The flowchart describing how the pipeline processes the data:

STEP 0: Getting/Copying files necessary for STEP 1.

0. Getting/Copying files necessary for 1. , from /localdisk/data/BPSM/ICA1/fastq to ICA1/fastq

Execute STEP0_copy_fastq.sh

```
. STEP0_copy_fastq.sh
```

1) Download the data
2) Move STEP 1 shell script into ./fastq
3) Change directory to ./fastq

STEP 1: Quality check on gzip fastq sequence data using fastqc

1. Quality check on gzip fastq sequence data using fastqc

Execute STEP1run_fastqc.sh by typing the command:

```
. STEP1_run_fastqc.sh
```

INPUT:
The fastq sequence gzip files:
sequence.gz

PROCESS:

1) Make a directory fastqc_out
2) Run fastqc on each .gz file and save the output in fastqc_out
3) Move STEP2 shell script into fastqc_out/
4) Change directory to fastqc_out

OUTPUT:
Summary text containing number of PASS/FAIL/WARN
Files:
Tco-123_1_fastqc.html
Tco-123_1_fastqc.zip --> summary.txt
(NO NEED TO UNZIP the output files, use script in STEP 2 to extract files instead.)

```
PASS    Basic Statistics        Tco-106_1.fq.gz
PASS    Per base sequence quality       Tco-106_1.fq.gz
PASS    Per sequence quality scores     Tco-106_1.fq.gz
FAIL    Per base sequence content       Tco-106_1.fq.gz
PASS    Per sequence GC content Tco-106_1.fq.gz
PASS    Per base N content      Tco-106_1.fq.gz
PASS    Sequence Length Distribution    Tco-106_1.fq.gz
PASS    Sequence Duplication Levels     Tco-106_1.fq.gz
WARN    Overrepresented sequences       Tco-106_1.fq.gz
PASS    Adapter Content Tco-106_1.fq.gz
```

STEP 2: Assess the numbers and quality of the raw sequence data based on the previous output.

2. Assess the numbers and quality of the raw sequence data based on the previous output.

Execute STEP2_extract_summary.sh:
sh STEP2_extract_summary.sh

INPUT:
summary.txt for each sequence

→

PROCESS Step 1: Write a shell script to extract "summary.txt" from each zip file

Instead of unzipping the zip file from step 1, we can simply do:
1) use a for loop, extract all summary.txt files from the directories, save them in a new directory "extracted_summaries/" and
2) rename them so that we don't overwrite anything

PROCESS Step 2:

1) Change directory to extracted_summaries/
2) Extract the count of PASS/FAIL/WARN in each summary file, write the output in a 3 separate text file with the sequence name
   Name of the 3 files:
   - pass_report.txt
   - fail_report.txt
   - warn_report.txt

Note: The script I wrote will read the first few lines of fail_report.txt and warn_report.txt for the reader to decide whether they'd filter out any sequence data.

→

OUTPUT:

1. A summary of the quality assessment of raw sequence data: /extracted_summaries/Tco-123_1_fastqc_summary.txt
2. Count of PASS/FAIL/WARN in each summary.txt file in 3 separate txt files:
   - pass_report.txt
   - fail_report.txt
   - warn_report.txt

(Would have done this given worse data)
PROCESS Step 3:

1) Implement a programme/criteria to rank the data as GOOD/ACCEPTABLE/BAD according to the average of the number of PASS/FAIL/WARN across all sequences
   e.g. GOOD: FAIL <avg_fail && WARN < avg_warn
   ACCEPTABLE: FAIL = avg_fail +- 1 && WARN = avg_warn +-1
   BAD: FAIL > avg_fail +- 1 OR WARN > avg_warn +- 1
2) Filter out all the data that are BAD.

STEP 3.1: Align the read pairs to the Trypanosoma congolense genome using bowtie2

3.1. Align the read pairs to the Trypanosoma congolense genome using bowtie2
Execute STEP3_1_align_read_pairs.sh:
. STEP3_1_align_read_pairs.sh

INPUT:
Trypanosoma congolense IL3000 genome sequence in fasta format (All files in directory */localdisk/data/BPSM/ICA1/Tcongo_genome/*)

→

PROCESS Step 1:
Unzip all .gz files so we can use it later for bowtie2 (in PROCESS step 3)

PROCESS Step 2:

1) Copy the Genome fasta file from */localdisk/data/BPSM/ICA1/Tcongo_genome/* to the current directory
2) Move the file inside to the current working directory

PROCESS Step 3: Run bowtie2 on the sequences, using TriTrypDB-46_TcongolenseIL3000_2019_Genome.fasta as an index

1) Build fasta_index, which we will use as a genome index for bowtie2
2) Using a for loop, align reads to the indexed sequence using bowtie2
   a. Extracting the sequence name
   b. Construct the corresponding _2.fq filename within the for loop
   c. Align reads to an indexed sequence using bowtie2

→

OUTPUT:
output_${fqname}.sam

STEP 3.2: Convert the output from 3.2. to indexed "bam" format with samtools

3.2. Convert the output from 3.2. to indexed "bam" format with samtools

Execute STEP3_2_use_samtools.sh:

```
. STEP3_2_use_samtools.sh
```

INPUT: output_${fqname}.sam →

PROCESS Step 1: Convert output.sam into a BAM alignment using samtools view

1) Save all sam files into a variable ${output_file} so we can feed into for loop
2) Convert output.sam into a BAM alignment using samtools view (BAM: Binary Alignment Map, which stores the same data as a compressed binary file (faster to work with))
3) For loop, taking each file from ${output_file} and convert output.sam into a BAM alignment using samtools view

STORAGE: output.bam

PROCESS Step 2: Sort the alignment using samtool sort

1) Save all bam files into a variable ${bamfile} so we can feed into for loop
2) For loop, feed each file from bamfile and sort using samtool sort →

OUTPUT: output.bam sorted_output.bam sorted_output.bam.bai

STORAGE: sorted_output.bam

PROCESS Step 3: This sorted BAM alignment file can now be indexed using samtools index.

1) Save all sorted bam files into ${sortedbam}
2) For loop feeding in each file from sortedbam and run samtools index

STORAGE: sorted_output.bam.bai

STEP 4: Generate counts data: the number of reads that align to the regions of the genome that code for genes (the bed file)

4. Generate counts data: the number of reads that align to the regions of the genome that code for genes

Execute STEP4_generate_count.sh:

```
. STEP4_generate_count.sh
```

INPUT:

- TriTrypDB-46_TcongolenseIL3000_2019.bed
- Tco2.fqfiles

PROCESS Step 1: Group things first according to the order in Tco2.fqfiles

1) Filter out the header row
2) Sort Tco2.fqfiles by SampleType (col2), Time (col 4) and Treatment (col 5) ignore "Replicate (col 3)"
3) Use a while loop:
   a. find rows that share the same values in SampleType (col2), Time (col 4) and Treatment (col 5), and group them.
   b. If they're different, then go to the next group and keep reading until it's different again
   c. Save the group name as SampleType_Time_Treatment
   d. Save the SampleName under the corresponding groups
4) *Do 1) to 3) again, and save to group_Tco2.txt , to resolve a bug in PROCESS step 2.*

STORAGE:
groups_Tco2.txt (used for PROCESS Step 2 only)
final_groups_Tco2.txt (for user to view)

PROCESS Step 2: Use bedtool multicov to get sequence counts in the genes listed by bedtool, iterate through all groups

1) Define the bedfile as "TriTrypDB-46_TcongolenseIL3000_2019.bed"
2) Create a (BIG) while loop which feeds in the groups_Tco2.txt file generated from step 1.
   a. If the line read starts with 'Group:', then we:
   b. Read the line starting with 'Group:' as GroupName, but without the prefix 'Group:', and create an empty array to store sample names in the group.
   c. Use a while loop to
      i. Read the sample names (so != "Group:"*) and stored in an array, again, without the prefix.
      ii. Store all the sample names under array so we can use it for input in bedtool multicov -bams
   d. Use a for loop, add the BAM file names of the samples into an empty array.
   e. Run bedtool multicov

→ OUTPUT: final_groups_Tco2.txt <GroupName>_genecounts.cov

This should have worked and processed all the lines in group_Tco2.txt, but I'm not sure why it magically skips a group each time the if statement is run. (I've checked that the statement is absolutely correct.)

My temporary solution:

1. Run step 1 first, so there will be 2 tables of the same content appended to each other, back to back. (Note this is intentional, so that the programme in step 2 will run all the "Group" rows even though it skipped them in the first round.

STORAGE: <GroupName>_genecounts.cov

The gene names are in the 4th column of the bedfile, followed by gene description in the 5th, the count data is contained in the 6th column onwards.

STEP 5: Generate average of the counts per gene

5. **Generate plain text tab-delimited output files that give the statistical mean (average) of the counts per gene (i.e. expression levels) for each group**
   Execute STEP5_generate_average.sh:
   `. STEP5_generate_average.sh`

INPUT: <GroupName>_genecounts.cov →

PROCESS Step 1: Calculate the average gene_count FOR each group

1) Store the names of all the genecounts.cov files in a variable
2) Use a for loop and call each item, use them as input for an awk script, which does the following:
   a. Sets sum & count as zero variables
   b. Loop through fields from the 6th field (the gene count) until the end and do the following:
      i. Add the value of the field/column read into sum
      ii. Add 1 to count each time, which will be the number sequences in that group
   c. If count is non-zero, calculate the average by dividing the sum by count; otherwise, average is 0
   d. Print the gene name ($4), gene description ($5) and average to a file called "${group_name}_average.txt"

→ OUTPUT: <GroupName>_average.txt

STEP 6: Compare groups and get fold change

6. Use the mean expression levels to generate "fold change" data
   Execute STEP6_generate_average.sh:
   `. STEP5_generate_average.sh`

INPUT: <GroupName>_average.txt from every group →

PROCESS: Step 1

1) Creating an array of file names corresponding to the average files generated from STEP 5

2) Use a for loop,
   - **a. Call each item in the average_file array each time, as "file1",**
   - **b. And record the iteration key as i.**
   - **c. i increases by 1 for each iteration.**
   - **d. Use a second for loop within this for loop,**
     - **i. Call each item in the average_file array each time, as "file2", but one step ahead of the first for loop.**
     - **ii. Record the iteration key as j, which is 1 larger than i. This is to prevent situations where duplicate/reverse comparisons are made i.e. group 1 v. group 2 is the same as group 2 v. group 1; We don't want group 2 v. group 1!!**
     - **iii. j also increases by 1 for each iteration.**
     - **iv. Only do the following if iteration key j is larger than i:**
       1) Use awk to get us the average value for both groups, save them as variables
       2) Use awk to get the gene names and descriptions, save as a variable gene_info
       3) Only if the average for group 2 is non-zero (i.e., we are not dividing by 0)
          a) Fold_change is avg1/avg2, which 5 decimal places.
       4) Otherwise, fold_change is automatically 0.
     - **v. Paste gene_info and fold_change into a file called** "${group_name1}_VS_${group_name2}_fold.txt"

3) Save the names of all files with the suffix _fold.txt into an array fold_file
4) Start a for loop, taking in every element from the array fold_file, for every element:
   a. Filter out rows that have empty column 3 (this is an unresolved mysterious issue remaining from 2).)
   b. Overwrite the files with the new filtered file free of unwanted rows of data (there were 10k of them!)
   c. Sort the 3rd column of the filtered fold_file, using natural sort so that it sorts in descending order of the magnitude of the values in that column.

→ OUTPUT:
<GroupName1>_VS_<GroupName2>_fold.txt_sorted.txt
(descending order in terms of column 3, the fold change)
<GroupName1>_VS_<GroupName2>_fold.txt

Programme parameters justifications:

All relevant justifications and explanations are indicated in the codes. I have written them in detail within the scripts so please do read them there.

User instructions:

There will be embedded codes that move the scripts around. Please have ALL the scripts ready in ICA1/ to start with.

****There is no need to download/copy/move anything, these are all done for you in the script****

STEP 0:

${PWD} = ICA1/

Execute the shell script STEP0_copy_fastq.sh:

```
. STEP0_copy_fastq.sh
```

STEP 1:

${PWD} = ICA1/fastq

Execute the shell script STEP1_run_fastqc.sh:

```
. STEP1_run_fastqc.sh
```

STEP 2:

${PWD} = ICA1/fastq/fastqc_out

Execute the shell script STEP2_extract_summary.sh:

```
. STEP2_extract_summary.sh
```

STEP 3.1:

${PWD} = ICA1/fastq

Execute the shell script STEP3_1_align_read_pairs.sh:

```
. STEP3_1_align_read_pairs.sh
```

STEP 3.2:

${PWD} = ICA1/fastq

Execute the shell script STEP3_2_use_samtools.sh:

```
. STEP3_2_use_samtools.sh
```

STEP 4:

${PWD} = ICA1/fastq

Execute the shell script STEP4_generate_count.sh:

```
. STEP4_generate_count.sh
```

STEP 5

${PWD} = ICA1/fastq

Execute the shell script STEP5_generate_average.sh:

```
. STEP5_generate_average.sh
```

STEP 6

${PWD} = ICA1/fastq

Execute the shell script STEP6_get_fold_change.sh:

```
. STEP6_get_fold_change.sh
```

Difficulties encountered:

1. The while loop issue in PROCESS step 2 of STEP 5: The code I wrote basically skips a row when reading the groups_Tco2.txt file and I've had to go back to PROCESS step 1 to make PROCESS step 1 run twice for it to compensate for the skipping issue. This should still work if future data is added, because running PROCESS step 1 twice makes it so that the programme in PROCESS step 2 has to run the rows it skipped in the first round. [RESOLVED]
2. During STEP 6, when appending/echoing/printfing/pasting everything together into 3 different columns, this did not work. It gave me 1 giant column with 30k rows instead! So I finally settled with the paste command (see STEP 6 shell script) and instead filtered out rows with empty column 3, and still ended up with the desired output file. [RESOLVED]

Alternative beneficial features:

1. STEP 2: The quality of the data we're given is generally ok, with everything having only 1 or 0 FAIL/WARN count. So nothing was filtered out and no programme was written for that purpose either. But in the future, you might get data that are of worse quality and will require such a filter. This is what I would have done after obtaining the PASS/FAIL/WARN counts:
    a. Implement a programme/criteria to rank the data as good/acceptable/bad according to the average of the number of PASS/FAIL/WARN
2. An interactive tool or programme to group the sequences so that it is more intuitive to interpret the output data by comparing experimental conditions. Give the user a choice about how which groups to choose to compare, instead of comparing all and outputting all, then letting the user to find the file themselves.
3. Having the sequence names within the final results table, so that it is easier to also see which sequences are used as input data, without having to go back to refer to final_groups_Tco2.txt.
4. Automatically generate a report with the names and description of the genes that have the highest fold change within a certain group-wise comparison.