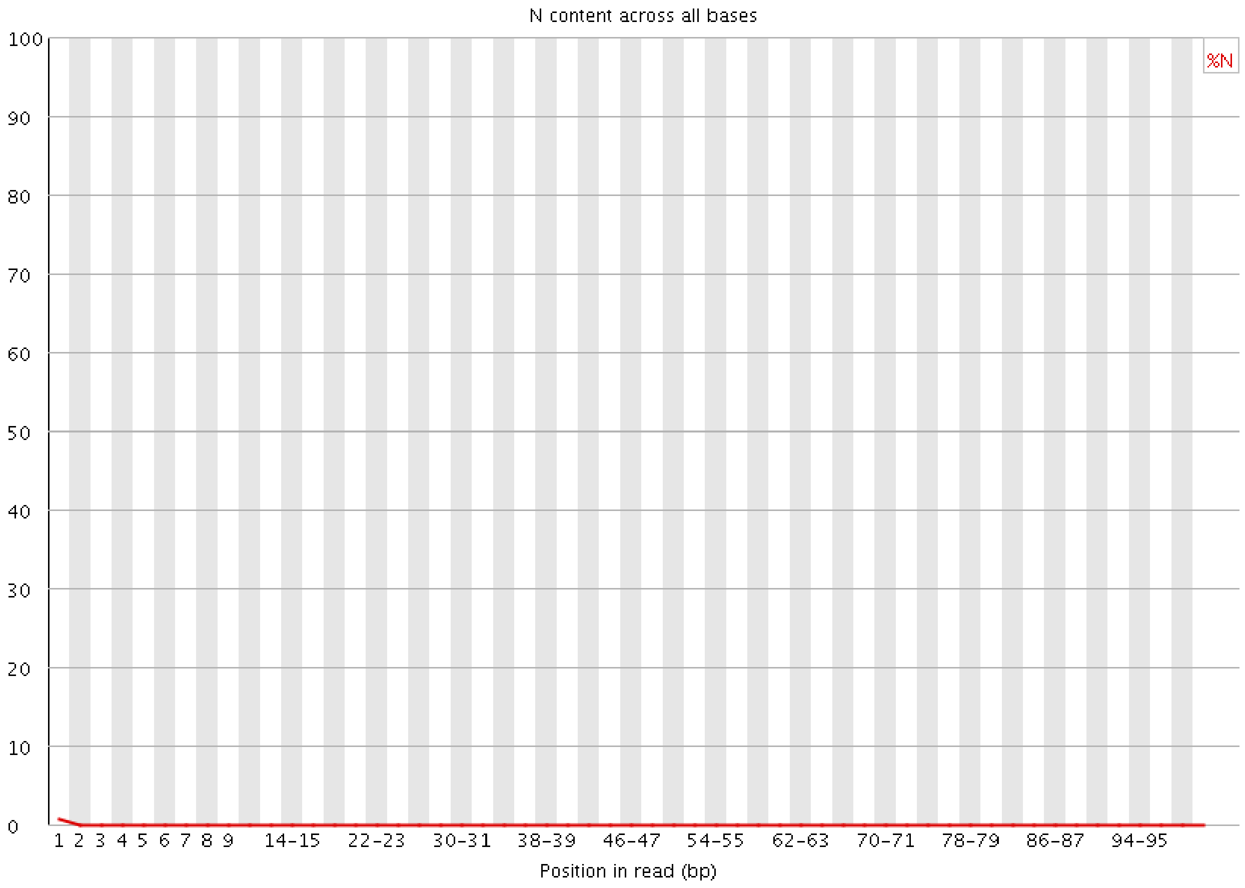

Figure 4: 29_4E_fox_S2 Read 2 Per-base N-content

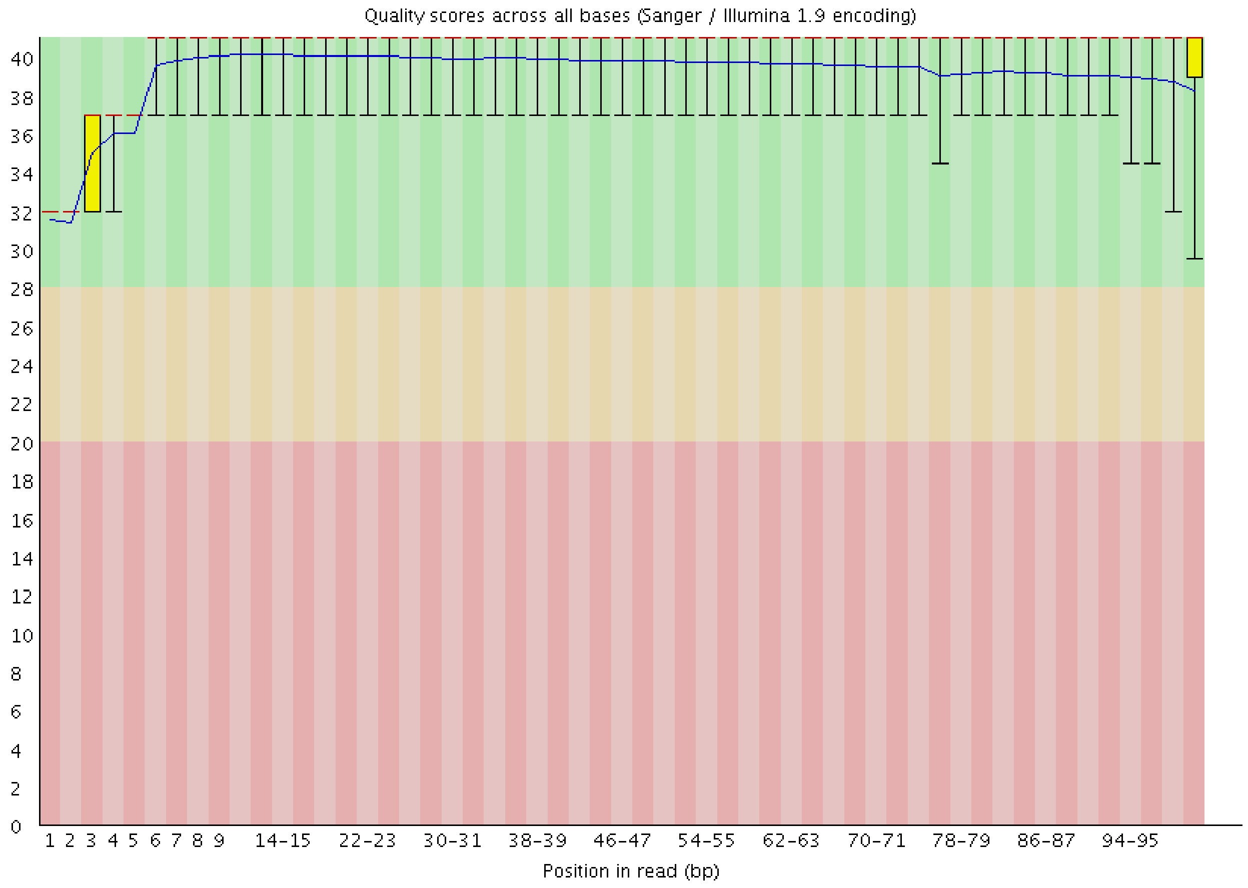

Figure 5: 2_2B_control_S2 Read1 QScore distribution

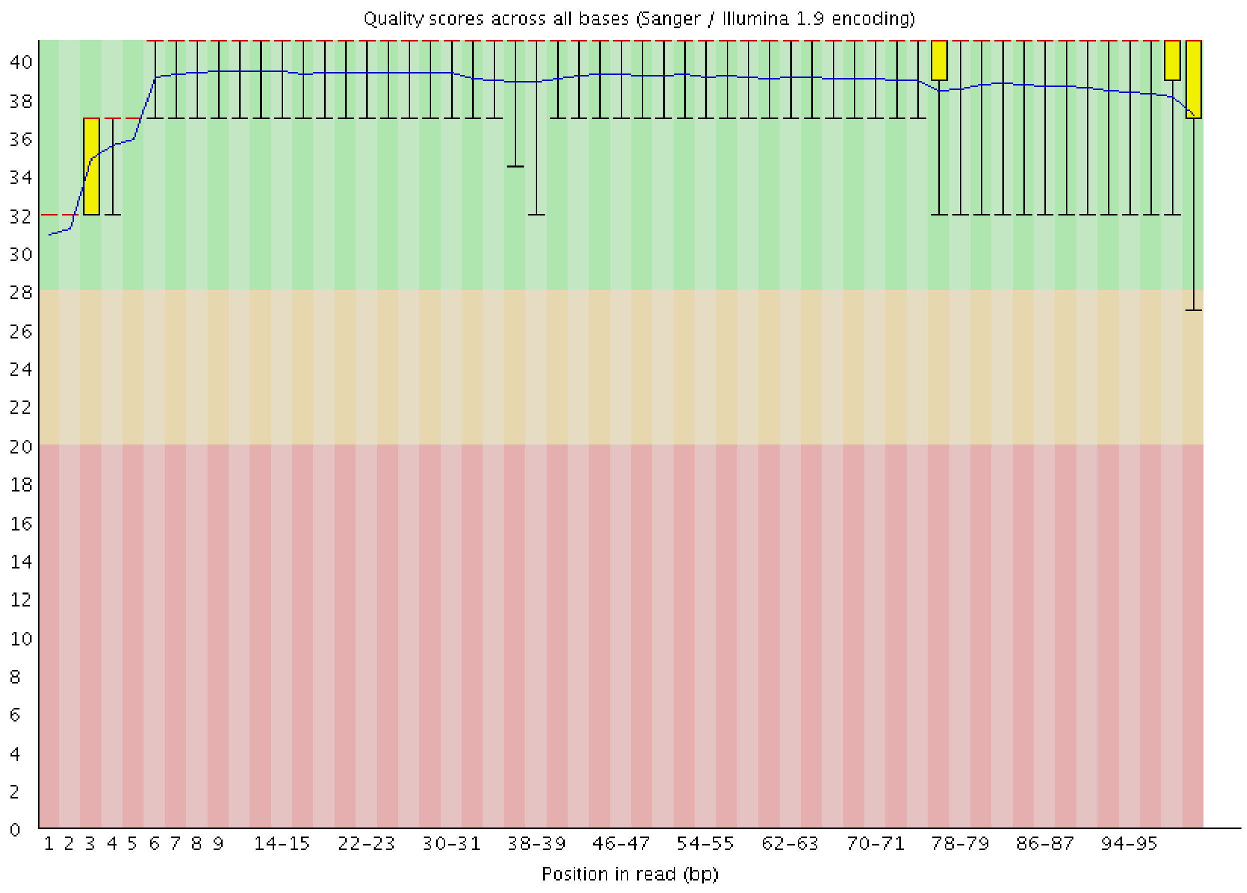

Figure 6: 2_2B_control_S2 Read2 QScore distribution

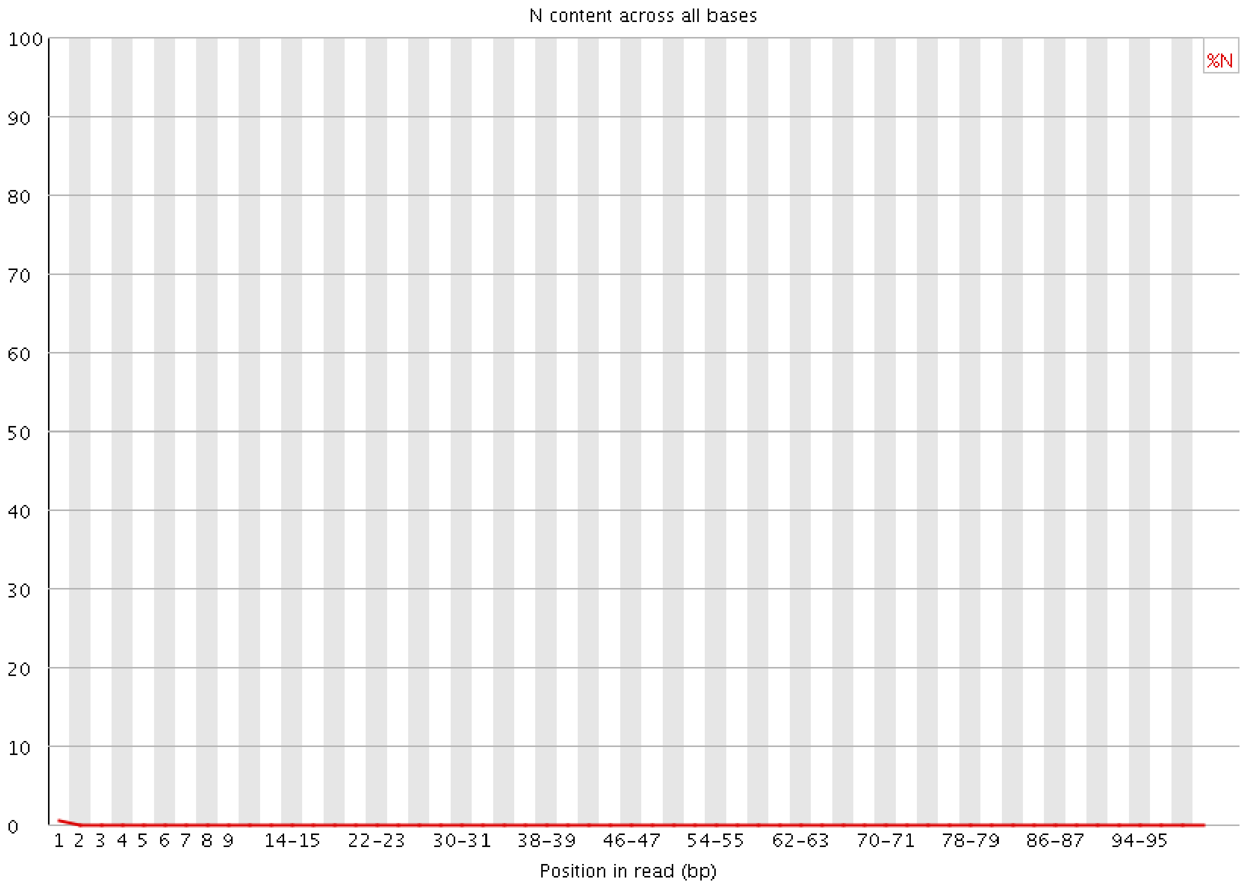

Figure 7: 2_2B_control_S2 Per-base N-content

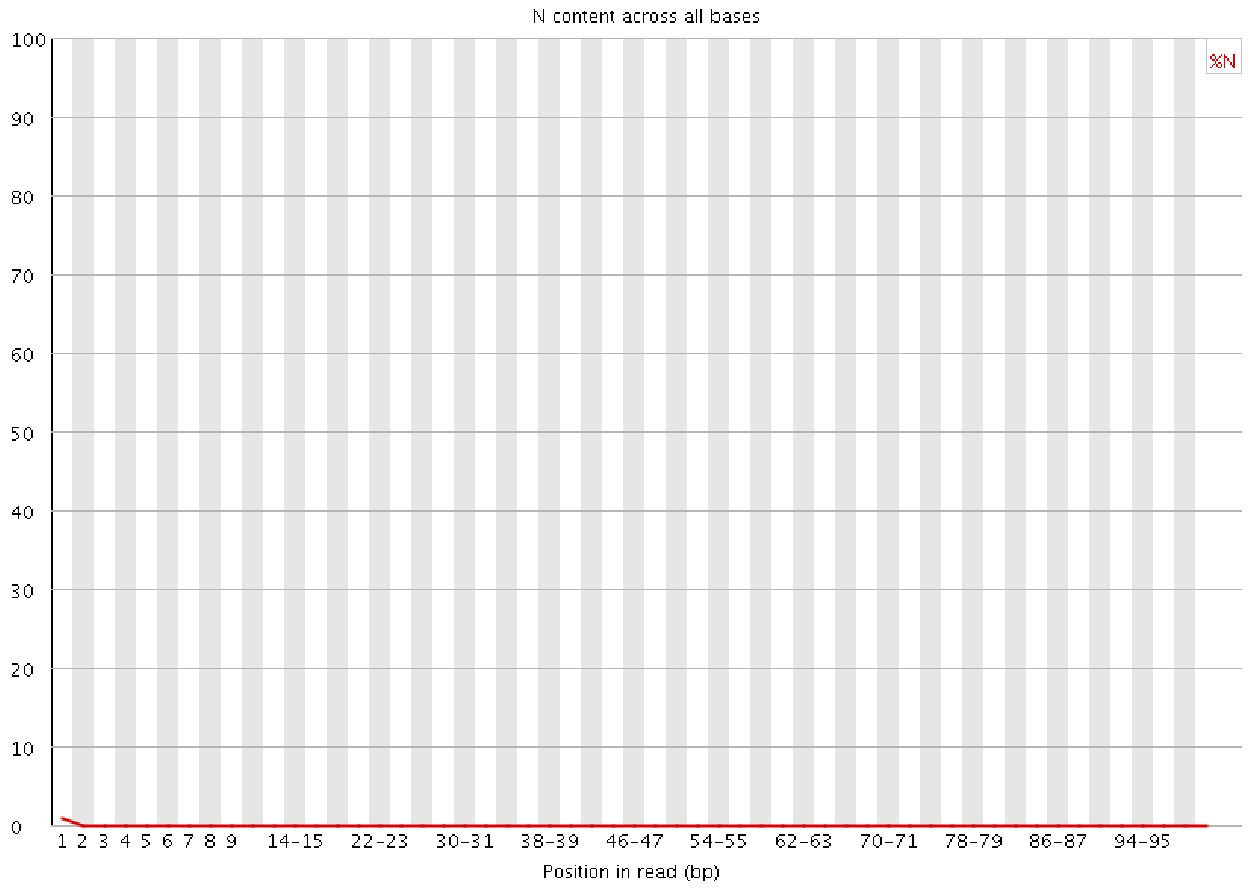

Figure 8: 2_2B_control_S2 Read2 Per-base N-content

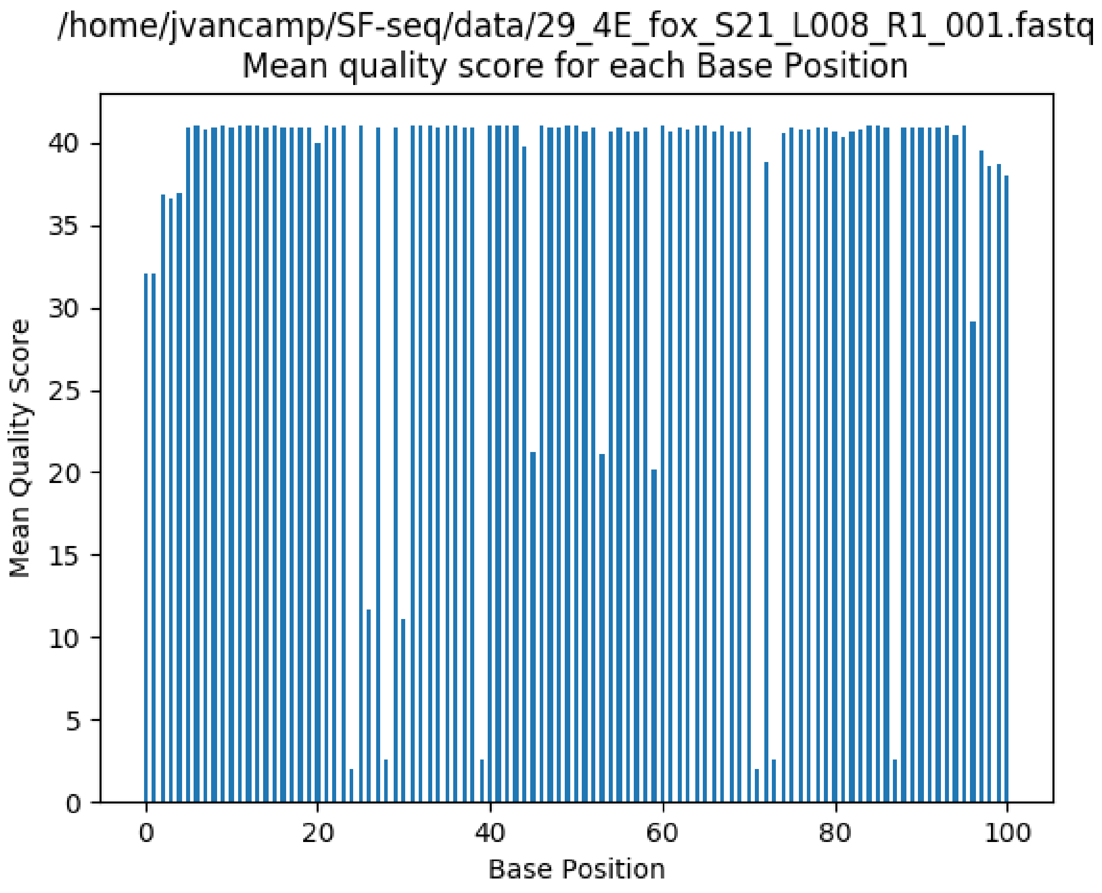

Figure 9: 29_4E_fox_S21 Read1 QScore distribution (My Script)

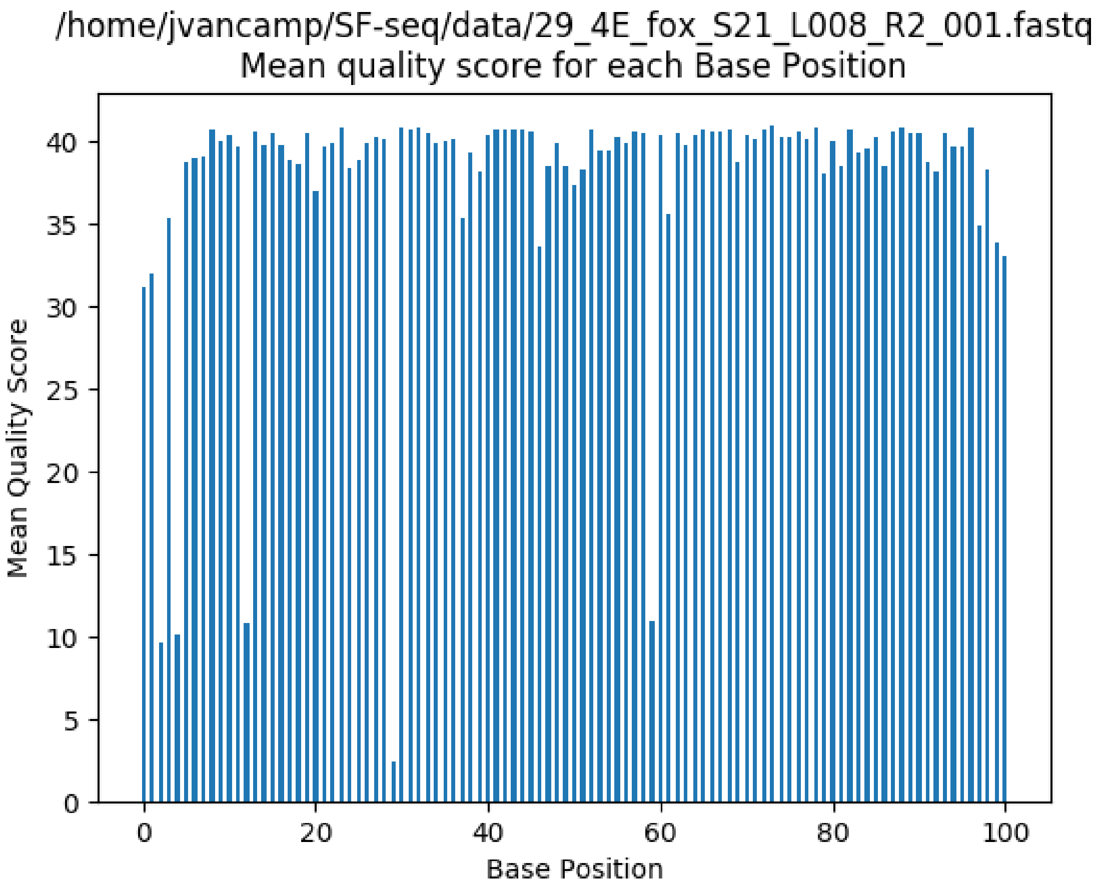

Figure 10: 29_4E_fox_S21 Read2 QScore distribution (My Script)

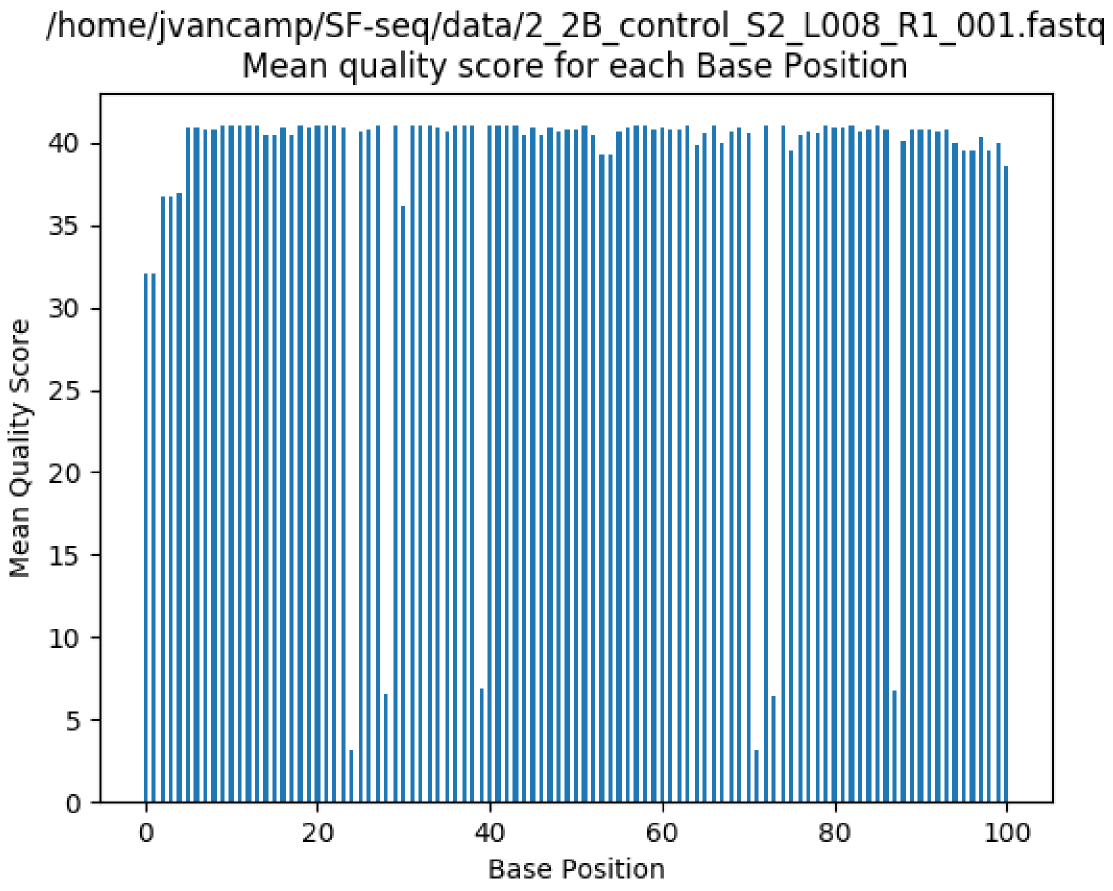

Figure 11: 2__2B__control__S2 Read1 QScore distribution (My Script)

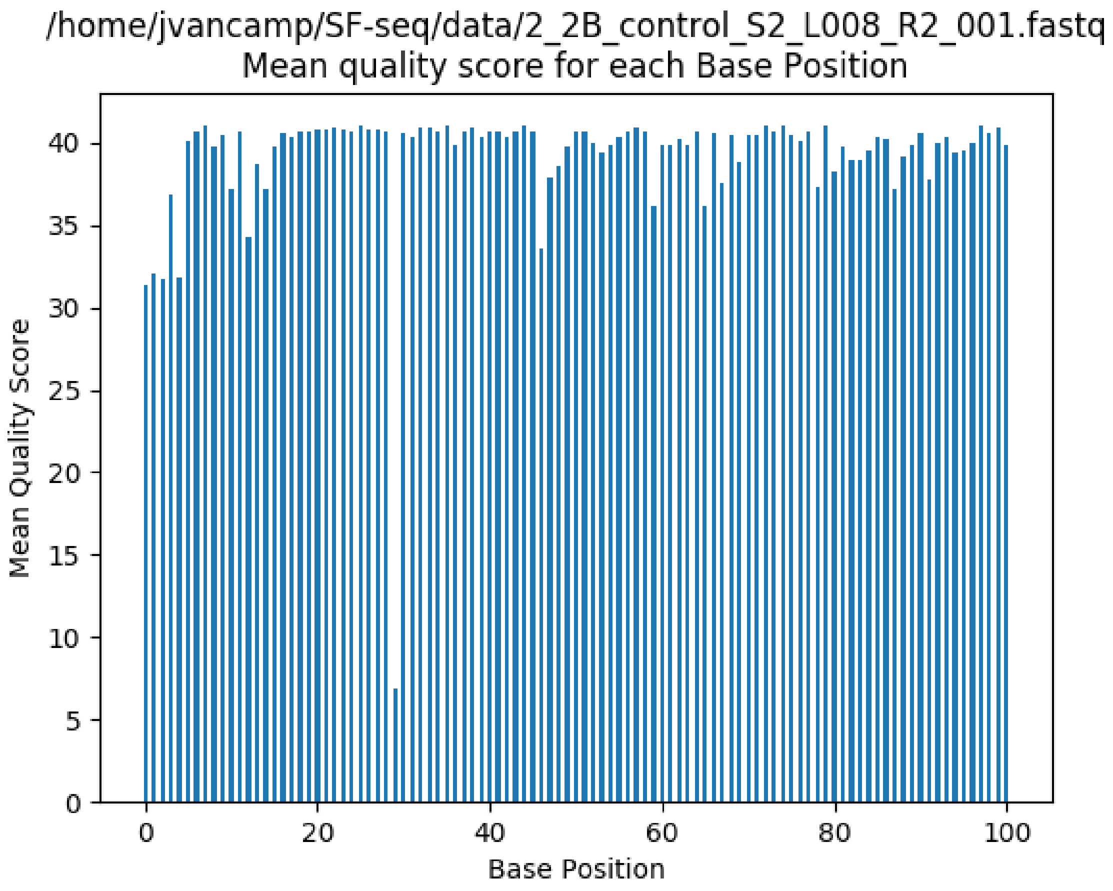

Figure 12: 2__2B__control__S2 Read2 QScore distribution (My Script)