

PHS 675

Special Topics: Analyzing RNA Sequencing Data

Homework 3

Model Solution

I. The line

```
s="M-TH-Chow-20wk-2307-Pancreas"
```

sets a variable named `s` to the name of the second sample. After this, `${s}` will resolve to `M-TH-Chow-20wk-2307-Pancreas`.

The command `trimmomatic PE` starts trimmomatic in paired-end mode.

The first two parameters are the locations of the forward read input file and reverse read input file. These will be set to `raw_reads/M-TH-Chow-20wk-2307-`

`Pancreas_R1.fastq.gz` and `raw_reads/M-TH-Chow-20wk-2307-Pancreas_R2.fastq.gz`, respectively.

The second pair of parameters are the locations of the output files for the forward reads; the first is for forward reads that have a matching reverse read, and the second for forward reads that do not have a matching reverse read. These are set to files with the same name as the forward read input file, but in a folder called `trimmed` for reads with matching pairs and `unpaired_trimmed` for reads without matching pairs.

The third pair of parameters are the locations of the output files for the reverse reads; the first is for reverse reads that have a matching forward read, and the second for reverse reads that do not have a matching forward read. These are set to files with the same name as the reverse read input file, but in a folder called `trimmed` for reads with matching pairs and `unpaired_trimmed` for reads without matching pairs.

The remaining parameters are the three trimming steps. Each step has the name of the step, followed by a colon and then parameters separated by colons.

- a. `ILLUMINACLIP:TruSeq3-PE-2-GGGGGG.fa:2:30:10` This clips adapter (or other technical) sequences from the reads. The first parameter is the name of the fasta file which defines the adapters sequences. The second is the number of mismatches allowed in the seed for alignment (so two mismatches are allowed

when trying to perform an initial match of the read sequence to the adapter sequence). The third is the minimum quality score for an alignment to recognize a read sequence as an adapter sequence when using "palindrome mode" (i.e. when considered the read pair and looking for reverse complements between the two pairs). The final parameter is the minimum quality score when performing a simple alignment between the read and adapter sequence.

- b. **SLIDINGWINDOW:4:14** This implements trimming for low quality bases. A "sliding window" of size 4 (the first parameter) is placed at the beginning of the read. If the average quality score over the four bases is below 14 (the second parameter), the read is truncated. The window is then shifted right one base, and the process repeated until the read is either truncated or the end of the read is reached (in which case the read is left without modification).
- c. **MINLEN:25** This step enforces a minimum length for the reads after the previous two steps have potentially trimmed them to be shorter than their original length. Reads less than 25 bases are discarded.

II. The code **samples=\$(ls raw_fastqs | cut -d '_' -f 1 | sort -u)** ds

The code **ls raw_fastqs** lists the contents of the **raw_fastqs** folder (which contains 34 fastq files whose form is **sample_R?.fastq.gz**, where sample is the sample name and the ? is either 1 or 2). The vertical bar after this command (**|**) sends the output to the next command, instead of to the console.

The code **cut -d '_' -f 1** cuts each line of input at the delimiter specified by **-d** (which in this case is the underscore character, **_**). The **-f** parameter determines which parts to output after this cutting occurs, so in this case only the first part is output. Since the input to this command is "piped" from the previous command, this will operate on each filename from the **raw_fastqs** folder. Since each of those files consists of the sample name, followed by the underscore, the output will just be the sample name. These sample names are output to the next command. Note that since each sample name has a read 1 and read 2 file, each sample name is output twice.

The **sort -u** command sorts the input (alphabetically). The **-u** parameter then only outputs unique values. Since the input to this command is the sample names from each file, each appearing twice, this will sort them alphabetically and output each sample name just once.

In summary, the total effect of **ls raw_fastqs | cut -d '_' -f 1 | sort -u** is to output each

sample name, ensuring each name appears only once.

The `$(...)` construct evaluates to the output of the enclosed command. So `samples=$(ls raw_fastqs | cut -d '_' -f 1 | sort -u)` assigns the output of `ls raw_fastqs | cut -d '_' -f 1 | sort -u` to the variable `samples`. (Omitting the `$(...)` construct would just assign the command itself to the variable.)

Overall, the net effect of `samples=$(ls raw_fastqs | cut -d '_' -f 1 | sort -u)` is to assign the list of sample names to the variable `samples`.

III. The construct

```
for s in ${samples}; do
```

```
...
```

```
done
```

iterates through all elements of `samples` and performs the enclosed code once for each element. On each iteration, the variable `s` contains the "current element". So this code will call

```
srun -p normal trimmomatic PE ...
```

once for each sample, with `s` holding a different sample name each time.

The code `srun -p normal ...` will submit the call to trimmomatic for the current sample to the high performance compute cluster. The `-p normal` option specifies the partition on which the job is run on the cluster.

Finally, the ampersand (`&`) at the end of the call to `srun -p normal trimmomatic ...` runs the submission in the background, so the next job can be submitted before the previous one completes.

IV. Using `cat logs/M-TH-Chow-20wk-2306-Pancreas_trim.log` we can see the contents of the trimming log file for the first sample. This contains the line

```
Input Read Pairs: 4090281 Both Surviving: 3895713 (95.24%) Forward Only Surviving: 180598 (4.42%) Reverse Only Surviving: 10555 (0.26%) Dropped: 3415 (0.08%)
```

which indicates that 95.24% of the reads for that sample survived the trimming.

We can see this line from all the log files using

```
grep "Input Read Pairs:" logs/*_trim.log
```

from which we can see that the survival rate for both reads ranged from 91.93% to 97.70%.

To fully automate this, we could do

```
grep -h "Input Read Pairs:" logs/*_trim.log > trim_summary.txt
```

and then in R do

```
trim_summary <-  
  read_tsv('trim_summary.txt', col_names='summary') %>%  
  mutate(  
    temp=gsub("Input Read Pairs: ([0-9]+) Both Surviving: ([0-9]+).*",  
              "\\1\\t\\2",  
              summary)  
  ) %>%  
  select(!summary) %>%  
  separate(temp, into=c("num_input", "num_surviving")) %>%  
  mutate(across(starts_with("num_"), as.numeric)) %>%  
  mutate(percent_surviving = 100 * num_surviving / num_input)  
  
trim_summary %>% pull(percent_surviving) %>% summary()
```