QAA

Dillon Brownell

9/3/2021

Contents

Filename shorthand . . . 1
Part 1 – Read quality score distributions . . . 2
Part 2 – Adaptor trimming comparison . . . 4
Part 3 – Alignment and strand-specificity . . . 7
FASTQC OUTPUTS . . . 10
Figure outputs for 10_2G_both_S8_L008_R1_001.fastq.gz . . . 10
Figure outputs for 10_2G_both_S8_L008_R2_001.fastq.gz . . . 16
Figure outputs for 22_3H_both_S16_L008_R1_001.fastq.gz . . . 22
Figure outputs for 22_3H_both_S16_L008_R2_001.fastq.gz . . . 28
SCRIPTS AND SCRIPT OUTPUTS . . . 33
Demultiplexing The First sctipt and outputs . . . 33
STAR_BASH_COMMAND . . . 39
Aligned_vs_unaligned . . . 41
Trimmed file length histograms and script . . . 42

Filename shorthand

I will be referring to each of the assigned fastq files by the following in my written answers:

22_3H_both_S16_L008_R1_001.fastq.gz will be referred to as 22R1
22_3H_both_S16_L008_R2_001.fastq.gz will be referred to as 22R2
10_2G_both_S8_L008_R1_001.fastq.gz will be referred to as 10R1
10_2G_both_S8_L008_R2_001.fastq.gz will be referred to as 10R2

There are clickable elements available in the downloaded pdf version that will ease your reading experience!

Part 1 – Read quality score distributions

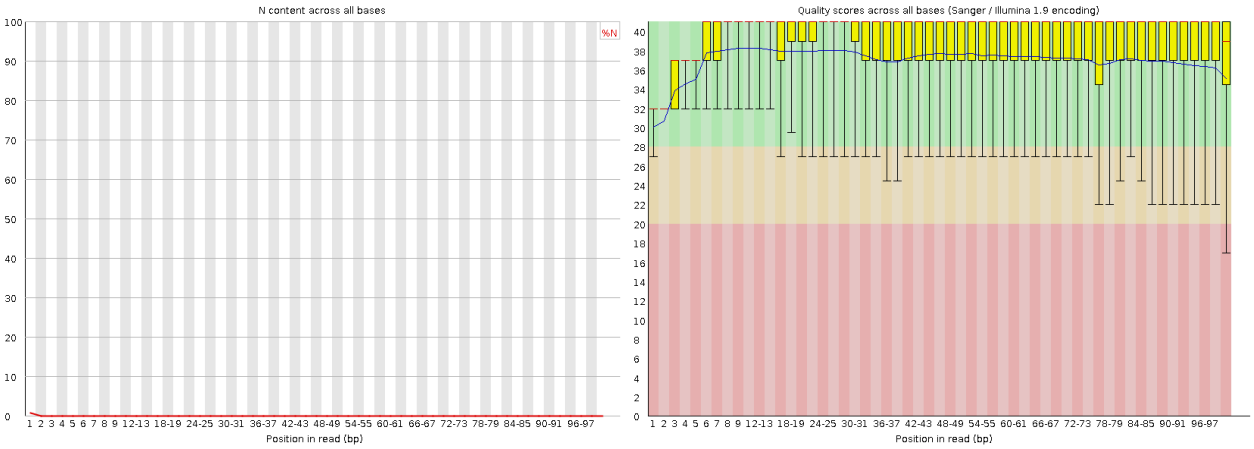
Using FastQC via the command line on Talapas, produce plots of quality score distributions for R1 and R2 reads. Also, produce plots of the per-base N content, and comment on whether or not they are consistent with the quality score plots.
The following command was used to generate FASTQC results...

fastqc -f fastq -o . /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_001.fastq.gz
/projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz
/projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_both_S8_L008_R1_001.fastq.gz
/projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_both_S8_L008_R2_001.fastq.gz

When comparing each quality score distribution (10R1_per_base_quality, 10R2_per_base_quality, 22R1_per_base_quality, 22R2_per_base_quality) to its respective per-base N content (10R1_per_base_n_content, 10R2_per_base_n_content, 22R1_per_base_n_content, 22R2_per_base_n_content), it is clear that Qscore across all bases and N across all bases have a relationship. This relationship is most easily observed in the R2 file-originating qscore distributions (10R2_per_base_quality, 22R2_per_base_quality) and their respective n-content distributions (10R2_per_base_n_content, 22R2_per_base_n_content) because of their lower overall levels of quality.

On average across all reads in all files, Qscore is typically low within the first 10 positions, and lowest at the very first base. On average across all reads in all files, N content is highest at the first position, evidenced by a small >1% increase.(~0.6 % in R1 files, ~0.95 % in R2 files).

See here the relationship between N content of 10R2 and quality of 10R2:



Run your quality score plotting script from your Demultiplexing assignment from Bi622. Describe how the FastQC quality score distribution plots compare to your own. If different, propose an explanation. Also, does the runtime differ? If so, why?

As far as the general shape, the graphs seem to be in agreement. There do not appear to be obvious differences resulting from disparities in raw data or processing of raw data. If small differences do exist, I would expect them to be due to differnt handling of N quality scores.

However, the FASTQC graphs differ most significantly from my own in terms of actual presentation. While my graphs (TF_10R1, TF_10R2, TF_22R1, TF_22R2) display average quality across per base as simple bar graphs, FASTQC offers additional information.

Taking 10R1_per_base_quality as an example, and consulting information found at here, the FASTQC graphs appear to show information on median, interquartile range, 10% and 90% percentile, average, and also includes a color guide for general levels of quality.

The runtime of my graphing script (Demultiplexing The First sctipt) varied quite a bit depending on the particular input file. When run on 10R1 and 10R2, the runtime was an immense 2800 seconds. (Demultiplexing The First runtime information + slurm output for 10R1, 10R2) When run on 22R1 and 22R2, the runtime was a short 172 seconds. (Demultiplexing The First runtime information + slurm output for 22R1, 22R2)

This difference may be attributable to sequence count. The fastqc report indicates that 22R1 and 22R2 each contain 4050899 total sequences while the 10R1 and 10R2 files contain 81477069 total sequences, constituting a 19.11 fold increase in sequences.

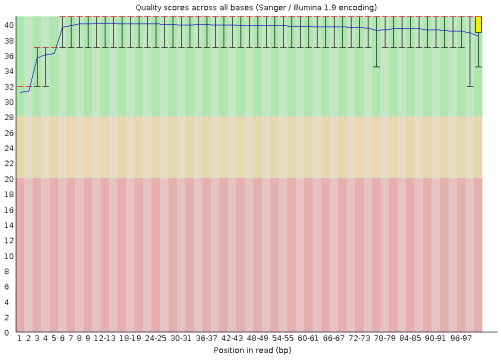
However, both of these runtimes were much longer than that of FASTQC, which finished in just a few seconds. I suspect these astonishing differences can be attributed to 2 main factors: programming skill and language.

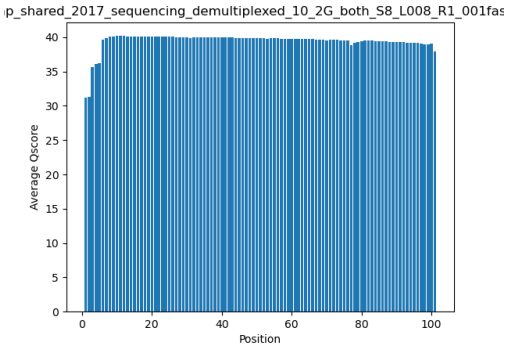
Fastqc is developed by a team of professional scientist and programming experts. I expect that they utilize software package tools which significantly boost their program runtime by organizing the data at hand differently than how I did, and because they are logical experts.

Additionally, the program is written in Java, a language well-known to outstrip Python on a basic level.

I make these comments in light of information found here

See here an easy-view comparison of graphs styles using 10R1_per_base_quality and TF_10R1:





Comment on the overall data quality of your two libraries.

Both libraries have good data quality. There is little N content in each. Qscores, though worse in each R2, are in the green. None have overrepresented sequences or adapter content. All have some level of sequence duplication and particular enrichment of a kmer. These are more obvious in ([10R2_duplication_levels10R2_kmer_profiles]) and ([22R2_duplication_levels10R2_kmer_profiles]). I suspect these are caused by untrimmed adapters.

Overall, good quality.

Part 2 – Adaptor trimming comparison

Create a new conda environment called QAA and install cutadapt and Trimmomatic. Google around if you need a refresher on how to create conda environments. Make sure you check your installations with:

- cutadapt –version (should be 3.4)
- trimmomatic -version (should be 0.39)

In /home/dbrowne2/bgmp/bioinfo/Bi623/Assignments/Assignment_QAA

Commands used in order...

conda create –name QAA python=3.9

conda activate QAA

module spider trimmomatic

module spider cutadapt

conda install trimmomatic

trimmomatic -version

conda install cutadapt

cutadapt –version

Using cutadapt, properly trim adapter sequences from your assigned files. Be sure to read how to use cutadapt. Use default settings. What proportion of reads (both forward and reverse) were trimmed?

See below.

Try to determine what the adapters are on your own. If you cannot (or if you do, and want to confirm), click here to see the actual adapter sequences used.

- Sanity check: Use your Unix skills to search for the adapter sequences in your datasets and confirm the expected sequence orientations. Report the commands you used, the reasoning behind them, and how you confirmed the adapter sequences.

I used Illumina's guide to adapter sequence trimming found here and confirmed using the dropdown to identify my adapter sequences.

zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_001.fastq.gz | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc -l reports 7563 sequences containing the full R1 adapter in 22R1 and 23922 when instead used on 10R1. These mostly fall closer to the 3' end if viewed with out wc -l.

zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -l reports 7848 sequences containing the full R2 adapter in 22R2 and 31424 when instead used on 10R2.

Testing R1 adapters on R2 files and vis versa reveals zero matches.

cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTG-TAGGGAAAGAGTGT -o ./22R1_cut.fastq.gz -p ./22R2_cut_.fastq.gz /projects/bgmp/shared/2017_sequencing/demultipl /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz

=== Summary ===

Total read pairs processed: 4,050,899
Read 1 with adapter: 153,089 (3.8%)
Read 2 with adapter: 186,534 (4.6%)
Pairs written (passing filters): 4,050,899 (100.0%)

cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTG-TAGGGAAAGAGTGT -o ./10R1_cut.fastq.gz -p ./10R2_cut_.fastq.gz /projects/bgmp/shared/2017_sequencing/demultipl /projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_both_S8_L008_R2_001.fastq.gz

=== Summary ===

Total read pairs processed: 81,477,069 Read 1 with adapter: 2,131,954 (2.6%)
Read 2 with adapter: 2,770,901 (3.4%)
Pairs written (passing filters): 81,477,069 (100.0%)

Use Trimmomatic to quality trim your reads. Specify the following, in this order:

- LEADING: quality of 3
- TRAILING: quality of 3
- SLIDING WINDOW: window size of 5 and required quality of 15
- MINLENGTH: 35 bases

trimmomatic PE -phred33 22R1_cut.fastq.gz 22R2_cut_.fastq.gz 22R1_cut_trimmed_paired.fastq.gz 22R1_cut_trimmed_unpaired.fastq.gz 22R2_cut_trimmed_paired.fastq.gz 22R2_cut_trimmed_unpaired.fastq.gz LEADING:3 TRAILING:3 SLIDINGWINDOW:5:15 MINLEN:35

Input Read Pairs: 4050899 Both Surviving: 3901127 (96.30%) Forward Only Surviving: 145063 (3.58%) Reverse Only Surviving: 2876 (0.07%) Dropped: 1833 (0.05%)

trimmomatic PE -phred33 10R1_cut.fastq.gz 10R2_cut_.fastq.gz 10R1_cut_trimmed_paired.fastq.gz 10R1_cut_trimmed_unpaired.fastq.gz 10R2_cut_trimmed_paired.fastq.gz 10R2_cut_trimmed_unpaired.fastq.gz LEADING:3 TRAILING:3 SLIDINGWINDOW:5:15 MINLEN:35

Input Read Pairs: 81477069 Both Surviving: 77520904 (95.14%) Forward Only Surviving: 3865386 (4.74%) Reverse Only Surviving: 53289 (0.07%) Dropped: 37490 (0.05%)

Be sure to output compressed files and clear out any intermediate files.

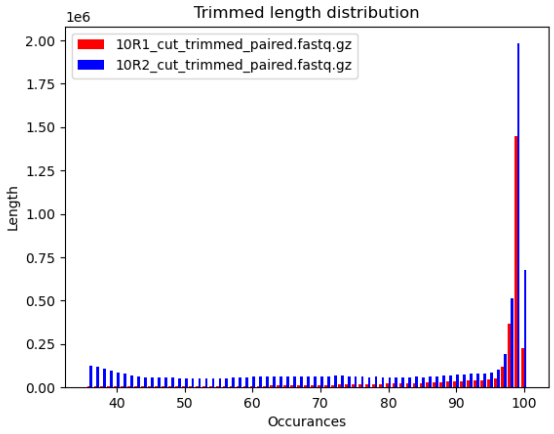
Plot the trimmed read length distributions for both R1 and R2 reads (on the same plot). You can produce 2 different plots for your 2 different RNA-seq samples. There are a number of ways you could possibly do this. One useful thing your plot should show, for example, is whether R1s are trimmed more extensively than R2s, or vice versa. Comment on whether you expect R1s and R2s to be adapter-trimmed at different rates.

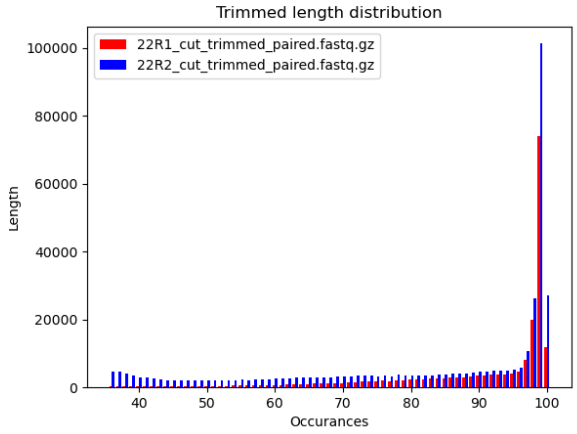
Plots at (10R1R2 histogram) and (22R1R2 histogram)

I would expect quality trimming to more heavily affect R2s due to chemical degradation.

As for adapter trimming, I do not know a particular reason to suspect a differnce outside of chemical degradation and temporal separation. I'm not sure if 3' bias plays a role.

See below miniaturized plots:





Part 3 – Alignment and strand-specificity

Install sofware. In your QAA environment, use conda to install:

- star
- numpy
- pysam
- matplotlib
- Then pip install HTSeq

Find publicly available mouse genome fasta files (Ensemble release 104) and generate an alignment database from them. Align the reads to your mouse genomic database using a splice-aware aligner. Use the settings specified in PS8 from Bi621.

- Hint - you will need to use gene models to perform splice-aware alignment, see PS8 from Bi621.

wget http://ftp.ensembl.org/pub/release-104/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz
wget http://ftp.ensembl.org/pub/release-104/gtf/mus_musculus/Mus_musculus.GRCm39.104.gtf.gz
Used STAR_BASH_COMMAND to generate sam files.

Using your script from PS8 in Bi621, report the number of mapped and unmapped reads from each of your 2 sam files. Make sure that your script is looking at the bitwise flag to determine if reads are primary or secondary mapping (update your script if necessary).

The program used can be found here Aligned_vs_unaligned

python counter2.py -f cutzone/10R1R2_vs_MouseAligned.out.sam

152719556 mapped
2322252 unmapped

python counter2.py -f cutzone/22R1R2_vs_MouseAligned.out.sam

7677904 mapped
124350 unmapped

Count reads that map to features using htseq-count. You should run htseq-count twice: once with –stranded=yes and again with –stranded=no. Use default parameters otherwise. Before running using default parameters, we muse sort out sam files by name.

module load samtools/1.5

samtools sort -n -O sam -o 10R1R2_vs_MouseAligned_sortedbyname 10R1R2_vs_MouseAligned.out.sam

samtools sort -n -O sam -o 22R1R2_vs_MouseAligned_sortedbyname 22R1R2_vs_MouseAligned.out.sam

htseq-count –stranded=yes 10R1R2_vs_MouseAligned_sortedbyname.sam Mus_musculus.GRCm39.104.gtf

___no_feature 69853236
___ambiguous 48198
___too_low_aQual 136867
___not_aligned 1088394
___alignment_not_unique 3493273

htseq-count –stranded=no 10R1R2_vs_MouseAligned_sortedbyname.sam Mus_musculus.GRCm39.104.gtf

___no_feature 3771976 ___ambiguous 4004790
___too_low_aQual 136867
___not_aligned 1088394
___alignment_not_unique 3493273

htseq-count –stranded=yes 22R1R2_vs_MouseAligned_sortedbyname.sam Mus_musculus.GRCm39.104.gtf

___no_feature 3524280 ___ambiguous 2301
___too_low_aQual 4982
___not_aligned 59545 ___alignment_not_unique 167705

htseq-count –stranded=no 22R1R2_vs_MouseAligned_sortedbyname.sam Mus_musculus.GRCm39.104.gtf

___no_feature 188032
___ambiguous 192536
___too_low_aQual 4982
___not_aligned 59545
___alignment_not_unique 167705

Demonstrate convincingly whether or not the data are from "strand-specific" RNA-Seq libraries. Include any comands/scripts used. Briefly describe your evidence, using quantitative statements (e.g. "I propose that these data are/are not strand-specific, because X% of the reads are y, as opposed to z."). Working with 10R1R2_stranded_htseq.txt and 10R1R2_unstranded_htseq.txt, I ripped code from ICA4 to determine the following...

cat 10R1R2_stranded_htseq.txt | awk '{total += $2} END {print total}' and cat 10R1R2_unstranded_htseq.txt | awk '{total += $2} END {print total}' shows that each file has 77520904 reads total.

cat 10R1R2_unstranded_htseq.txt | awk '$1 ~ /EN/ {mapped += $2} END {print mapped}' shows that the unstranded htseq has 65025604 mapping reads, which is 83.88% of the total

cat 10R1R2_stranded_htseq.txt | awk '$1 ~ /EN/ {mapped += $2} END {print mapped}' shows that the stranded htseq has 2900936 mapping reads, which is 3.742% of the total

I propose that these data are not strand-specific, because 83.88% of the reads map in unstranded htseq as opposed to 3.742% in stranded htseq.

It is my reasoning that, were the data stranded, there would be much less difference between these percents. I propose that many reads match when handled as non-stranded and far less match when handled as stranded because handling them together does not reflect the reality of the library preparation, thus decreasing specificity to the target in almost every case.

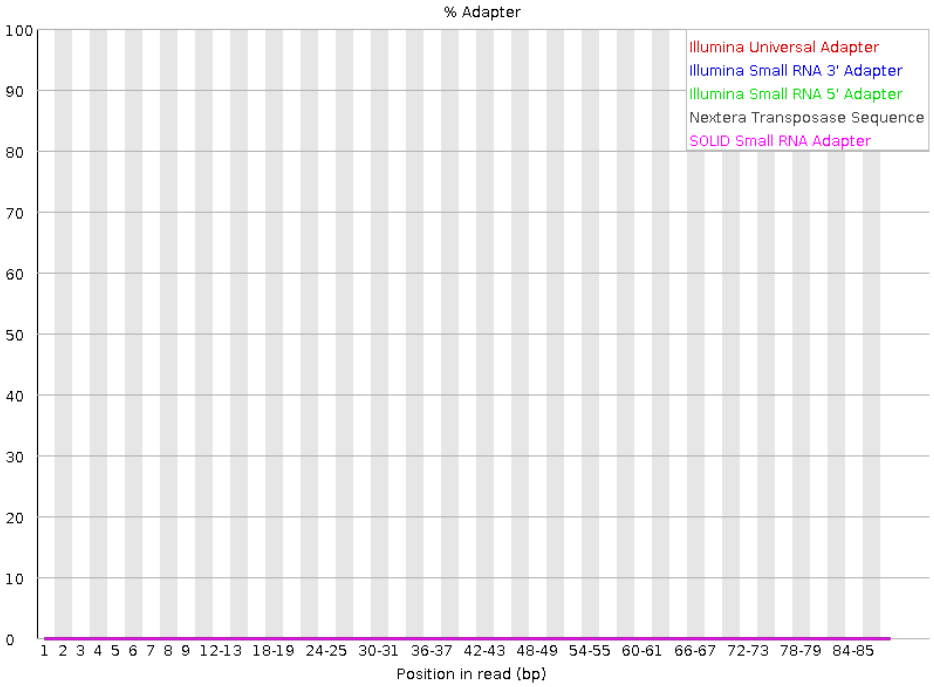
FASTQC OUTPUTS

Figure outputs for 10_2G_both_S8_L008_R1_001.fastq.gz

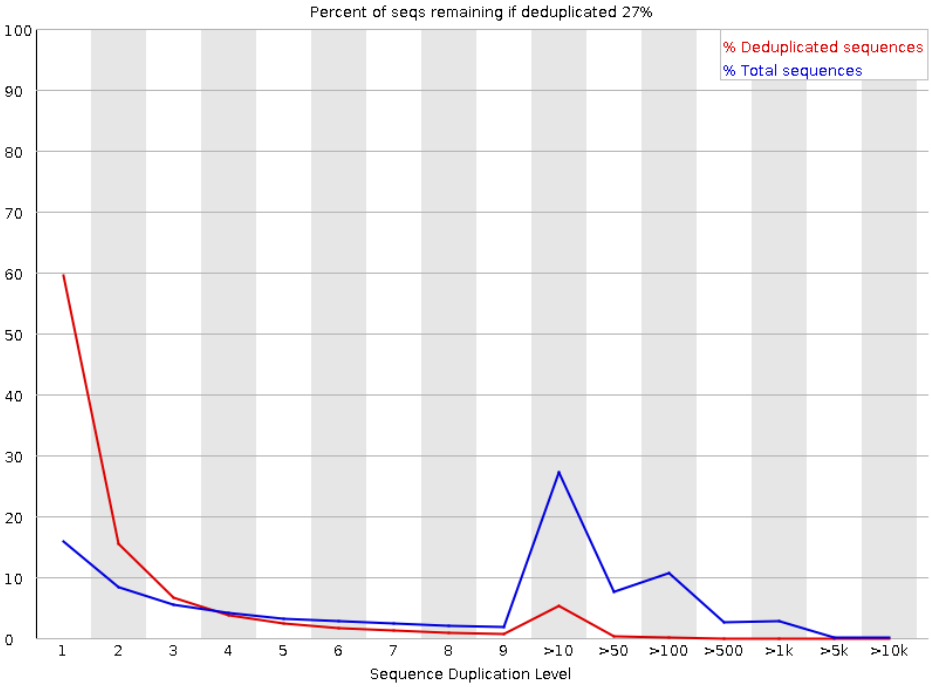
10_2G_both_S8_L008_R1_001.fastq.gz Summary

PASS Basic Statistics 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Per base sequence quality 10_2G_both_S8_L008_R1_001.fastq.gz
FAIL Per tile sequence quality 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Per sequence quality scores 10_2G_both_S8_L008_R1_001.fastq.gz
FAIL Per base sequence content 10_2G_both_S8_L008_R1_001.fastq.gz
WARN Per sequence GC content 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Per base N content 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Sequence Length Distribution 10_2G_both_S8_L008_R1_001.fastq.gz
FAIL Sequence Duplication Levels 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Overrepresented sequences 10_2G_both_S8_L008_R1_001.fastq.gz
PASS Adapter Content 10_2G_both_S8_L008_R1_001.fastq.gz
FAIL Kmer Content 10_2G_both_S8_L008_R1_001.fastq.gz

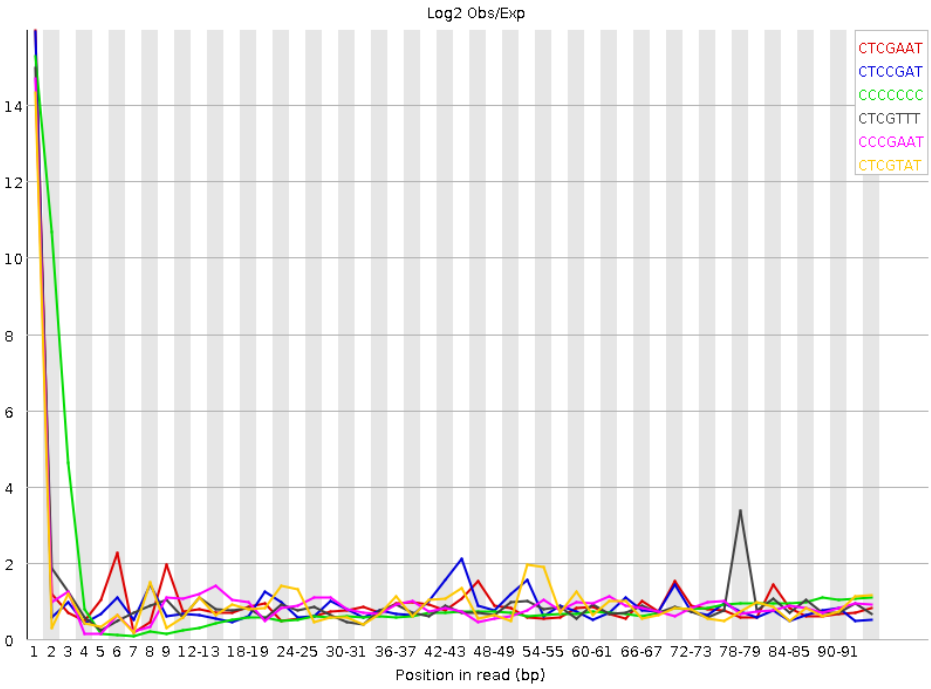
10R1_adapter_content



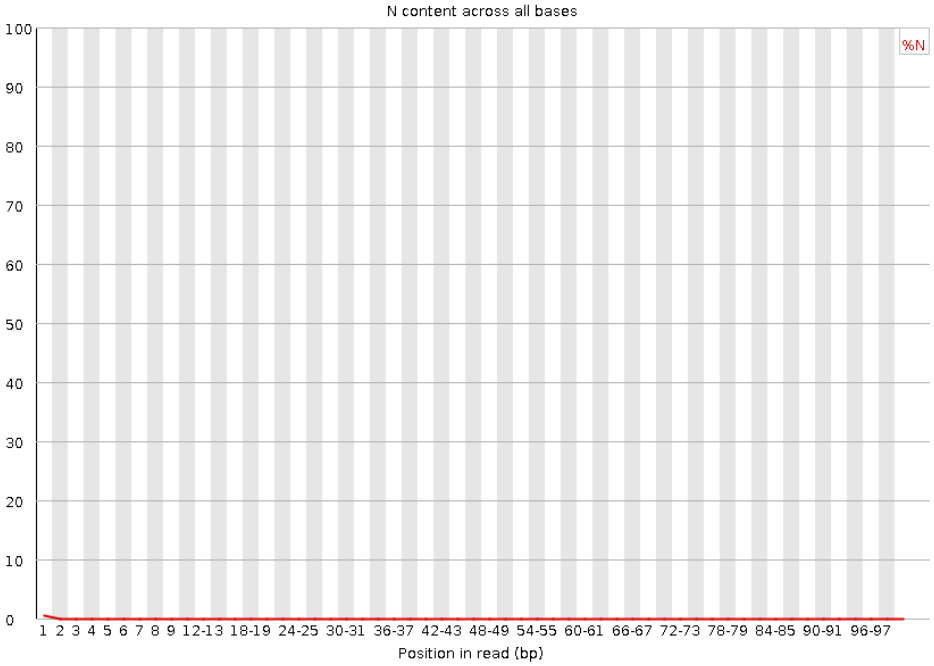
10R1_duplication_levels



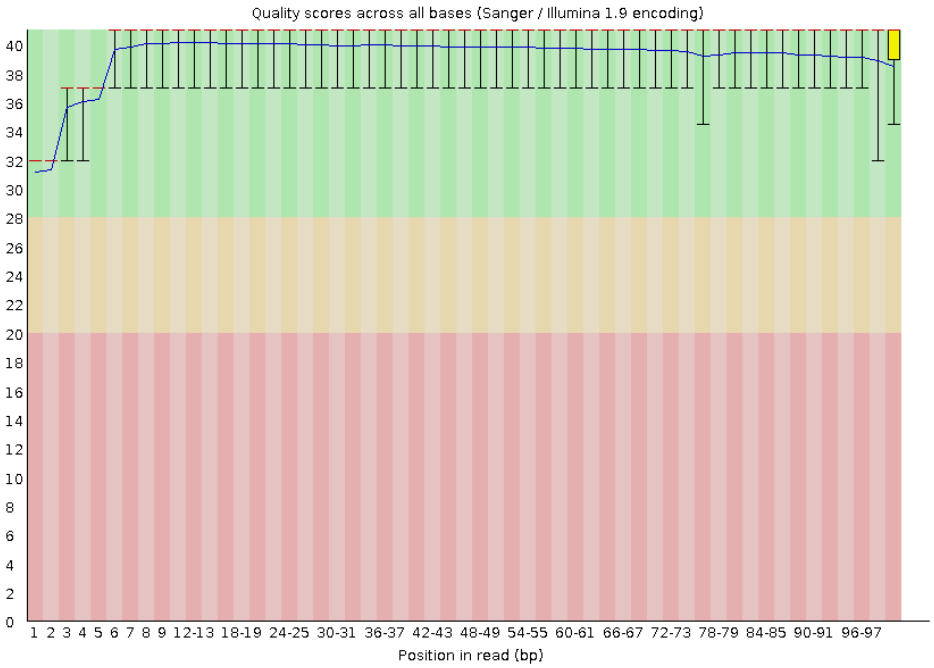
10R1_kmer_profiles



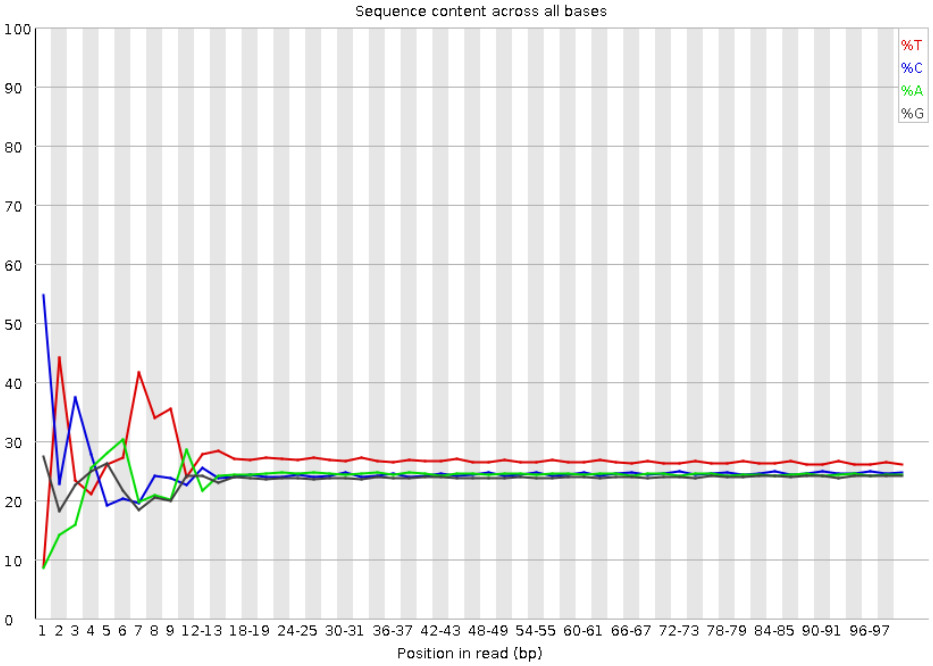
10R1__per__base__n__content



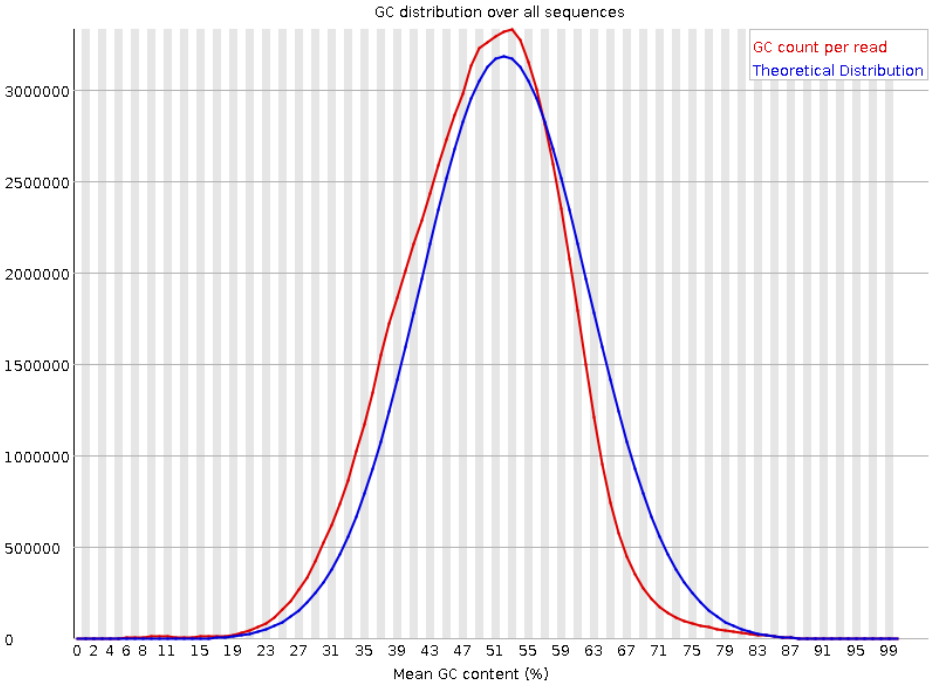
10R1__per__base__quality



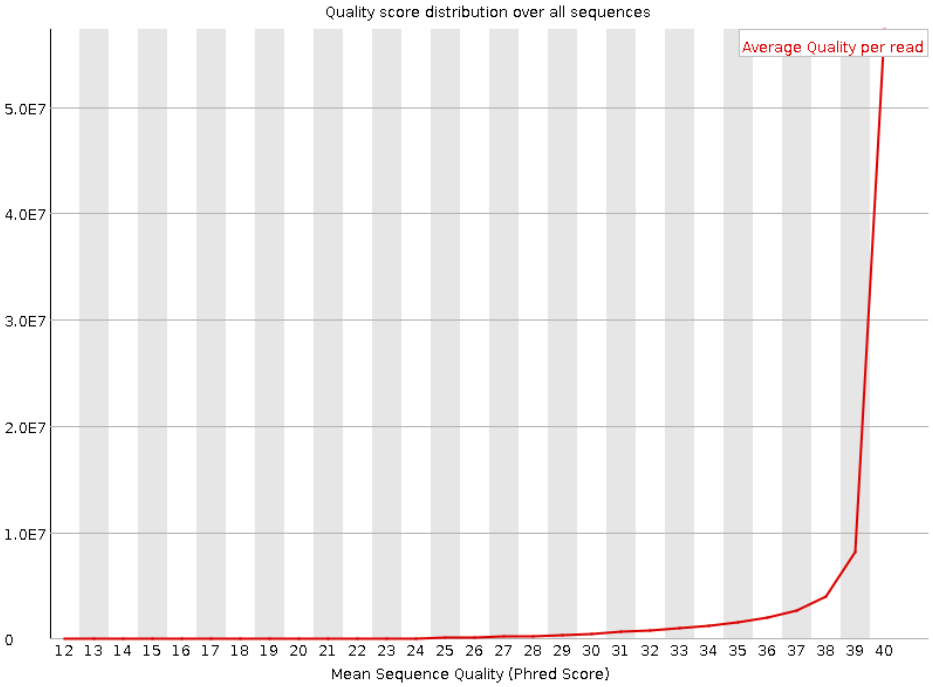
10R1__per__base__sequence__content



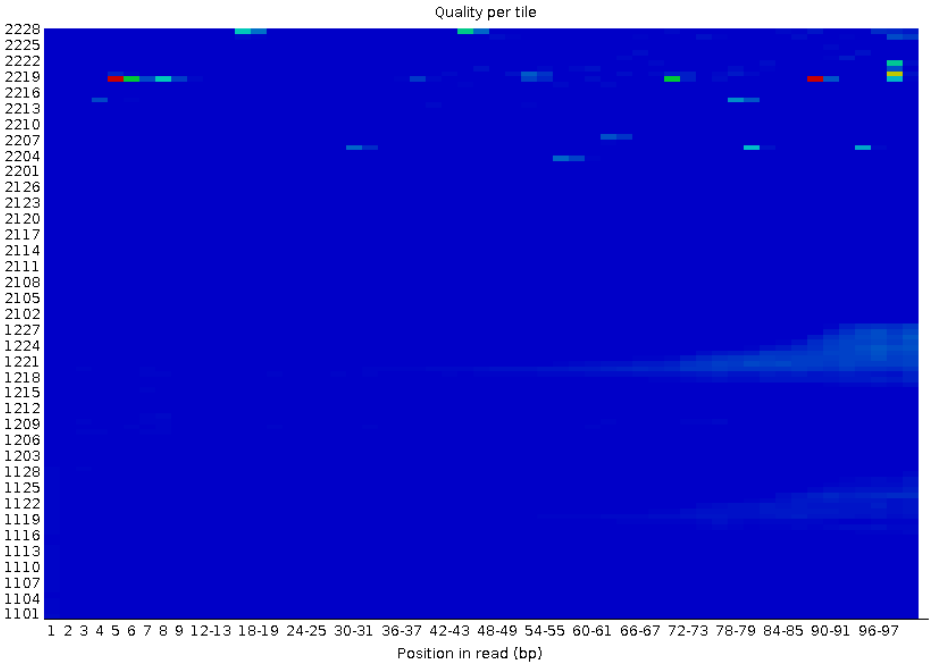
10R1__per__base__gc__content



10R1_per_sequence_quality



10R1_per_tile_quality



10_sequence_length_distribution

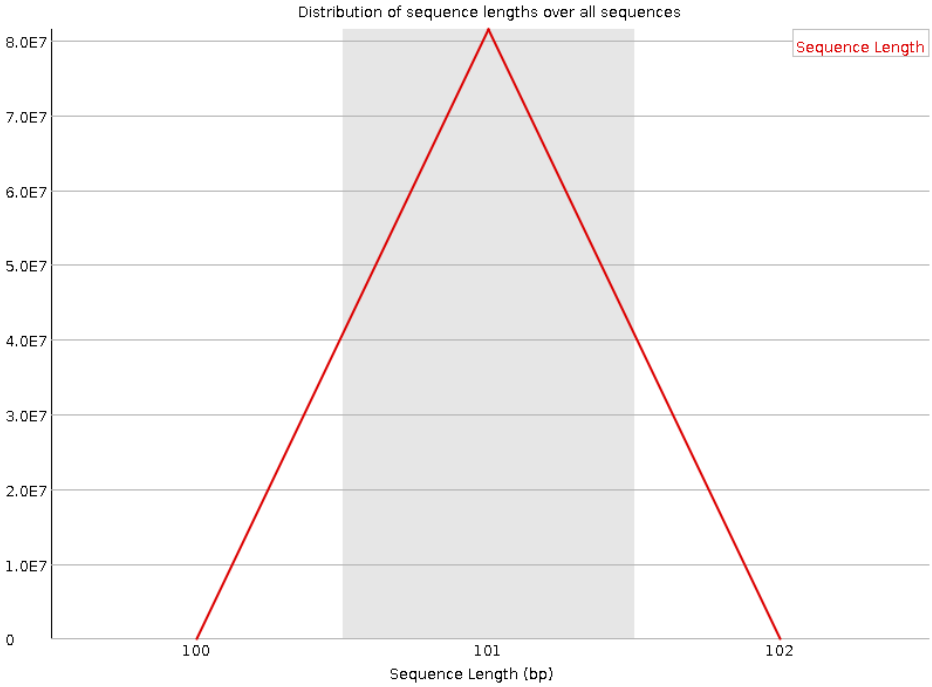
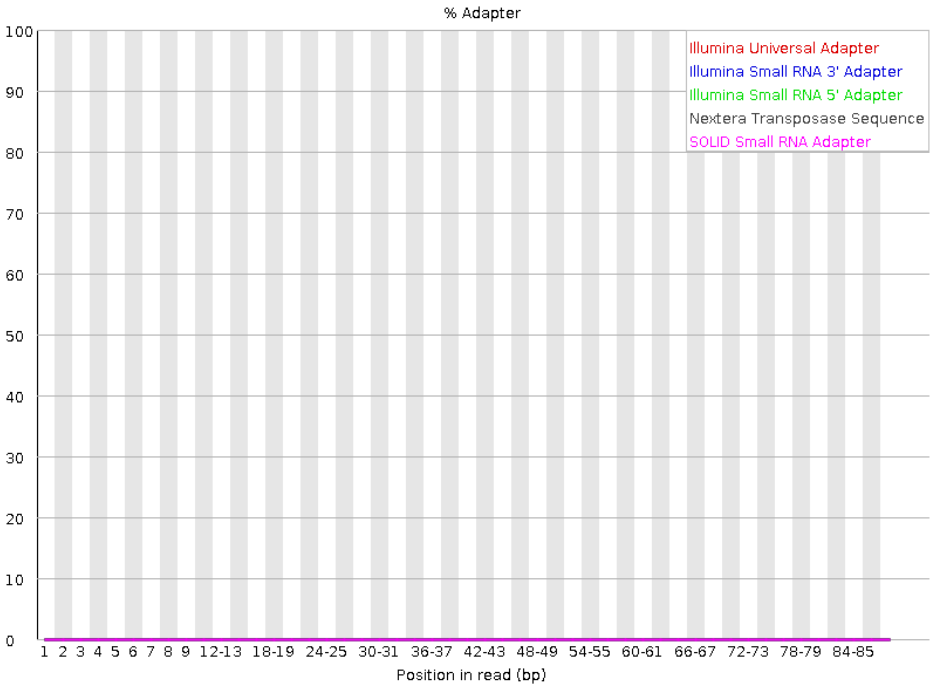


Figure outputs for 10_2G_both_S8_L008_R2_001.fastq.gz

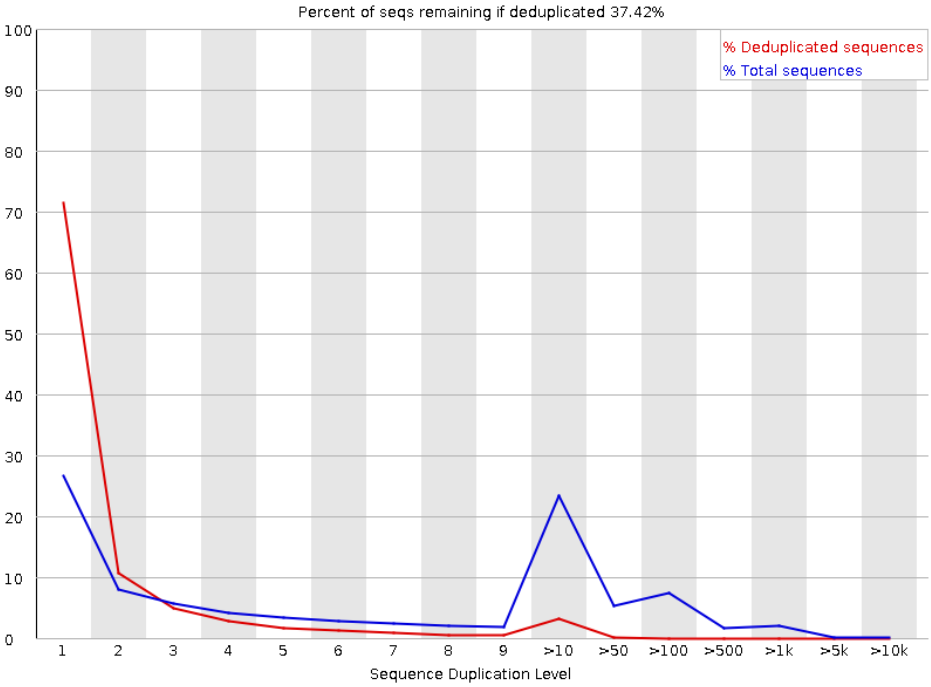
10_2G_both_S8_L008_R2_001.fastq.gz Summary

PASS Basic Statistics 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Per base sequence quality 10_2G_both_S8_L008_R2_001.fastq.gz
WARN Per tile sequence quality 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Per sequence quality scores 10_2G_both_S8_L008_R2_001.fastq.gz
FAIL Per base sequence content 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Per sequence GC content 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Per base N content 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Sequence Length Distribution 10_2G_both_S8_L008_R2_001.fastq.gz
FAIL Sequence Duplication Levels 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Overrepresented sequences 10_2G_both_S8_L008_R2_001.fastq.gz
PASS Adapter Content 10_2G_both_S8_L008_R2_001.fastq.gz
FAIL Kmer Content 10_2G_both_S8_L008_R2_001.fastq.gz

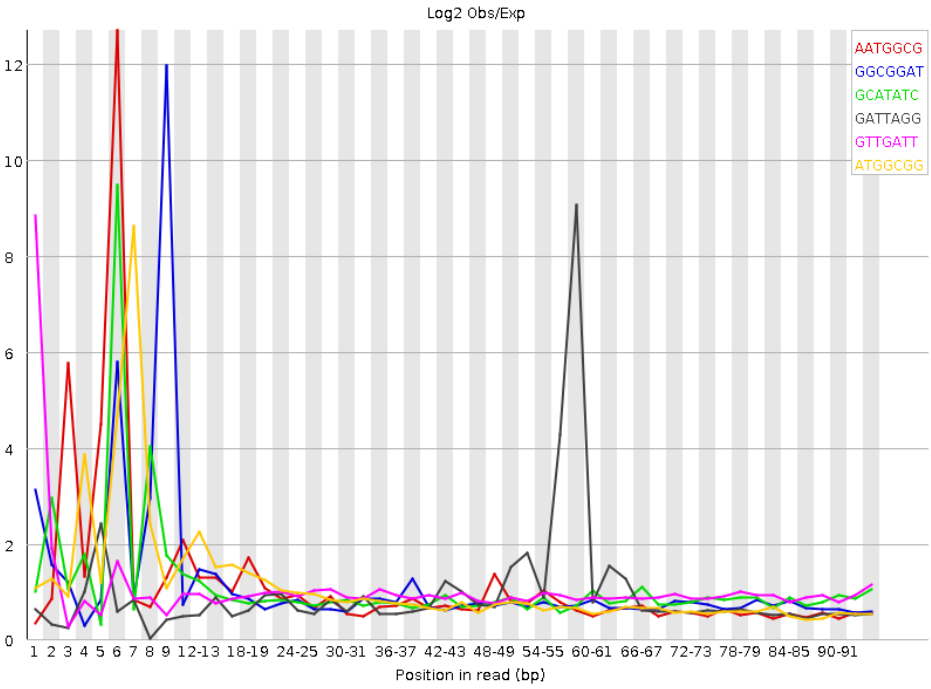
10R2_adapter_content



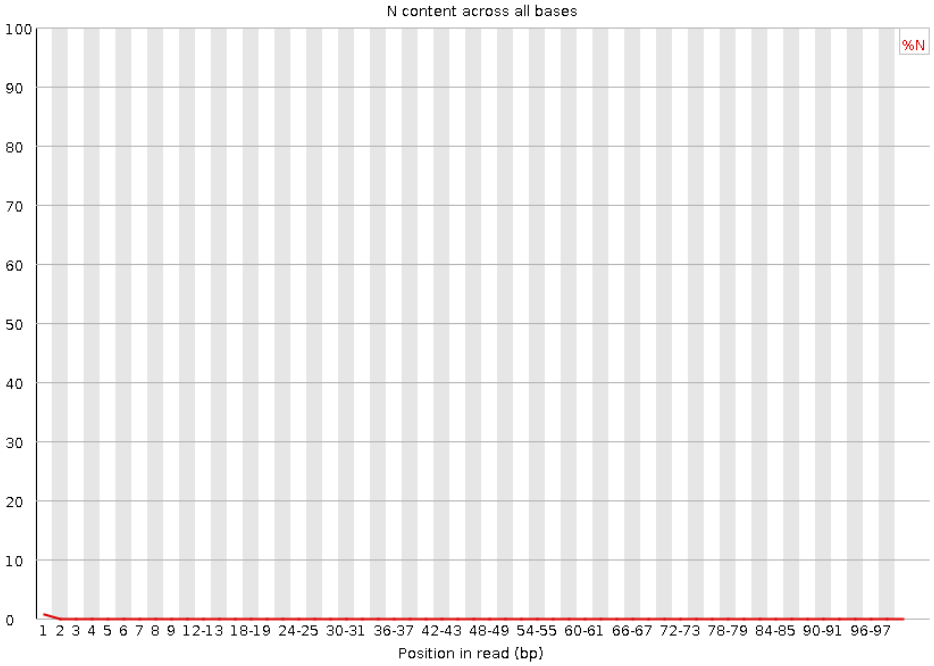
10R2_duplication_levels



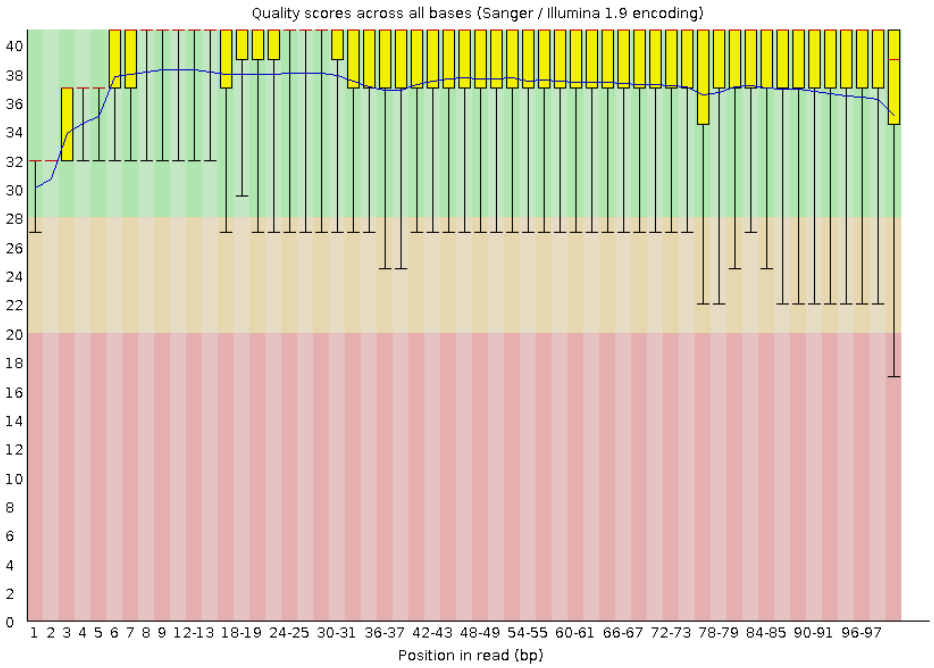
10R2_kmer_profiles



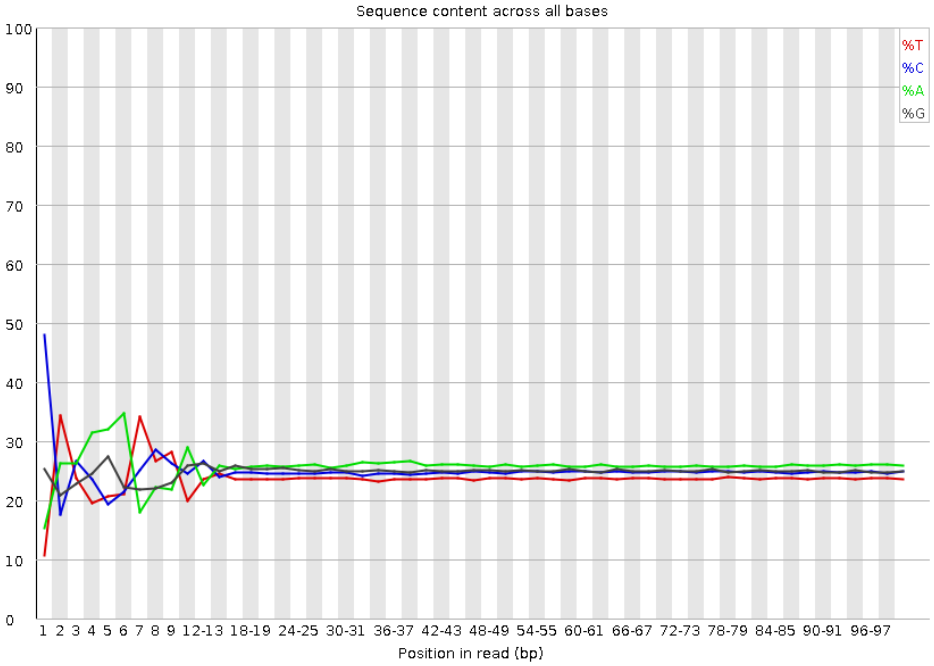
10R2__per__base__n__content



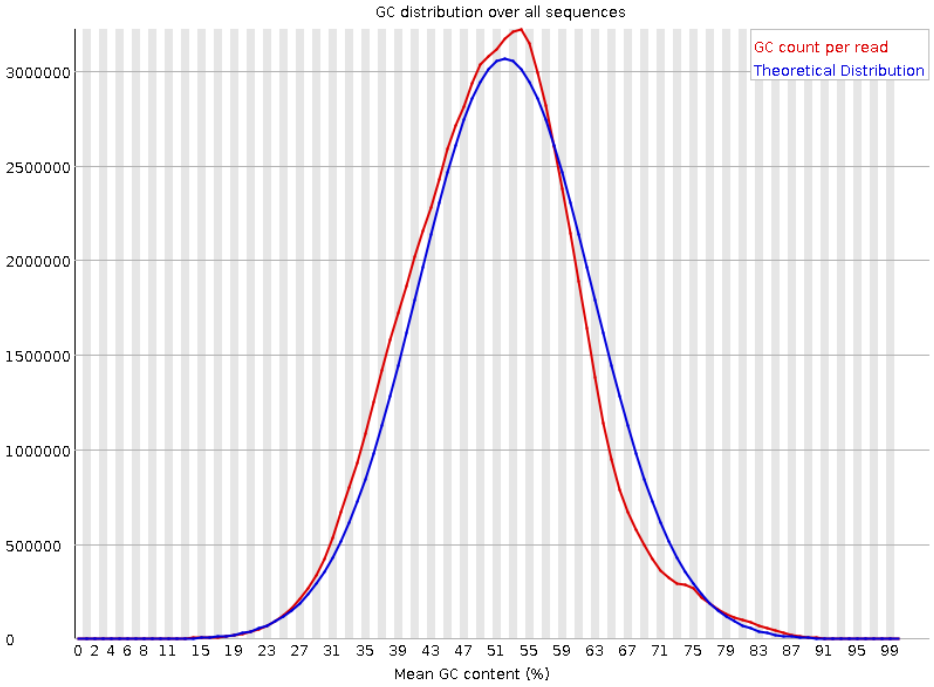
10R2__per__base__quality



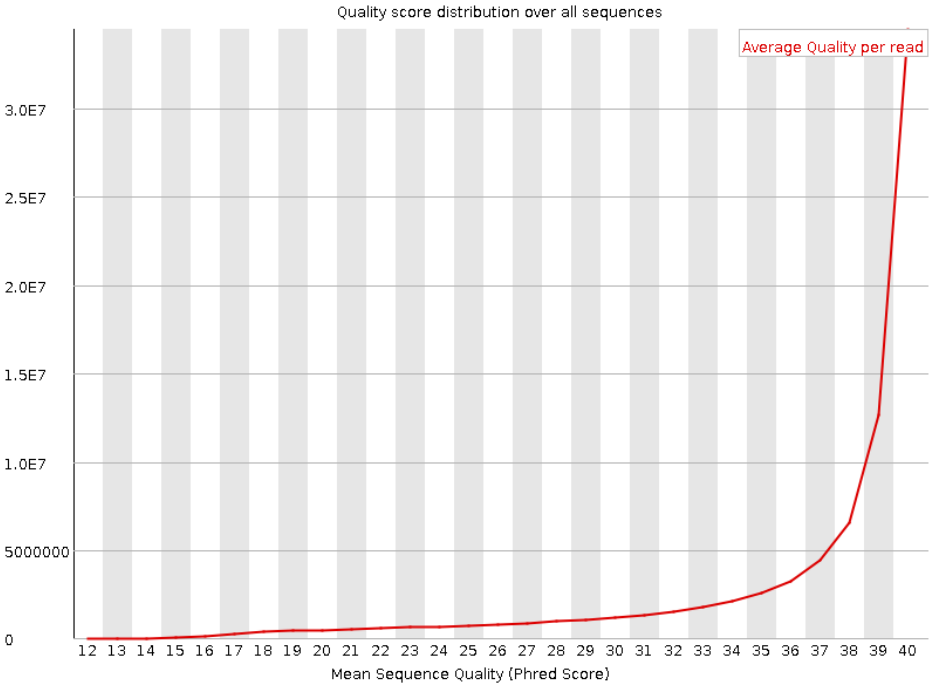
10R2__per__base__sequence__content



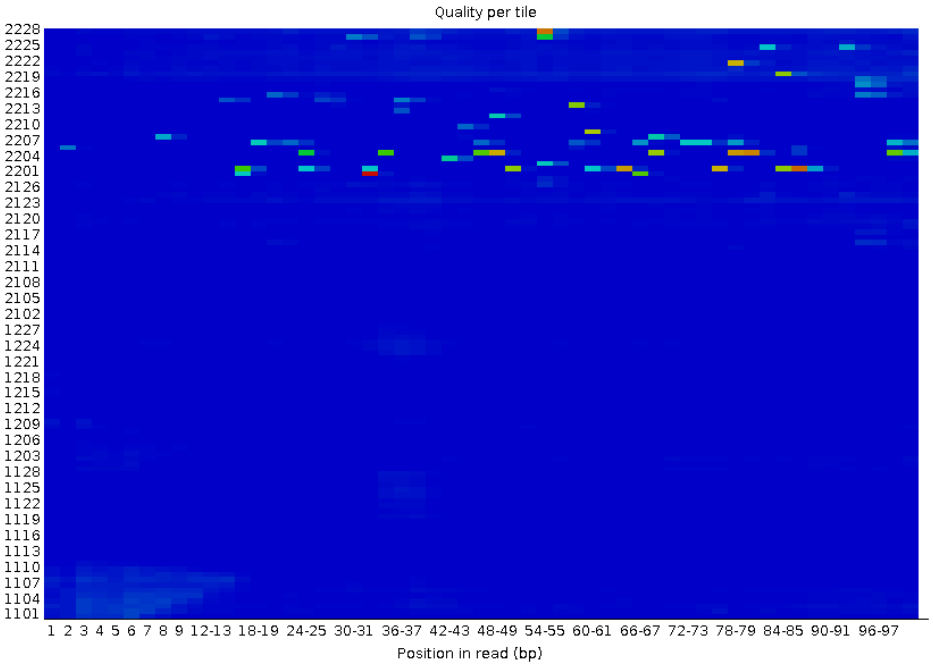
10R2__per__sequence__gc__content



10R2_per_sequence_quality



10R2_per_tile_quality



10R2_sequence_length_distribution

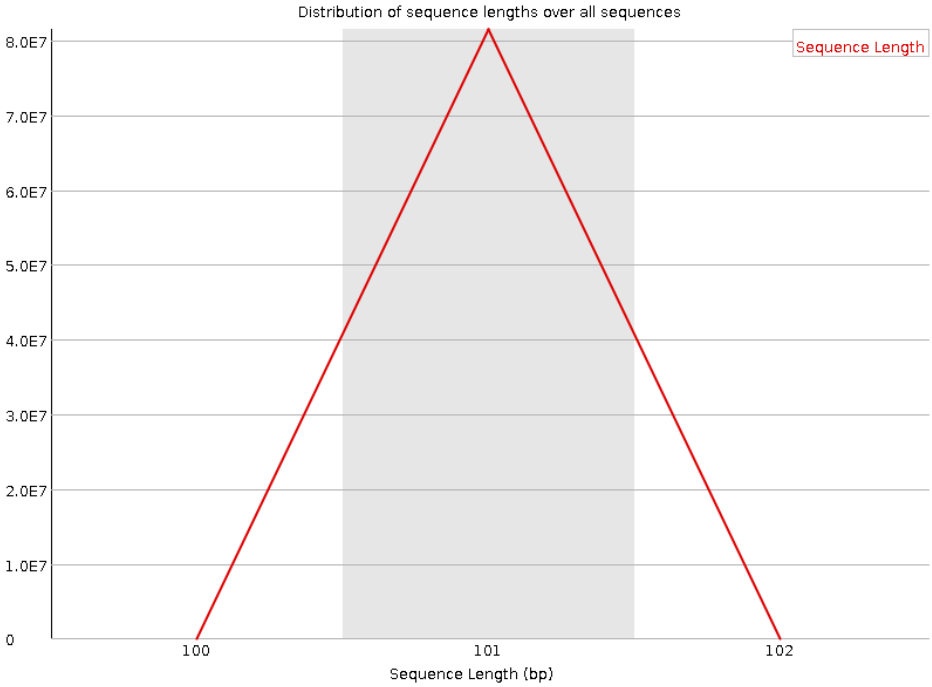
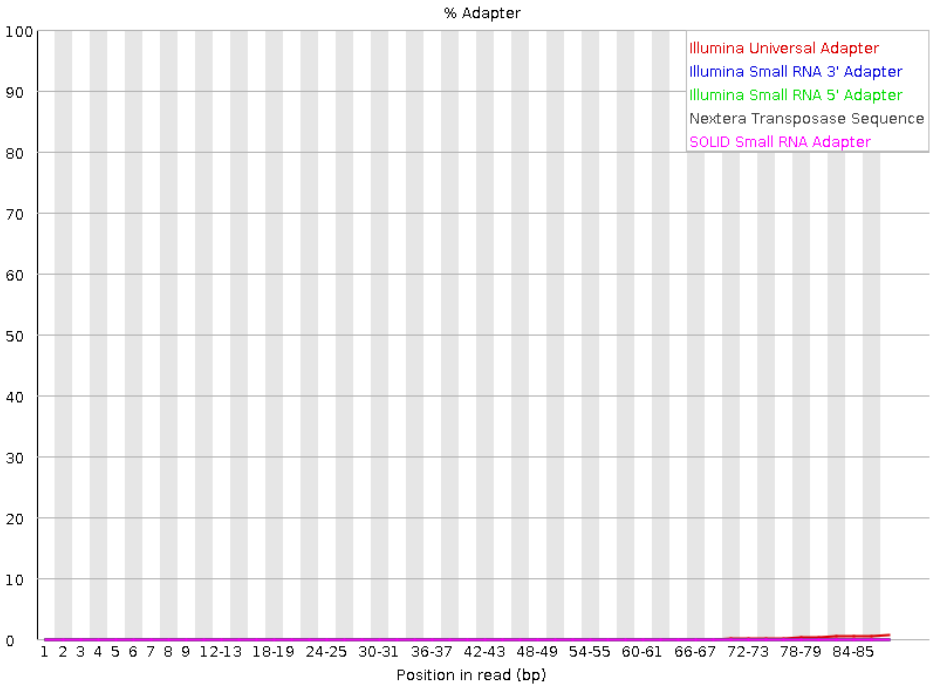


Figure outputs for 22_3H_both_S16_L008_R1_001.fastq.gz

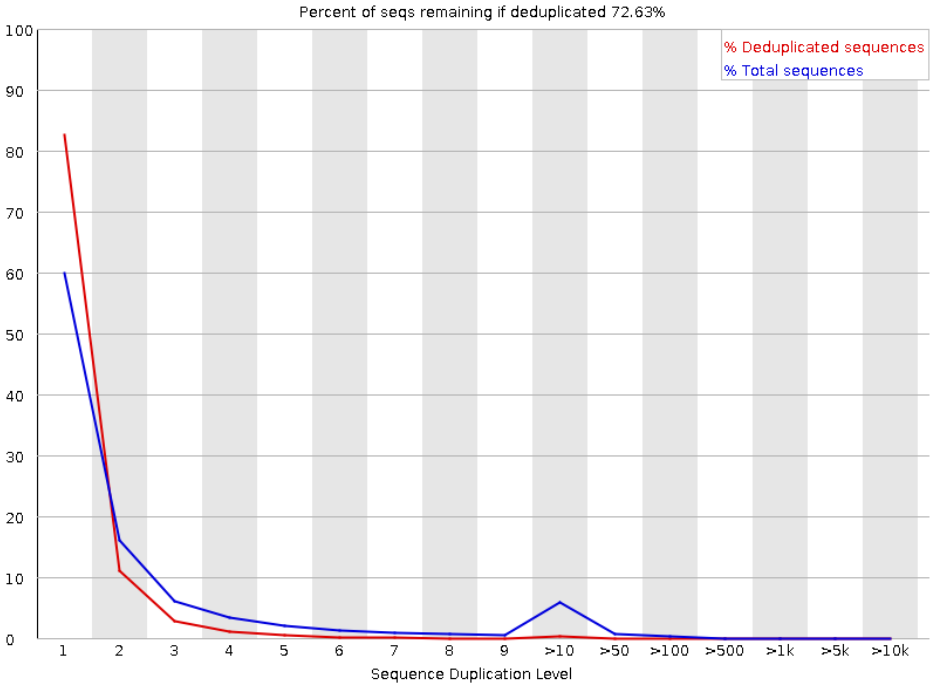
22_3H_both_S16_L008_R1_001.fastq.gz Summary

PASS Basic Statistics 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Per base sequence quality 22_3H_both_S16_L008_R1_001.fastq.gz
FAIL Per tile sequence quality 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Per sequence quality scores 22_3H_both_S16_L008_R1_001.fastq.gz
WARN Per base sequence content 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Per sequence GC content 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Per base N content 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Sequence Length Distribution 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Sequence Duplication Levels 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Overrepresented sequences 22_3H_both_S16_L008_R1_001.fastq.gz
PASS Adapter Content 22_3H_both_S16_L008_R1_001.fastq.gz
FAIL Kmer Content 22_3H_both_S16_L008_R1_001.fastq.gz

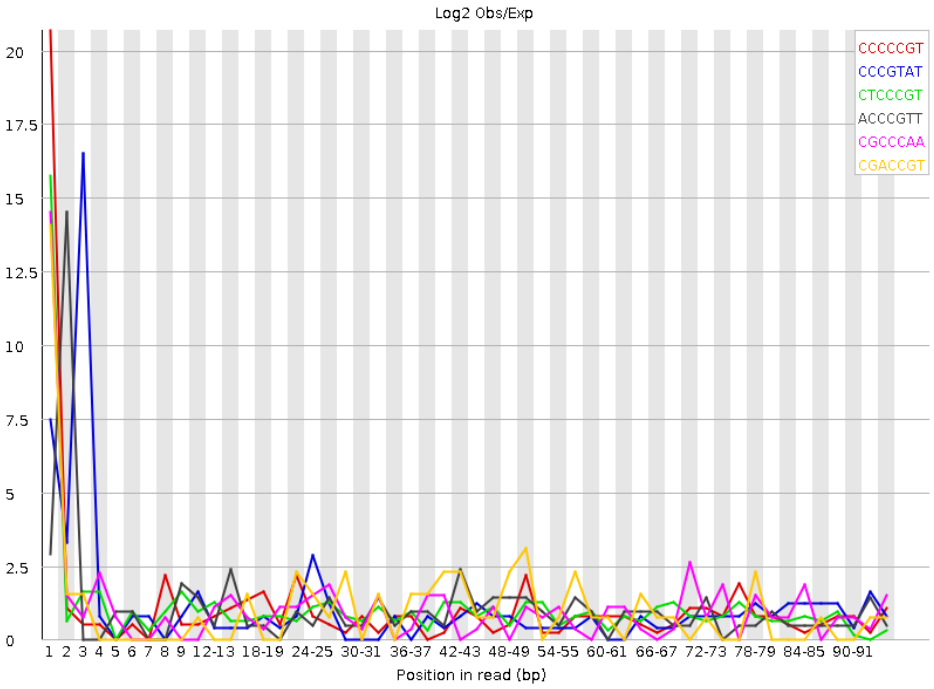
22R1_adapter_content



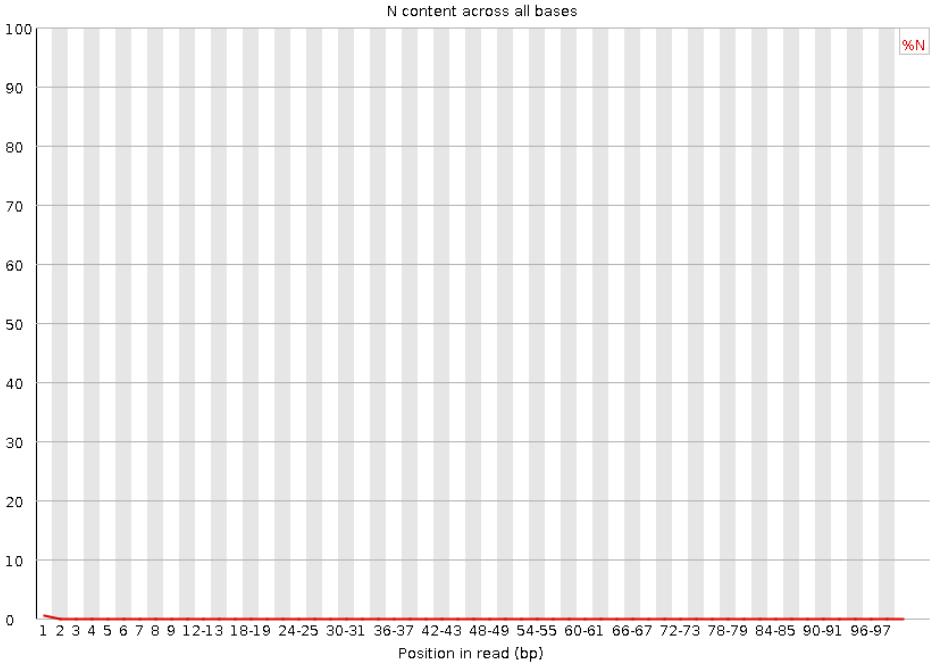
22R1_duplication_levels



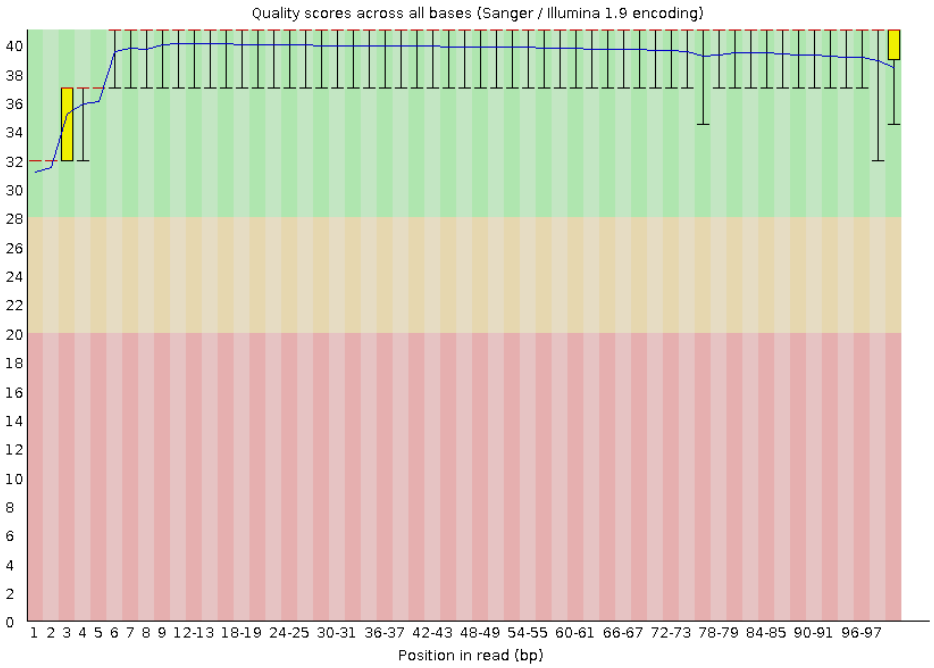
22R1_kmer_profiles



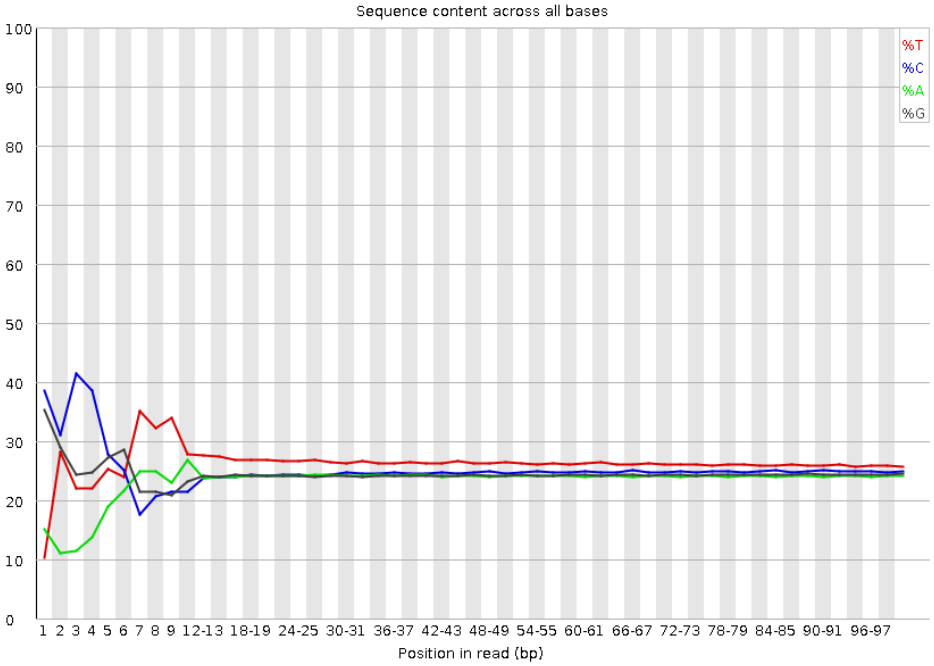
22R1_per_base_n_content



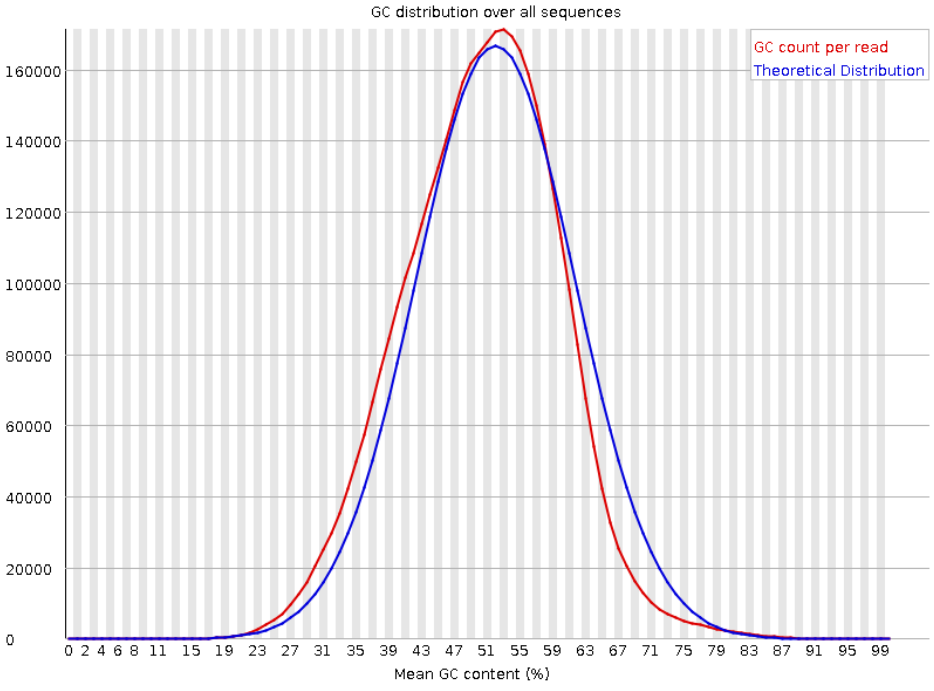
22R1_per_base_quality



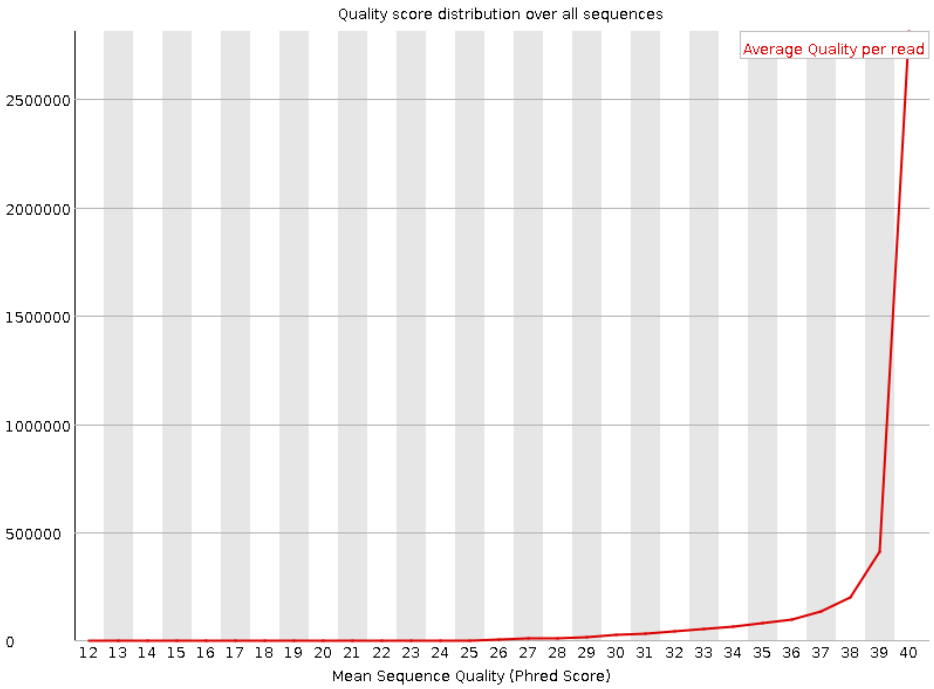
22R1__per__base__sequence__content



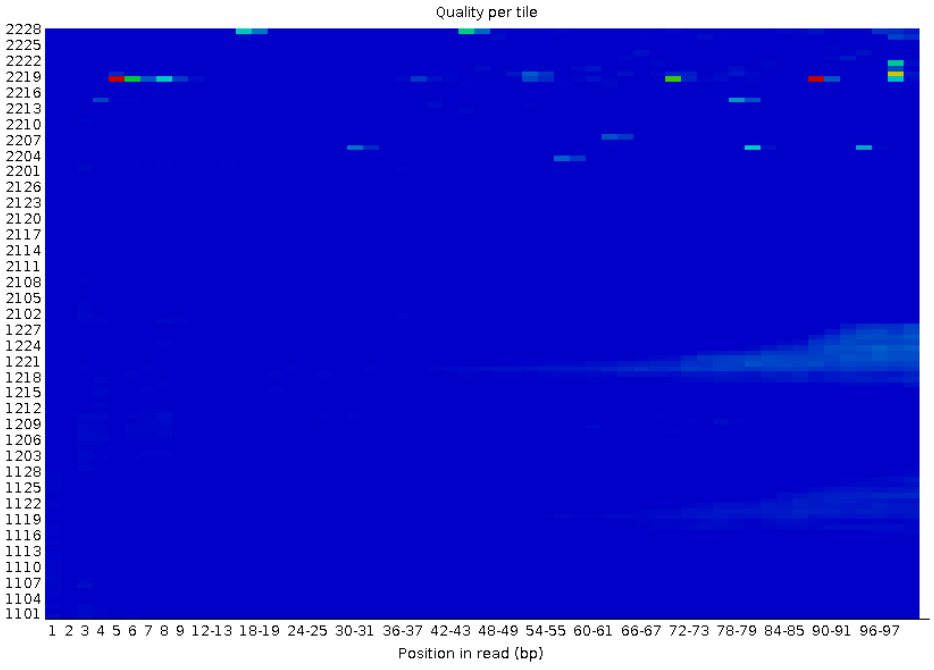
22R1__per__sequence__gc__content



22R1_per_sequence_quality



22R1_per_tile_quality



22R1_sequence_length_distribution

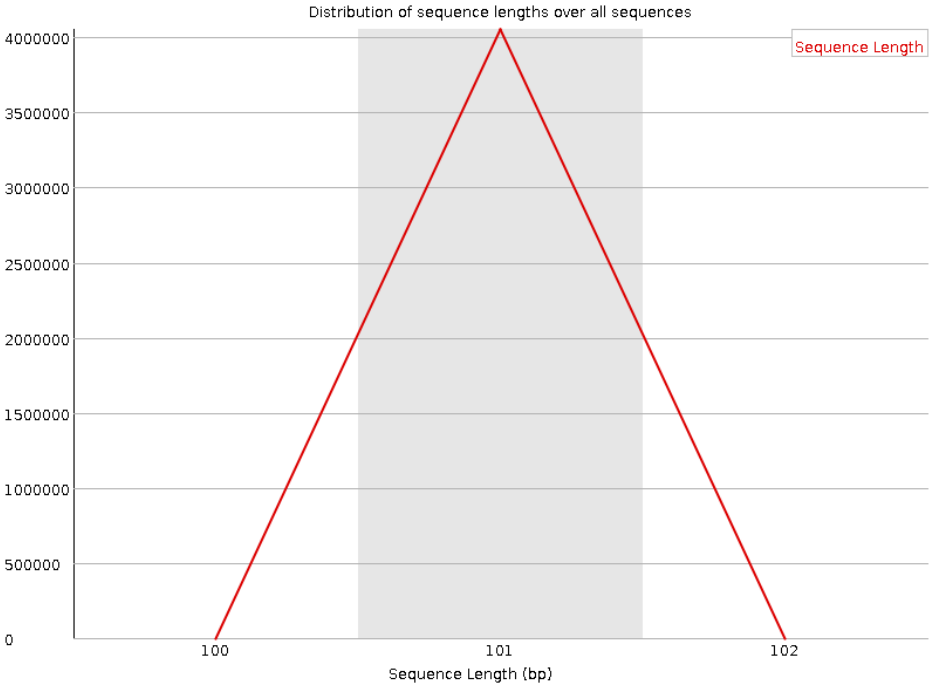
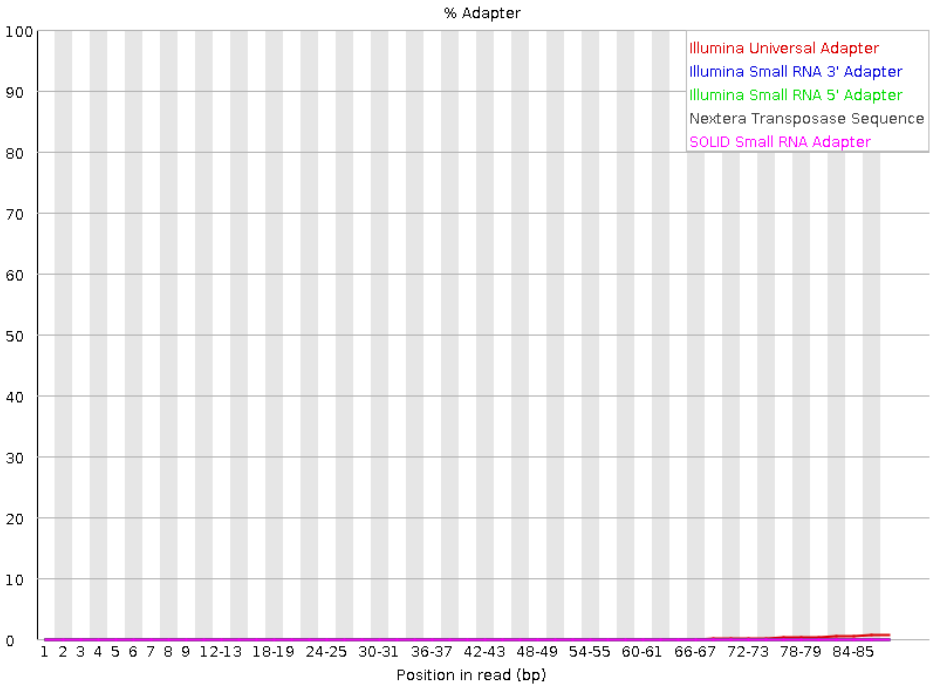


Figure outputs for 22_3H_both_S16_L008_R2_001.fastq.gz

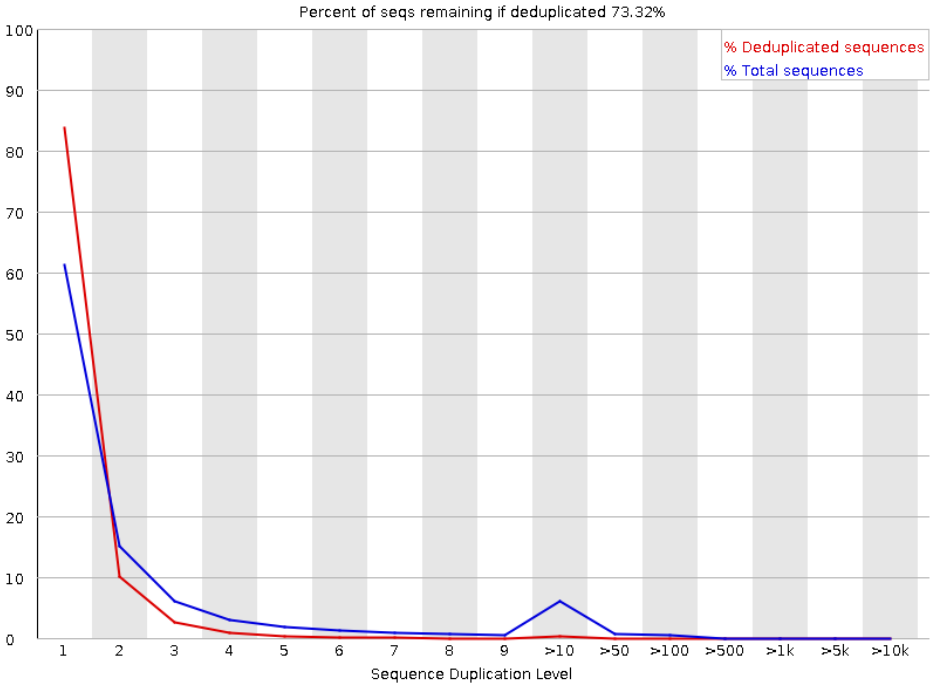
22_3H_both_S16_L008_R2_001.fastq.gz Summary

PASS Basic Statistics 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Per base sequence quality 22_3H_both_S16_L008_R2_001.fastq.gz
WARN Per tile sequence quality 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Per sequence quality scores 22_3H_both_S16_L008_R2_001.fastq.gz
WARN Per base sequence content 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Per sequence GC content 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Per base N content 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Sequence Length Distribution 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Sequence Duplication Levels 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Overrepresented sequences 22_3H_both_S16_L008_R2_001.fastq.gz
PASS Adapter Content 22_3H_both_S16_L008_R2_001.fastq.gz
FAIL Kmer Content 22_3H_both_S16_L008_R2_001.fastq.gz

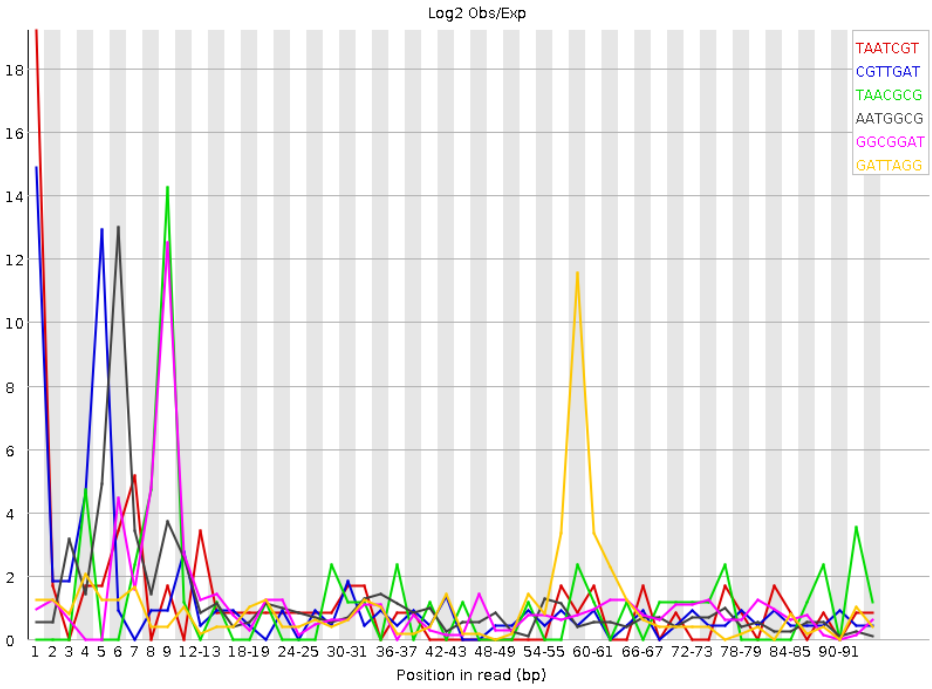
22R2_adapter_content



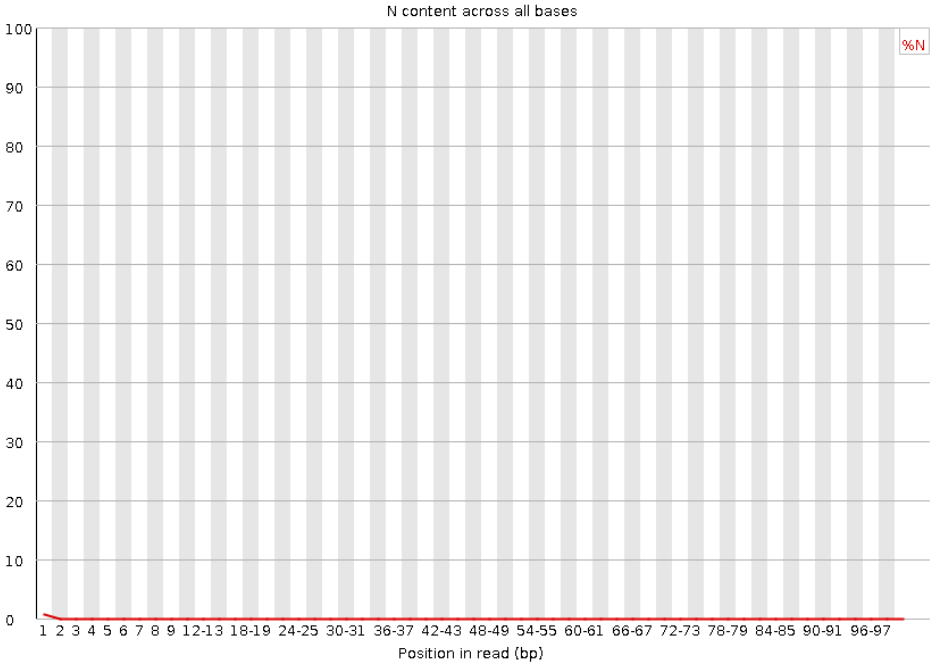
22R2_duplication_levels



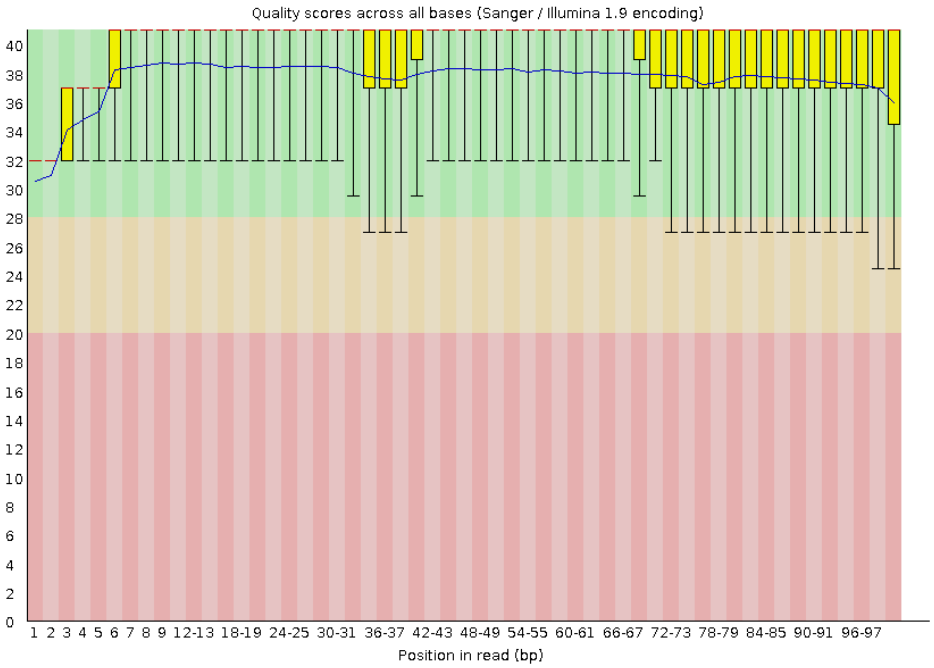
22R2_kmer_profiles



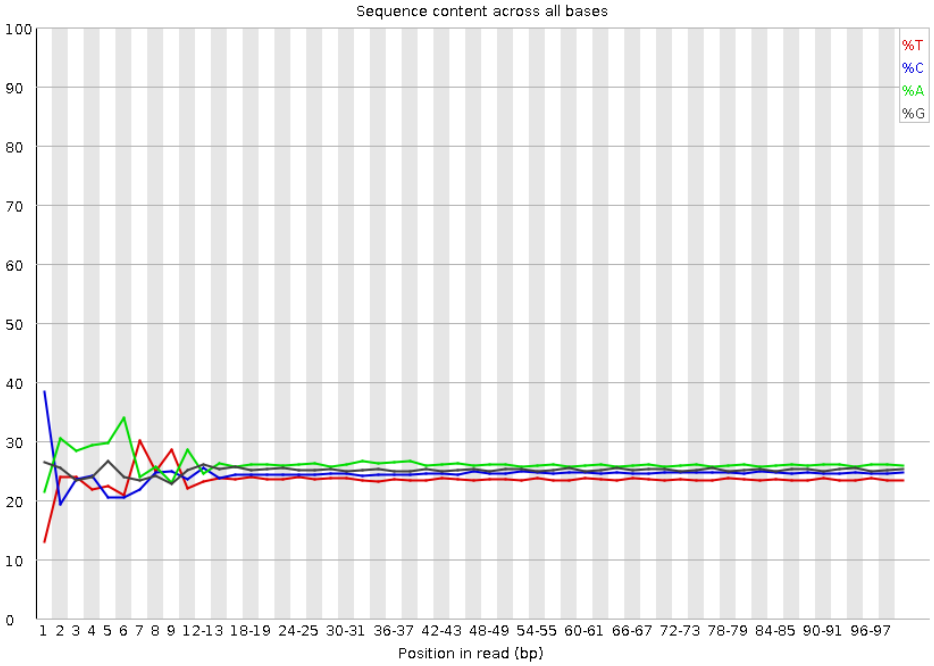
22R2__per__base__n__content



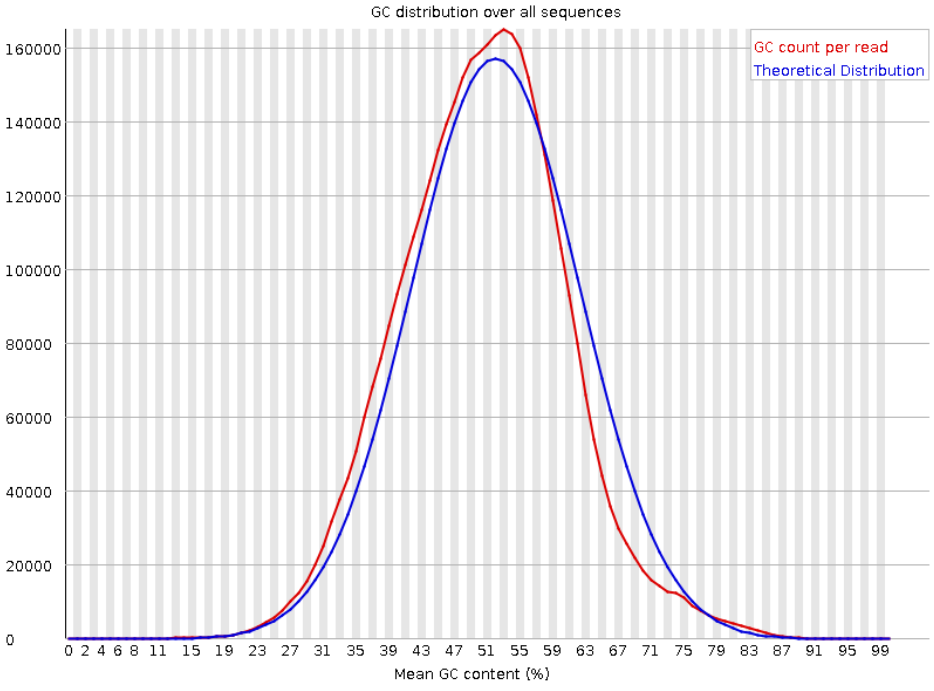
22R2__per__base__quality



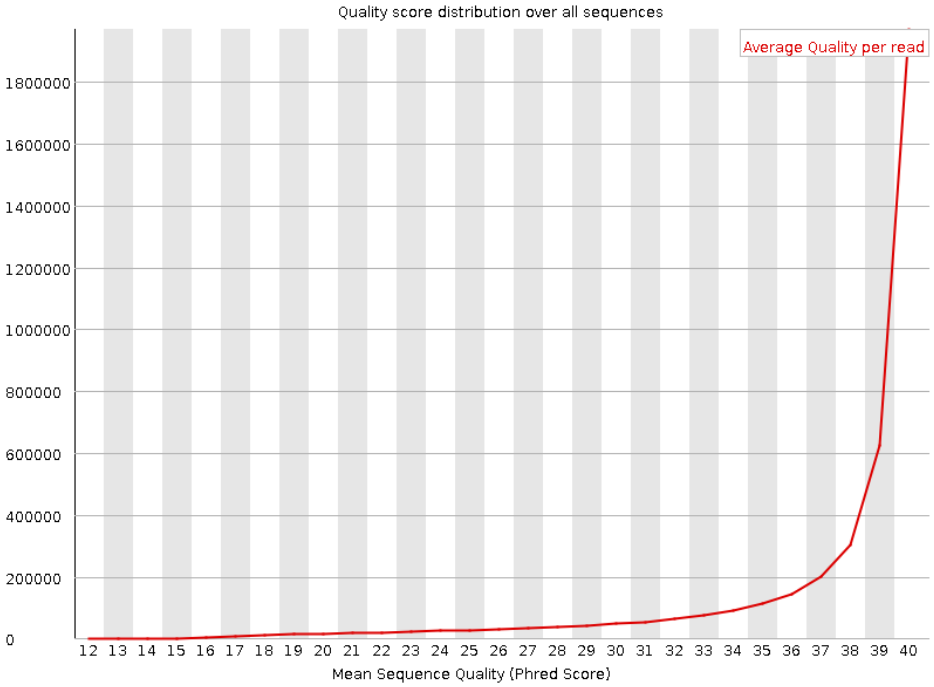
22R2__per__base__sequence__content



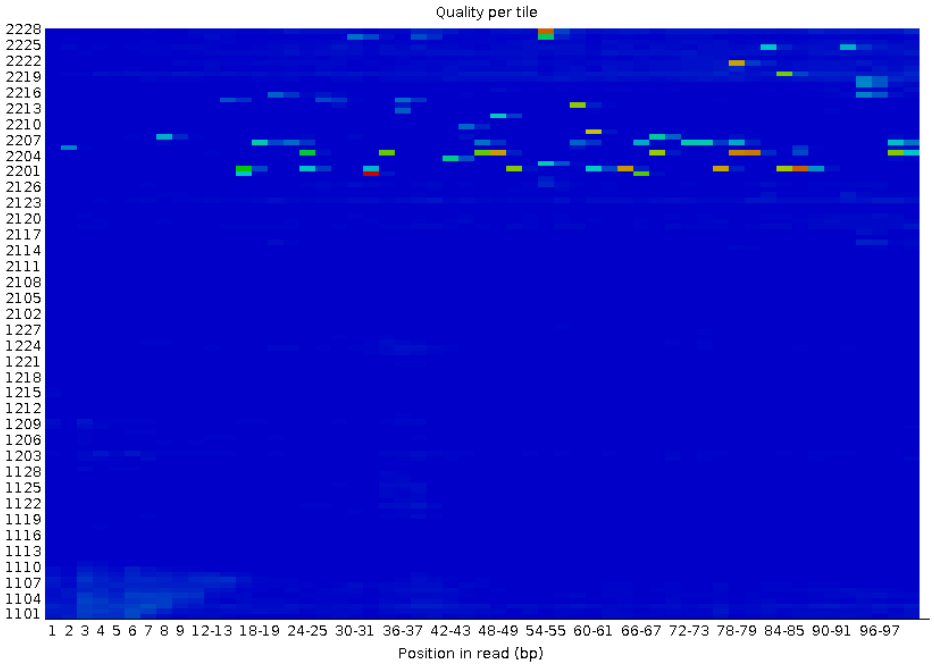
22R2__per__sequence__gc__content



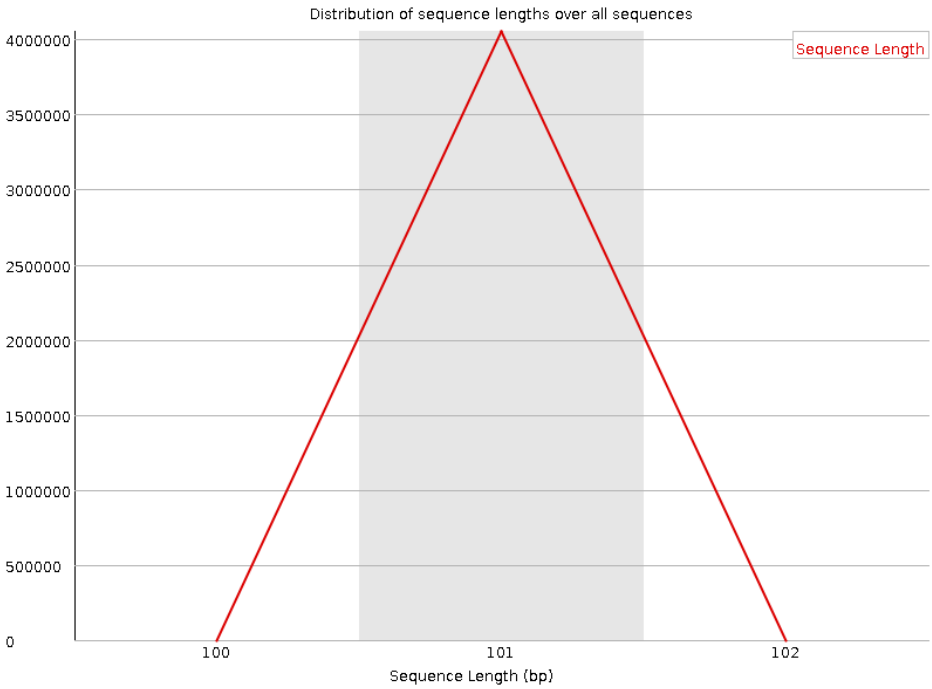
22R2_per_sequence_quality



22R2_per_tile_quality



22R2_sequence_length_distribution



SCRIPTS AND SCRIPT OUTPUTS

Demultiplexing The First sctipt and outputs

Demultiplexing The First sctipt

```
#!/usr/bin/env python3

import argparse
import numpy
import matplotlib.pyplot as plt
import gzip
print("pythonworking")


#argparse.
parser = argparse.ArgumentParser()
parser.add_argument("-seq1", default=1)
parser.add_argument("-seq2", default=1)
parser.add_argument("-index1", default=1)
parser.add_argument("-index2", default=1)
parser.add_argument("-dictionary", default=1)
args = parser.parse_args()

seq1 = args.seq1
seq2 = args.seq2
```

```
index1 = args.index1
index2 = args.index2
dictfile = args.dictionary

inputfiles = [seq1, seq2, index1, index2]


def savefig(name, x, y):
    plt.figure(name)
    plt.bar(x, height=y)
    plt.xlabel("Position")
    plt.ylabel("Average Qscore")
    plt.title(name + " qscore dist")
    plt.savefig(name)


def convert_phred(string: str):

    if len(string) == 1:
        return ord(string) - 33
    else:
        return [ord(x) - 33 for x in string]









error_default_names = ["read1", "read2", "index1", "index2"]
errorcounter = 0

for inputfile in inputfiles:
    if inputfile == 1:
        continue
    print("started one")
    Firstline2 = True
    sums = 1
    records = 0
    linelength = 0
    with gzip.open(inputfile, "rb") as input:
        for i, line in enumerate(input):



            if i%4 == 3:

                line = line.decode()
                line = line.strip()
                records += 1
                if Firstline2:
                    linelength = len(line)
```

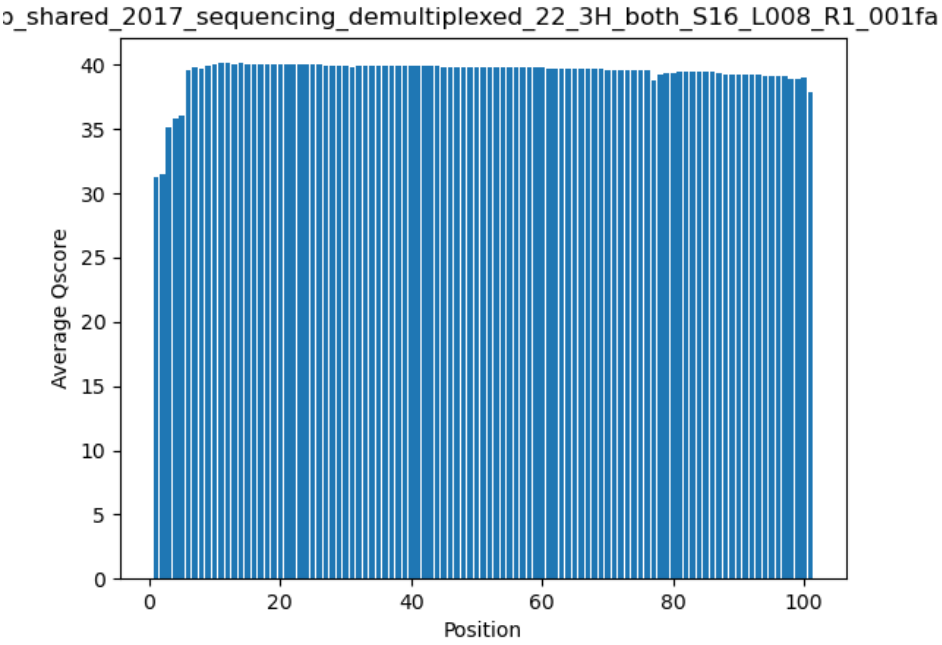
```
                    sums = numpy.zeros(linelength)
                    Firstline2 = False

                sums += numpy.array(convert_phred(line))
    averages = sums/records
    x = [v + 1 for v in range(linelength)]
    inputfile = inputfile.replace("/", "_")
    inputfile = inputfile.replace(".", "")
    try:
        savefig(inputfile, x, averages)
    except:
        savefig(error_default_names[errorcounter], x, averages)
    errorcounter += 1
```

Demultiplexing The First bash command used to run for 22R1, 22R2

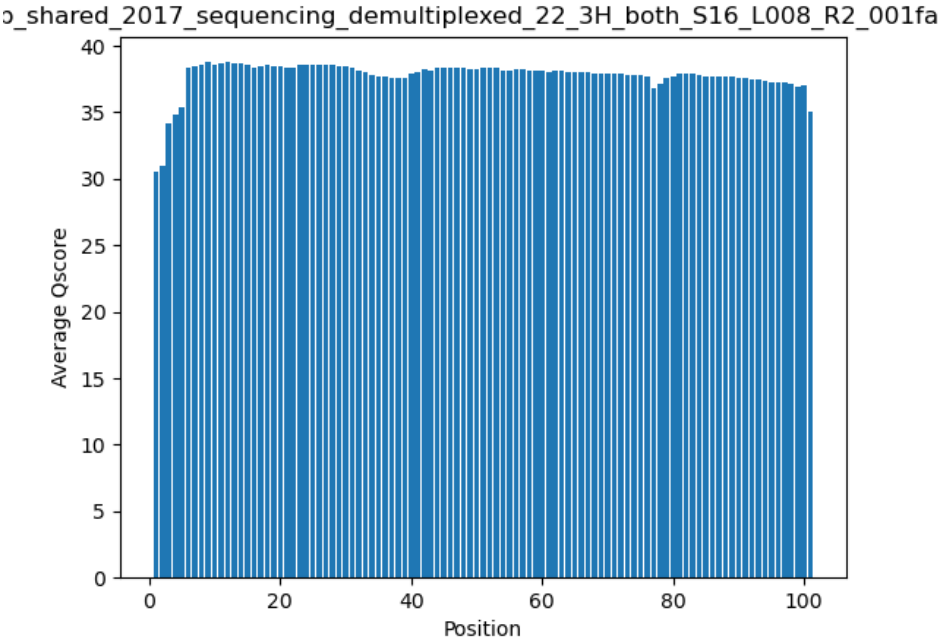
```
#!/bin/bash
#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=5



echo working...

/usr/bin/time -v python histomaker.py -seq1 /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_b
```

Demultiplexing The First runtime information + slurm output for 22R1, 22R2 working... pythonworking started one started one Command being timed: "python histomaker.py -seq1 /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_001.fastq.gz -seq2 /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz" User time (seconds): 171.97 System time (seconds): 0.22 Percent of CPU this job got: 99% Elapsed (wall clock) time (h:mm:ss or m:ss): 2:52.75 Average shared text size (kbytes): 0 Average unshared data size (kbytes): 0 Average stack size (kbytes): 0 Average total size (kbytes): 0 Maximum resident set size (kbytes): 53340 Average resident set size (kbytes): 0 Major (requiring I/O) page faults: 0 Minor (reclaiming a frame) page faults: 35429 Voluntary context switches: 1017 Involuntary context switches: 405 Swaps: 0 File system inputs: 0 File system outputs: 0 Socket messages sent: 0 Socket messages received: 0 Signals delivered: 0 Page size (bytes): 4096 Exit status: 0

TF__22R1



TF__22R2



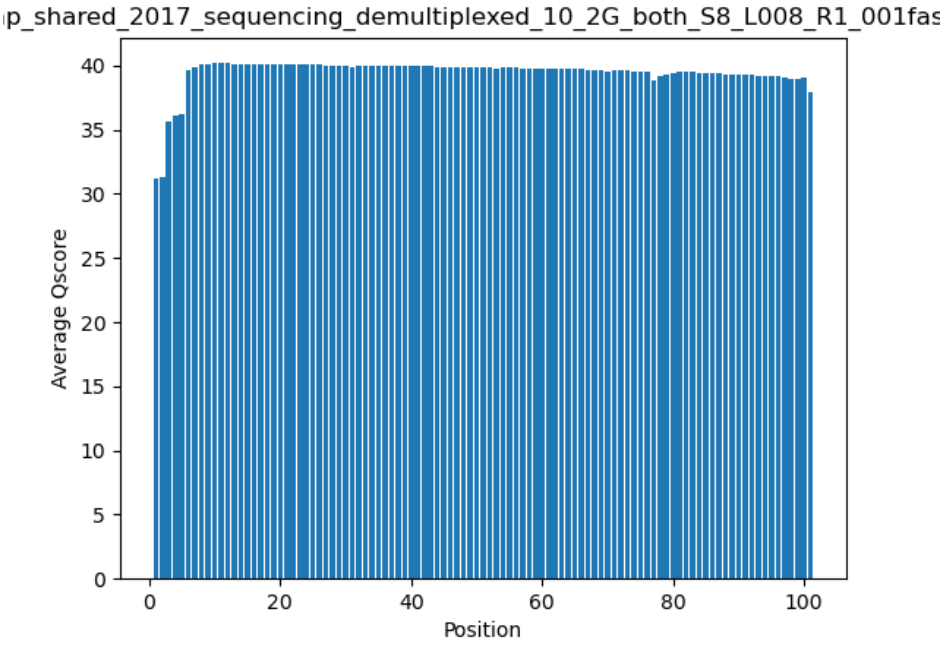
Demultiplexing The First bash command used to run for 10R1, 10R2

```
#!/bin/bash
#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=120



echo working...

/usr/bin/time -v python histomaker.py -seq1 /projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_b
```

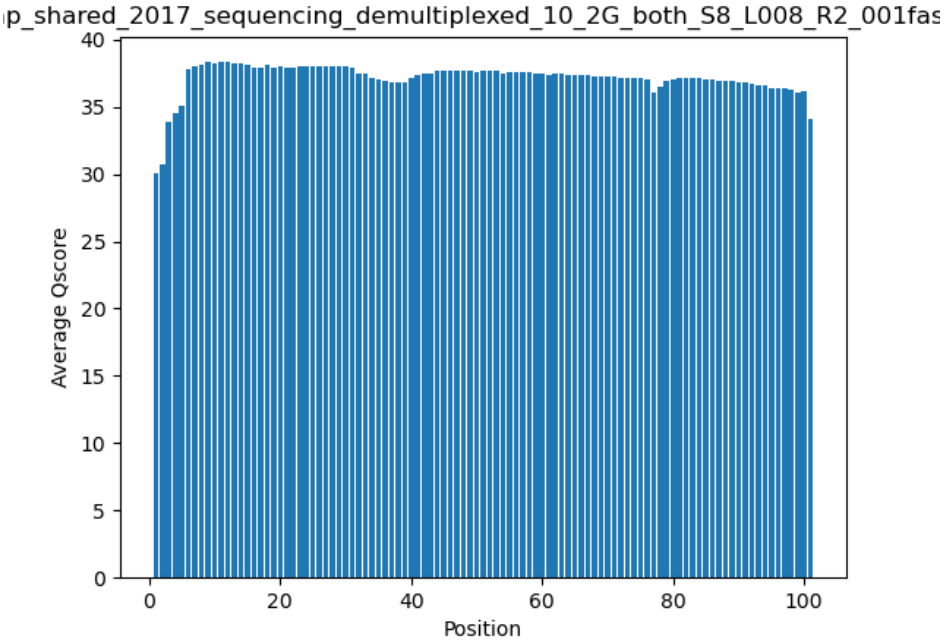
Demultiplexing The First runtime information + slurm output for 10R1, 10R2

```
working...
pythonworking
started one
started one
    Command being timed: "python histomaker.py -seq1 /projects/bgmp/shared/2017_sequencing/demultiplexe
    User time (seconds): 2800.80
    System time (seconds): 2.02
    Percent of CPU this job got: 99%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 46:50.33
    Average shared text size (kbytes): 0
    Average unshared data size (kbytes): 0
    Average stack size (kbytes): 0
    Average total size (kbytes): 0
    Maximum resident set size (kbytes): 53340
    Average resident set size (kbytes): 0
    Major (requiring I/O) page faults: 0
    Minor (reclaiming a frame) page faults: 70734
    Voluntary context switches: 1062
    Involuntary context switches: 6747
    Swaps: 0
    File system inputs: 0
    File system outputs: 0
    Socket messages sent: 0
    Socket messages received: 0
    Signals delivered: 0
    Page size (bytes): 4096
    Exit status: 0
```

TF_10R1



TF_10R2



STAR_BASH_COMMAND

```
#!/bin/bash
#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=20:00:00


path2readfiles=/projects/bgmp/shared/Bi621
path2PS8=/home/dbrowne2/bgmp/bioinfo/Bi623/Assignments/Assignment_QAA/cutzone


/usr/bin/time -v STAR --runThreadN 8 --runMode genomeGenerate \
    --genomeDir $path2PS8/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz_ENS_108_30-Mar-2021_11:332.7.9
    --genomeFastaFiles $path2PS8/dre/Mus_musculus.GRCm39.dna.primary_assembly.fa \
    --sjdbGTFfile $path2PS8/Mus_musculus.GRCm39.104.gtf

echo marker1

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
    --outFilterMultimapNmax 3 \
    --outSAMunmapped Within KeepPairs \
    --alignIntronMax 1000000 --alignMatesGapMax 1000000 \
    --readFilesCommand zcat \
    --readFilesIn $path2PS8/22R1_cut_trimmed_paired.fastq.gz $path2PS8/22R2_cut_trimmed_paired.fastq.gz
    --genomeDir $path2PS8/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz_ENS_108_30-Mar-2021_11:332.7.9
    --outFileNamePrefix 22R1R2_vs_Mouse
echo marker2

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
    --outFilterMultimapNmax 3 \
    --outSAMunmapped Within KeepPairs \
    --alignIntronMax 1000000 --alignMatesGapMax 1000000 \
    --readFilesCommand zcat \
    --readFilesIn $path2PS8/10R1_cut_trimmed_paired.fastq.gz $path2PS8/10R2_cut_trimmed_paired.fastq.gz
    --genomeDir $path2PS8/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz_ENS_108_30-Mar-2021_11:332.7.9
    --outFileNamePrefix 10R1R2_vs_Mouse
```

10R1R2 STAR metrics

```
                             Started job on |   Sep 06 19:09:39
                         Started mapping on |   Sep 06 19:09:50
                                Finished on |   Sep 06 19:20:33
       Mapping speed, Million of reads per hour |   434.02

                           Number of input reads |   77520904
```

```
                     Average input read length |   199
                                  UNIQUE READS:
                Uniquely mapped reads number |   72939237
                     Uniquely mapped reads % |   94.09%
                       Average mapped length |   199.15
                    Number of splices: Total |   55735572
         Number of splices: Annotated (sjdb) |   55385233
                    Number of splices: GT/AG |   55171465
                    Number of splices: GC/AG |   443410
                    Number of splices: AT/AC |   61228
            Number of splices: Non-canonical |   59469
                   Mismatch rate per base, % |   0.23%
                      Deletion rate per base |   0.00%
                     Deletion average length |   1.96
                     Insertion rate per base |   0.00%
                    Insertion average length |   1.42
                         MULTI-MAPPING READS:
      Number of reads mapped to multiple loci |   3493273
           % of reads mapped to multiple loci |   4.51%
      Number of reads mapped to too many loci |   479486
           % of reads mapped to too many loci |   0.62%
                               UNMAPPED READS:
Number of reads unmapped: too many mismatches |   0
     % of reads unmapped: too many mismatches |   0.00%
          Number of reads unmapped: too short |   575010
               % of reads unmapped: too short |   0.74%
              Number of reads unmapped: other |   33898
                   % of reads unmapped: other |   0.04%
                               CHIMERIC READS:
                     Number of chimeric reads |   0
                          % of chimeric reads |   0.00%
```

22R1R2 STAR metrics

```
                             Started job on |   Sep 06 19:08:53
                         Started mapping on |   Sep 06 19:09:03
                                Finished on |   Sep 06 19:09:39
     Mapping speed, Million of reads per hour |   390.11

                        Number of input reads |   3901127
                     Average input read length |   199
                                  UNIQUE READS:
                Uniquely mapped reads number |   3673877
                     Uniquely mapped reads % |   94.17%
                       Average mapped length |   199.00
                    Number of splices: Total |   2891600
         Number of splices: Annotated (sjdb) |   2873637
                    Number of splices: GT/AG |   2861963
                    Number of splices: GC/AG |   23339
```

```
                        Number of splices: AT/AC |   3190
                Number of splices: Non-canonical |   3108
                       Mismatch rate per base, % |   0.22%
                          Deletion rate per base |   0.00%
                         Deletion average length |   1.94
                         Insertion rate per base |   0.00%
                        Insertion average length |   1.40
                              MULTI-MAPPING READS:
       Number of reads mapped to multiple loci |   167705
            % of reads mapped to multiple loci |   4.30%
       Number of reads mapped to too many loci |   25301
            % of reads mapped to too many loci |   0.65%
                                  UNMAPPED READS:
  Number of reads unmapped: too many mismatches |   0
       % of reads unmapped: too many mismatches |   0.00%
            Number of reads unmapped: too short |   32699
                 % of reads unmapped: too short |   0.84%
                Number of reads unmapped: other |   1545
                     % of reads unmapped: other |   0.04%
                                  CHIMERIC READS:
                       Number of chimeric reads |   0
                            % of chimeric reads |   0.00%
```

Aligned_vs_unaligned

```
#!/usr/bin/env python3

import argparse




parser = argparse.ArgumentParser()
parser.add_argument("-f")

args = parser.parse_args()

inputfile = args.f



unmapped = 0
total_aligned = 0

with open(inputfile, "r") as fh:
    for line in fh:
```

```
        if line[0] != '@':


            flag = int(line.split(sep = "\t")[1])

            if ((flag & 256) != 256):
                if ((flag & 4) != 4):
                    total_aligned += 1
                else:
                    unmapped += 1


print(total_aligned)
print(unmapped)
```

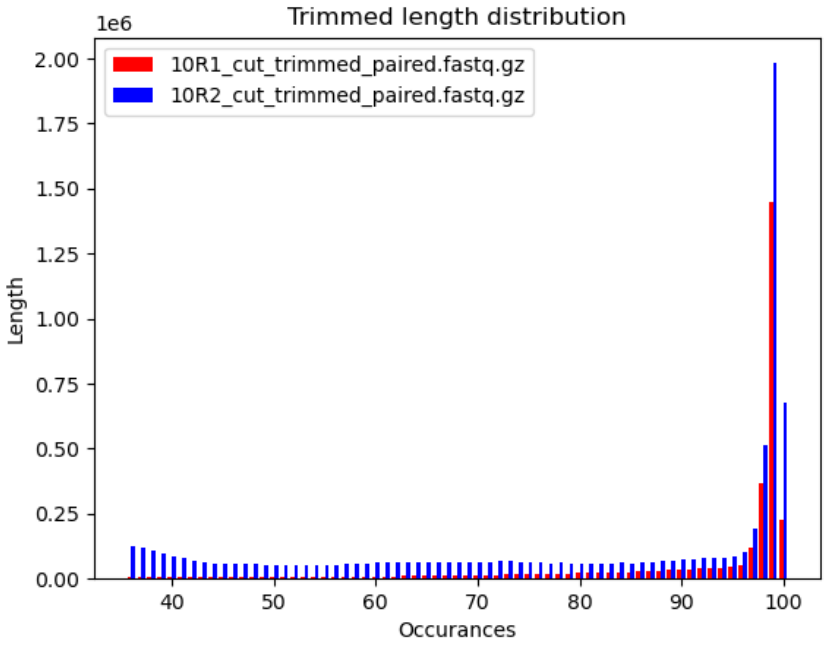
Trimmed file length histograms and script

Trimmed file length script

```
#!/usr/bin/env python3

import argparse

import matplotlib.pyplot as plt
import gzip
import numpy


#argparse.
parser = argparse.ArgumentParser()
parser.add_argument("-seq1")
parser.add_argument("-seq2")
parser.add_argument("-read_length")

args = parser.parse_args()

seq1 = args.seq1
seq2 = args.seq2
read_length = args.read_length


inputfiles = [seq1, seq2]


def savefig(name, x, y):
    plt.figure(name)
    plt.bar(x, height=y)
```

```
    plt.xlabel("Position")
    plt.ylabel("Average Qscore")
    plt.title(name + " qscore dist")
    plt.savefig(name)


read_length = int(read_length)



unzippedfiles = [gzip.open(x, 'rt') for x in inputfiles]

seqlengths = {}



for i, line in enumerate(zip(unzippedfiles[0], unzippedfiles[1])):
    if i%4 == 1:

        seq1len = len(line[0])
        seq2len = len(line[1])

        if seq1len  < read_length:
            if seq1len in seqlengths:
                seqlengths[seq1len][0] += 1
            else:
                seqlengths[seq1len] = [0,0]

        if seq2len  < read_length:
            if seq2len in seqlengths:
                seqlengths[seq2len][1] += 1
            else:
                seqlengths[seq2len] = [0,0]

colors = ['red', 'blue']
sortedkeys = sorted(seqlengths.keys())

seq1occs = [seqlengths[x][0] for x in sortedkeys]
seq2occs = [seqlengths[x][1] for x in sortedkeys]

sortedkeys = numpy.array(sortedkeys)

plt.bar(sortedkeys - 0.2, seq1occs, 0.4, color = colors[0])
plt.bar(sortedkeys + 0.2, seq2occs, 0.4, color = colors[1])

plt.xlabel("Occurances")
plt.ylabel("Length")
plt.title("Trimmed length distribution")
plt.legend(inputfiles)

plt.savefig("10R1R2_HISTO")
```

10R1R2 histogram



22R1R2 histogram