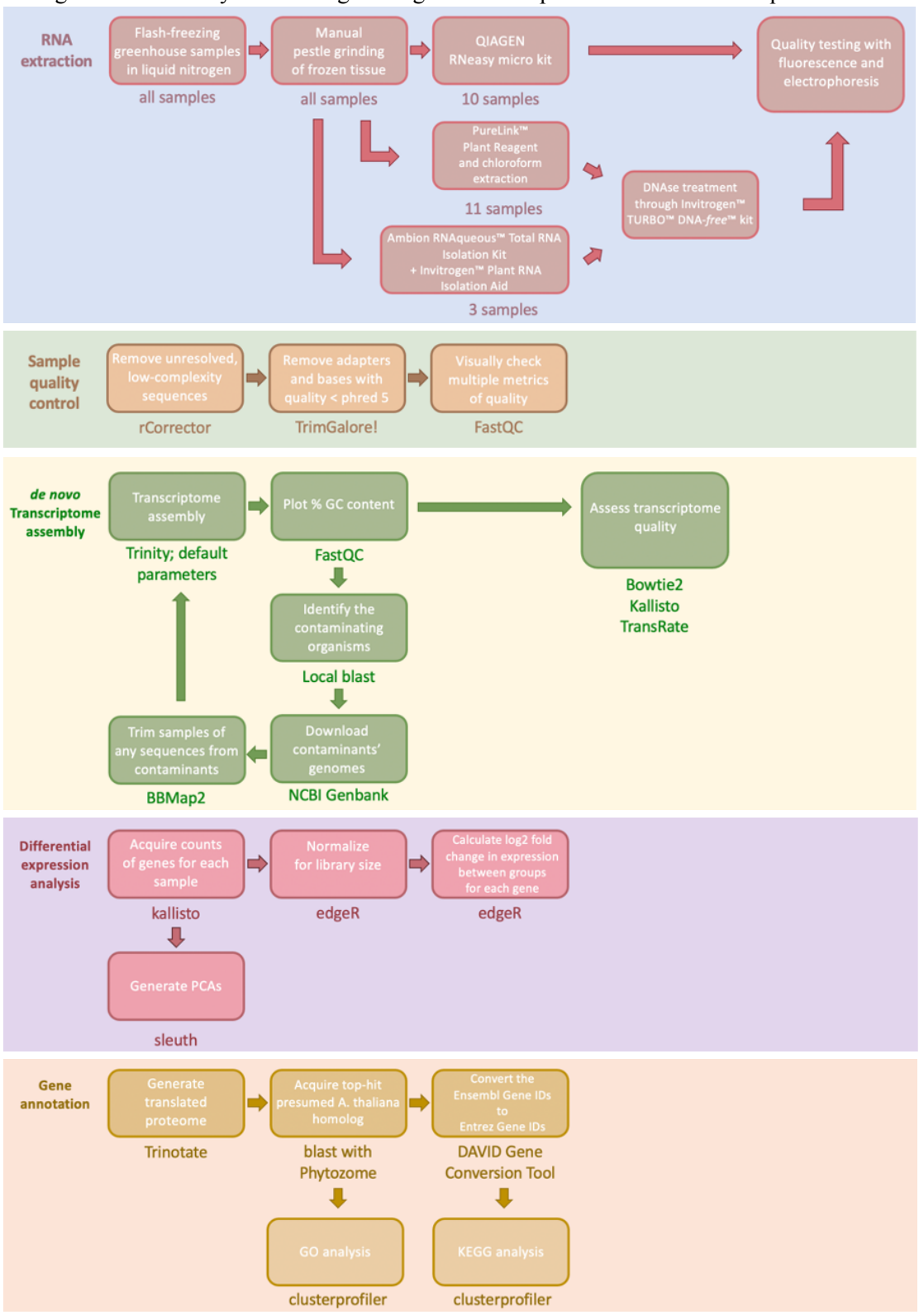
Abagail Burrus
Kramer Lab, July 7, 2022

De novo transcriptome assembly guide

This guide covers the yellow background/green-boxed portion of this RNA Seq workflow:



This guide assumes you have previous experience with your sequences on Harvard's RC cluster computing system.
If you want a refresher on cluster usage, please consult the guide here: "RNA sequence quality guide + primer on the Cluster"

Initial *de novo* transcriptome assembly

When I assembled my sequences, I chose the program Trinity for it. Trinity remained state-of-the-art up until that point; maybe it's been surpassed by now, but it did pretty well with my sequences and it works on the Cluster. Here's an older guide on using it that RC published: https://informatics.fas.harvard.edu/best-practices-for-de-novo-transcriptome-assembly-with-trinity.html

- **Make a bash file for the program named trinity.sh**

```
#!/bin/sh
# use an entire node in the specified Slurm partition
#SBATCH --nodes=1
#SBATCH --mem=0
#SBATCH --exclusive
# Adjust wall time limit as appropriate for partition. If the job is cancelled
# due to time limit exceeded, the job script can be resubmitted, and Trinity
# will resume execution after the last completed.
#SBATCH --time=72:00:00
# Resubmit to the "bigmem" partition if inchworm std::bad_alloc error occurs.
# The job will then stop after inchworm completes, and can be resubmitted to
the
# "shared" or "test" partition, which are preferred for faster / more reliable
# execution and sooner job start time.
#SBATCH --partition=shared

set -o nounset -o errexit -o xtrace

#######################################
# parameters
#######################################
readonly
SINGULARITY_IMAGE=/n/singularity_images/informatics/trinityrnaseq/trinityrnaseq
.v2.12.0.simg
readonly TRINITY_OUT_DIR=trinity_out_dir
# To see all options:
#    singularity exec --cleanenv ${SINGULARITY_IMAGE}  Trinity
--show_full_usage_info
readonly TRINITY_OPTIONS="--output ${TRINITY_OUT_DIR} --max_memory
$((8*$(ulimit -m)/(1024**2)/10))G --CPU ${SLURM_CPUS_ON_NODE} $@"

#######################################
# ... don't modify below here ...
```

```
if [ ! -s "${TRINITY_OUT_DIR}/read_partitions.img" ]
then
  mkdir -p "${TRINITY_OUT_DIR}"
  readonly tmpdir=$(mktemp -d)
  mkdir -m 777 -p ${tmpdir}/upper ${tmpdir}/work
  truncate -s 2T "${TRINITY_OUT_DIR}/read_partitions.img"
  singularity exec --cleanenv ${SINGULARITY_IMAGE} mkfs.ext3 -d "${tmpdir}"
"${TRINITY_OUT_DIR}/read_partitions.img"
  singularity exec --cleanenv --overlay ${TRINITY_OUT_DIR}/read_partitions.img
${SINGULARITY_IMAGE} mkdir /read_partitions
  ln -sf /read_partitions ${TRINITY_OUT_DIR}/read_partitions
  rm -rf "${tmpdir}"
fi

# if on a bigmem node, stop after inchworm
case ${SLURM_JOB_PARTITION} in
  bigmem) no_run_chrysalis='--no_run_chrysalis' ;;
       *) no_run_chrysalis='' ;;
esac

srun -n 1 env time -v singularity exec \
                            --cleanenv \
                            --no-home \
                            --overlay ${TRINITY_OUT_DIR}/read_partitions.img \
                            "${SINGULARITY_IMAGE}" \
   Trinity ${TRINITY_OPTIONS} ${no_run_chrysalis}
```

- **Running Trinity**

Trinity is best run on the holyscratch system, like all big jobs.
The format for submitting this job is as follows:

```
sbatch trinity.sh --seqType fq --left $comma-separated R1 fastq files --right
$comma-separated R2 fastq files
```

Here's an example of what that would look like:

sbatch trinity.sh --seqType fq --left sample1_1.fq,sample2_1.cor_val_1.fq,sample3_1.cor_val_1.fq --right sample1_2.cor_val_2.fq,sample2_2.cor_val_2.fq,sample3_2.cor_val_2.fq

Note the comma and no spaces between the right sequences, and note the space only being present between your parameters in the submission:

sbatch trinity.sh --seqType fq --left sampleone_1.fq,sampletwo_1.fq,samplethree_1.fq --right sampleone_2.fq,sampletwo_2.fq,samplethree_2.fq

Note also that the “left” list corresponds to the samples ending in _1.fq (forward reads), and the “right” list corresponds to the samples ending in _2.fq (reverse reads)
You can directly reference the file names of your sequences if you have your trinity.sh file (and run the job) in the same location/same folder. If you have your sequences elsewhere, you have to include the paths in your job submission, like this for example:

```
sbatch trinity.sh --seqType fq --left
~/path/to/sample1_1.fq,~/path/to/sample2_1.cor_val_1.fq,~/path/tosample3_1.cor_val_1.fq --right
~/path/to/sample1_2.cor_val_2.fq,~/path/to/sample2_2.cor_val_2.fq,~/path/to/sample3_2.cor_val_2.fq
```

Trinity is a BIG job to run, so it will probably sit in the queue for a while before it gets to actually run (you can keep track of it using the *sacct* command or *scontrol show job $jobid*)

If something went wrong, it’s likely going to be frustrating that you had to wait a DAY or a FEW DAYS of the job sitting in the queue before finding out something was wrong! It happens. Often if it happened to me, it was some typo in the input sample names.

If the job runs out of operational time (TIMEOUT), just run it again – it automatically kicks off from a checkpoint within the protocol that it reached before the timeout.
This detail and memory error handling is described at the top of the trinity.sh file code (see the file you made above)

If the job ran successfully, you’ll see a number of files in a new directory where you ran the job. The new directory is called Trinity_out_dir, and if you check the sizes of the files in that directory (like by using *ls -l*), your transcriptome will be Trinity.fa and should have a non-zero byte file size. (It’s probably a really big file.)

Checking transcriptome quality

- **Using basic Trinity assembly metrics**

Use the following command within that Trinity_out_dir:

```
srun --pty -p shared -t 00:20:00 --mem 500 /bin/bash
singularity exec --cleanenv /n/singularity_images/informatics/trinityrnaseq/trinityrnaseq.v2.12.0.simg sh -c
'$TRINITY_HOME/util/TrinityStats.pl Trinity.fasta' > Trinity_assembly.metrics
```

Then use nano, vim, etc. look at Trinity_assembly.metrics- it should look something like this:

```
################################
## Counts of transcripts, etc.
################################
Total trinity 'genes':  193248
Total trinity transcripts:      350461
Percent GC: 40.61

########################################
Stats based on ALL transcript contigs:
########################################

        Contig N10: 4713
        Contig N20: 3667
        Contig N30: 3048
        Contig N40: 2553
        Contig N50: 2137
```

```
       Median contig length: 615
       Average contig: 1161.23
       Total assembled bases: 406964301
```

```
#################################################
## Stats based on ONLY LONGEST ISOFORM per 'GENE':
#################################################

       Contig N10: 4081
       Contig N20: 2995
       Contig N30: 2280
       Contig N40: 1679
       Contig N50: 1116

       Median contig length: 348
       Average contig: 669.81
       Total assembled bases: 129438767
```

Ideally, you'll have a large number of 'genes' and 'transcripts' - and generally, the higher N50 the better. The N50 number represents the length of a contig in base pairs (nucleotides). N50 in this case is a proxy for how contiguous your transcripts are; you don't want a really low N50 because that means at least 50% of your contigs are that really low number long, and that indicates a highly fragmented transcriptome. A highly fragmented transcriptome likely doesn't represent the source tissue well, among other issues - you're probably missing a lot of the important genes you want.

- **Using bowtie2**

A lot of this is directly from the tutorial I linked above:
First, make a bowtie1.sh script:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 4 #Number of cores
#SBATCH -t 60:09:00  #Runtime in minutes
#SBATCH -p shared  #Partition to submit to
#SBATCH --mem=8000  #Memory per node in MB
#SBATCH -e bt2build%A.e
```

```
#SBATCH -o b2build.o

# $1 = your assembly fasta
assembly_prefix=$(basename $1 |sed 's/.fasta//g')

readonly SINGULARITY_EXEC='singularity exec --cleanenv /n/singularity_images/in$

${SINGULARITY_EXEC} bowtie2-build --threads 4 $1 $assembly_prefix
```

Also make a bowtie2.sh script:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 16 #Number of cores
#SBATCH -t 12:00:00  #Runtime in minutes
#SBATCH -p serial_requeue,shared  #Partition to submit to
#SBATCH --mem=16000  #Memory per node in MB

readonly SINGULARITY_EXEC='singularity exec --cleanenv
/n/singularity_images/informatics/trinityrnaseq/trinityrnaseq.v2.12.0.simg'

# $1 name of your assembly (without the .fasta suffix)
# $2 comma separated list of left read file names
# $3 comma separated list of right read file names

   ${SINGULARITY_EXEC} bowtie2 -p 10 -q --no-unal -k 20 -x $1 -1 $2 -2 $3
   2>align_stats.txt| ${SINGULARITY_EXEC} samtools view -@10 -Sb -o bowtie2.bam
```

The bowtie1.sh script builds an index for the assembly, and then bowtie2.sh maps the reads and calculates assembly statistics

Run bowtie1.sh with:
sbatch bowtie1.sh Trinityfilenamehere.fasta
Then run bowtie2.sh with:
sbatch bowtie2.sh Trinityfilenamehere
sample_one_file_1.fq,sample_two_file_1.fq,sample_three_file_1.fq
sample_one_file_2.fq,sample_two_file_2.fq,sample_three_file_2.fq

Check on bowtie2.sh's results with align_stats.txt, and you should see something similar to this:

```
765496725 reads; of these:
  765496725 (100.00%) were paired; of these:
    36963974 (4.83%) aligned concordantly 0 times
    105856674 (13.83%) aligned concordantly exactly 1 time
    622676077 (81.34%) aligned concordantly >1 times
    ----
    36963974 pairs aligned concordantly 0 times; of these:
      1491173 (4.03%) aligned discordantly 1 time
    ----
    35472801 pairs aligned 0 times concordantly or discordantly; of these:
      70945602 mates make up the pairs; of these:
        16495472 (23.25%) aligned 0 times
        10975538 (15.47%) aligned exactly 1 time
        43474592 (61.28%) aligned >1 times
98.92% overall alignment rate
```

Higher overall alignment rate is better!

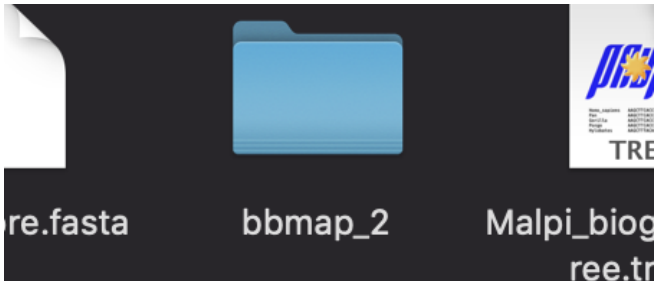
- **Using FASTQC**

Just as FASTQC helped us visualize sample quality, we'll use a trick to make it plot our transcriptome's GC content for us.

FASTQC takes input files of .fq format, but the transcriptome is in FASTA format, Trinity.fa - .fq files have quality scores for each base, so we will reformat our transcriptome with a supplied false quality score:

First, we need our tool for reformatting the file: BBMap, a suite of useful scripts. You can download this Java set of scrips here:
https://sourceforge.net/projects/bbmap/

I downloaded this to my home system directly from the browser, then installed it from that downloaded file, and this made a folder of BBMap files, in my case called bbmap_2:



(You don't need to install anything else special for BBMap, only Java.)

I then downloaded the transcriptome from the Cluster into that same folder.

Using Terminal locally, I used *cd* to navigate into the bbmap_2 folder, then submit:

./reformat.sh in=Trinity.fasta out=Trinity.fq qfake=30

This Trinity.fq is our reformatted transcriptome, and we then use our file transfer system to move Trinity.fq back onto the cluster.

Yes, we could have also run BBMap on the cluster directly (as we'll do later), but this method made us essentially produce a backup of our transcriptome when we downloaded it - and this is always good!

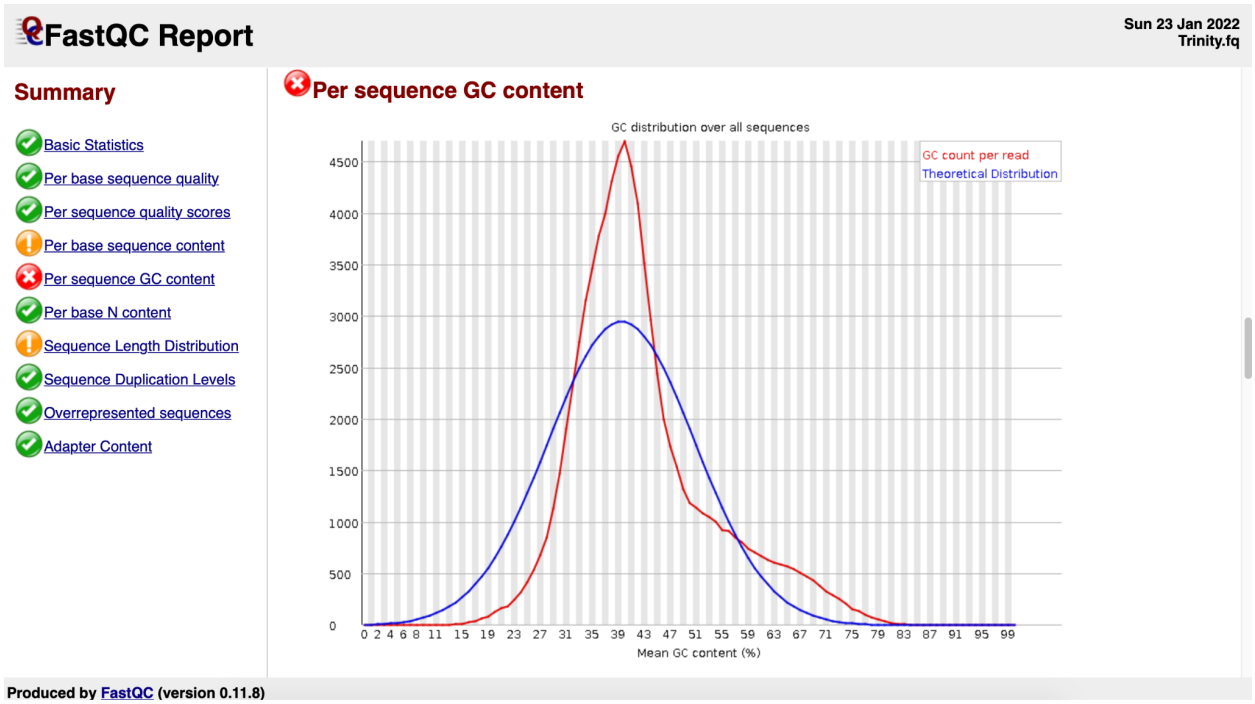
If you don't already have the FASTQC script from the RNA sequence quality guide, make it with this code:

```
#!/bin/bash
#SBATCH -p serial_requeue        # Partition to submit to
#SBATCH -N 1
#SBATCH -n 6                     # Number of cores
#SBATCH -t 0-3:00                # Runtime in days-hours:minutes
#SBATCH --mem 6000               # Memory in MB
#SBATCH -J FastQC                # job name
#SBATCH -o FastQC.%A.out         # File to which standard out will be written
#SBATCH -e FastQC.%A.err         # File to which standard err will be written
#SBATCH --mail-type=ALL          # Type of email notification-
BEGIN,END,FAIL,ALL
#SBATCH --mail-user=<PUT YOUR EMAIL ADDRESS HERE>  # Email to which
notifications will be sent

readonly SINGULARITY_IMAGE='singularity exec --cleanenv
/n/singularity_images/informatics/trinityrnaseq/trinityrnaseq.v2.12.0.simg'

   ${SINGULARITY_EXEC} fastqc --threads ${SLURM_CPUS_ON_NODE} --outdir `pwd` $1
```

Then run the script on the assembly as:

sbatch myfastqcscript.sh Trinity.fq

In looking at your results (through the .html file that the job gives as output), look at the GC content plot:



That trailing tail to the right of the read distribution is indicative of contamination in the samples that became this transcriptome. (Plant GC content is typically 40%, so sequences that have GC content other than 40% are likely not plants.)

Because *de novo* transcriptome work by definition lacks a reference genome, we want to be really confident that the genes of interest we find are due to actual biological signal from our desired tissue, not a gene of some pest or pathogen.

Decontaminating your samples

There are many ways to decontaminate your samples if they seem to be throwing off the transcriptome (like we see above).

The way I recommend doing it is BLAST.
I did it a really tedious way, but it worked for me.

You need to identify the contaminants in your transcriptome, so I did the following:

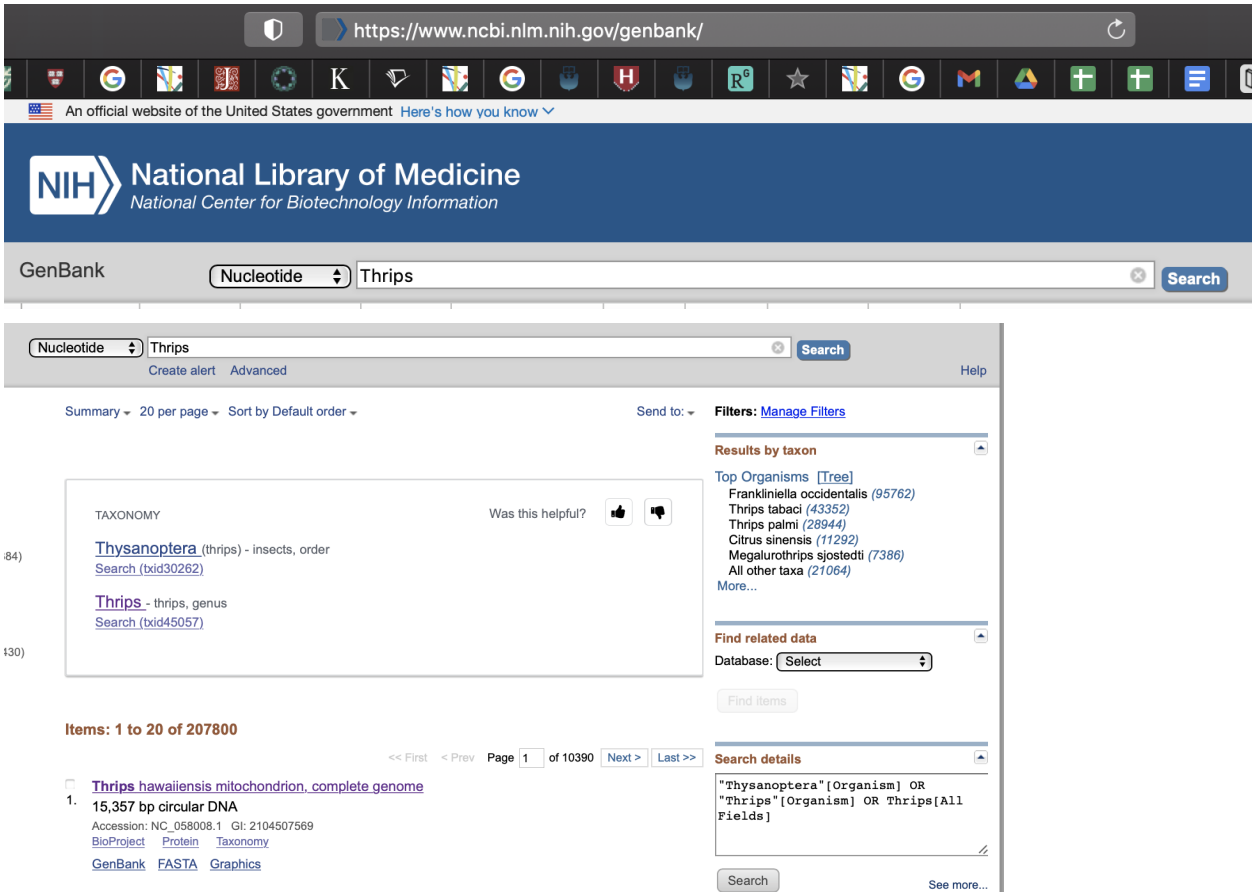
- I used a text editor to select portions of the transcriptome (Trinity.fasta) and pasted each chunk into its own new text file, creating like 30 different portions of the transcriptome (part 1, part 2, part 3…)
- I submitted each portion to BLAST in-browser
- I looked at the results of each BLAST for any species that weren't the one I directly studied - there will probably be some arthropods, mammals, bacteria, fungi, etc.

- I made a list of the genus + species names for these BLAST results

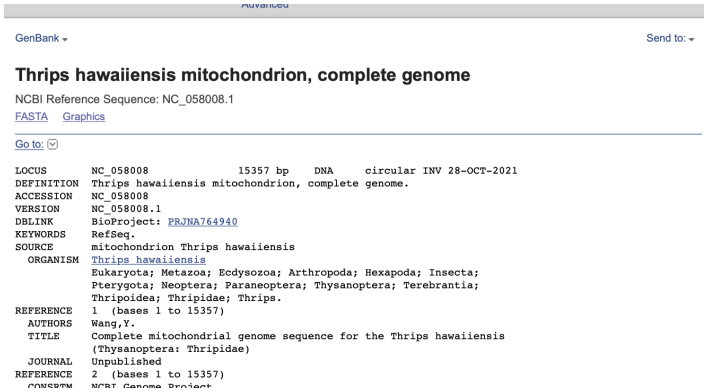
It was very tedious, like I said, but as long as you BLAST a good amount of your transcriptome, you'll likely get a good representation of what else is in those samples. (You can always go back and BLAST more if you still seem to have contamination issues after we eventually check our transcriptome with FASTQC again. Orrr you probably have another way of doing this.)

Now we need to acquire the genomes of our contaminants, so we know which sequences to filter out of our samples.

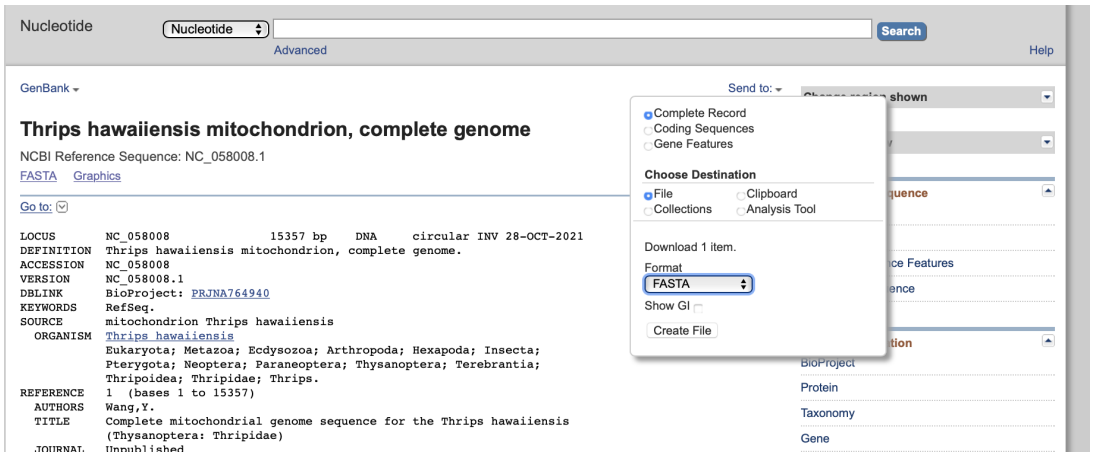
- I searched for the genus + species (sometimes performing this for many species from that same genus, just to be safe) on NCBI's Genbank, here: https://www.ncbi.nlm.nih.gov/genbank/
- I downloaded the full genome of each contaminant in my list, directly from Genbank, illustrated below with an example of having searched for *Thrips*:



I tried to target entries that had complete genomes - good coverage of what you want to filter out of your samples. I clicked that first result and from here:



I then clicked “Send to:” in the top right corner and selected these options:



This sent a FASTA file of the genome to my Downloads folder.

With this repeated for each identified contaminant, I had a big set of files with a genome of each contaminant.

At this point, we have identified the contaminants, we have their genomes, and we have our samples (the ones that went into our last transcriptome and probably have contamination). The next step is filtering our samples, and we accomplish this again through the BBMap suite.

- In credit, a scientist I was never actually introduced to named Dale (affiliation unknown) said I should use the BBMap suite’s bbduk script- and BBMap was a great idea for this, but the bbduk script is actually a lot less efficient for this purpose than the bbsplit script!
- Both scripts do the following: it takes a sample to scan, and then it uses a specific contaminant genome as the reference for which it’s scanning. For example, I give it sampleone_1.fq and the specific contaminant Thripsgenome.fasta. It then looks for any sequences in sampleone_1.fq that match sequences in Thripsgenome.fasta, and matching portions are removed from sampleone_1.fq… But this is where they differ:
    - Bbduk.sh compiles a list of your filtered-out reads (the sample bits that matched the contaminants) along with producing your decontaminated sample.
    - Bbsplit.sh just removes those filtered-out reads and the only output is your decontaminated sample.

With bbduk, you end up with extra files you have no use for, and it works a lot slower for huge files (like whole genomes) than bbsplit does.

Copy your bbmap_2 directory of scripts over to your cluster space, and do one of the following:

(1) Either know the path names to all your sample files and contaminant genomes, OR
(2) Upload/transfer your contaminant genomes into this directory and then copy over your sample files into this directory.

There is probably a faster way to do this in array jobs, but formatting that is hard and it's educationally a lot clearer if I walk you through the detailed job process… even though that unfortunately means submitting a different job for each sample getting scanned (I actually ran my decontamination in two rounds of jobs because I had so many contaminants and many of them were eukaryotes, which have wonderfully large genomes- big genomes and more contaminants means more memory and time demands for your jobs).

So, in your directory that has all those components in it, create a series of job scripts formatted like the following:

```
#!/bin/bash

#SBATCH -J samplename
#SBATCH -n 3
#SBATCH -t 0-13:00
#SBATCH -p shared
#SBATCH --mem=17000
#SBATCH -o sample1.out
#SBATCH -e sample1.err

module purge
module load GCC/7.3.0-2.30 OpenMPI/3.1.1 BBMap/38.35

bbsplit.sh in1=sampleone_1.fq in2=sampleone_2.fq
ref=Serratiamarcescens.fasta,Serratiaureilytica.fasta,Cutibacterium.fasta,moraxellaosloensis.fasta,thrips.fasta,ecoli.fa
sta,planococcus.fasta,pseudococcus.fasta,paracoccus.fasta,Acinetobacter2.fasta basename=out_%.fq
outu1=sampleone_1clean.fq outu2=sampleone_2clean.fq
```

You can save it as anything - firstdecontamination.sh, for example

Then create another job script for each sample, such as seconddecontamination.sh here for example:

```
#!/bin/bash

#SBATCH -J samplename
#SBATCH -n 3
#SBATCH -t 0-13:00
#SBATCH -p shared
#SBATCH --mem=17000
#SBATCH -o sample2.out
```

```
#SBATCH -e sample2.err

module purge
module load GCC/7.3.0-2.30 OpenMPI/3.1.1 BBMap/38.35

bbsplit.sh in1=sampletwo_1.fq in2=sampletwo_2.fq
ref=Serratiamarcescens.fasta,Serratiaureilytica.fasta,Cutibacterium.fasta,moraxellaosloensis.fasta,thrips.fasta,ecoli.fa
sta,planococcus.fasta,pseudococcus.fasta,paracoccus.fasta,Acinetobacter2.fasta basename=out_%.fq
outu1=sampletwo_1clean.fq outu2=sampletwo_2clean.fq
```

If your samples or contaminants are in other places, you have to include the paths to the files, like I give in example here:

```
#!/bin/bash

#SBATCH -J sample1
#SBATCH -n 3
#SBATCH -t 0-13:00
#SBATCH -p shared
#SBATCH --mem=17000
#SBATCH -o sample1.out
#SBATCH -e sample1.err

module purge
module load GCC/7.3.0-2.30 OpenMPI/3.1.1 BBMap/38.35

bbsplit.sh in1=/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
sampleone_1.fq in2=/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
sampleone_2.fq ref=/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
Serratiamarcescens.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
Serratiaureilytica.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
Cutibacterium.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
moraxellaosloensis.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
thrips.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
ecoli.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
planococcus.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
pseudococcus.fasta,paracoccus.fasta,/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/
Acinetobacter2.fasta basename=out_%.fq
outu1=/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/sampleone_1clean.fq
outu2=/n/holyscratch01/davis_lab/aburrus/testdata/corrected/output/sampleone_2clean.fq
```

You then run each job as so:

```
sbatch firstdecontamination.sh
sbatch seconddecontamination.sh
```

Etc.

Once these jobs are done, take these cleaned samples, then start this document over again - redo the transcriptome assembly with Trinity, using your clean samples as input instead of your old samples. (Copy over your Trinity.sh script to your clean samples' directory with *cp*

old/path/to/trinity.sh /new/place/you/want/trinity.sh - do NOT run Trinity in the same directory you did the first time. Trinity doesn't like that.)

This new Trinity run will make a new Trinity_out_dir (output directory of the job) wherever you submitted the trinity.sh file

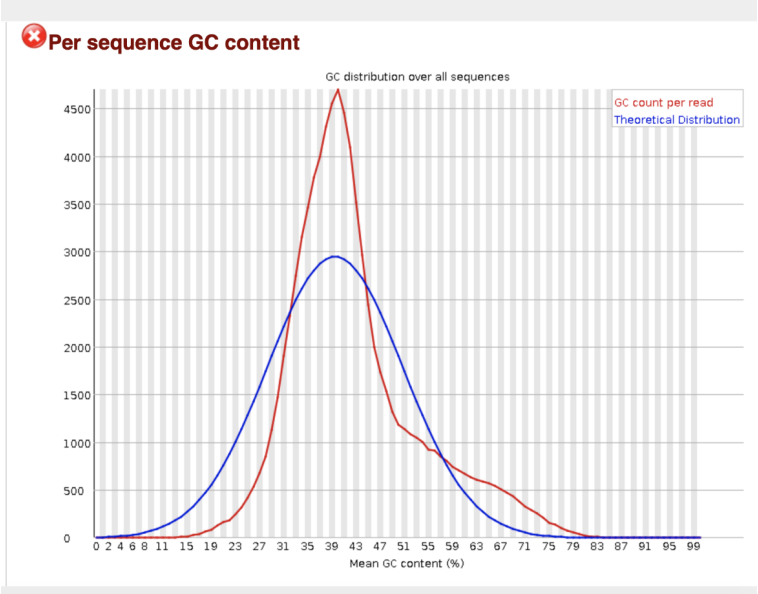
Repeat the FASTQC trick with your new Trinity.fasta transcriptome (maybe name it something special so you don't get it mixed up with your old transcriptome)

Here's how I did that reformatting trick on the cluster, submitting a script named reformattingTrinity.sh with this content:

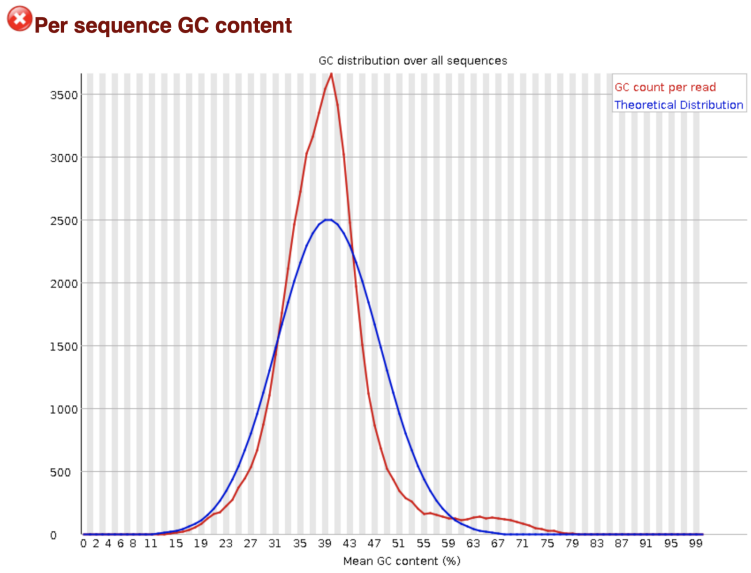
```
#!/bin/bash

#SBATCH -J reformat
#SBATCH -n 5
#SBATCH -t 0-03:00
#SBATCH -p shared
#SBATCH --mem=20000
#SBATCH -o reformat.out
#SBATCH -e reformat.err

module purge
module load GCC/7.3.0-2.30 OpenMPI/3.1.1 BBMap/38.35

reformat.sh in=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta
out=/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fq qfake=30
```

Don't forget to customize that to your own file pathnames!

Here's an example of how much my transcriptome improved with decontamination:



<- old transcriptome



<- new transcriptome. Better!!

With this done, I took my new transcriptome through the assembly_metrics and Bowtie2 quality testing methods I saw above. I also used Transrate to assess the quality.

- TransRate

The module for TransRate is already available on the cluster, so I made a file named translate.sh as follows:

transrate.sh:

#!/bin/bash

```
#SBATCH -J transrate
#SBATCH -n 30
#SBATCH -t 0-13:00
#SBATCH -p shared
#SBATCH --mem=37000
#SBATCH -o transrate.out
#SBATCH -e transrate.err

module purge
module load transrate/1.0.1-fasrc01

transrate --assembly /n/holyscratch01/davis_lab/aburrus/trinity_out_dir/Trinityredone.fasta --left
sampleone_1clean.fq,sampletwo_1clean.fq,samplethree_1clean.fq --right
sampleone_2clean.fq,sampletwo_2clean.fq,samplethree_2clean.fq --output
/n/holyscratch01/davis_lab/aburrus/trinity_out_dir/results_directory
```

(IF you try running this and it doesn't like you putting it in "results_directory" or another new directory because the directory doesn't exist yet, try going and making mkdir directory_name_here where you want it to be- maybe you don't need to put anything in it. Or maybe make a single txt file with a single space in it in there just so it isn't a totally empty directory. Try the fixes in that order. You probably won't need to try any of them, but it's best to know your options.)

TransRate gives you an output file with a bunch of useful statistics in it regarding your transcriptome's quality. Here's the manual of the metrics, to tell you what all the different numbers mean:
https://hibberdlab.com/transrate/metrics.html