

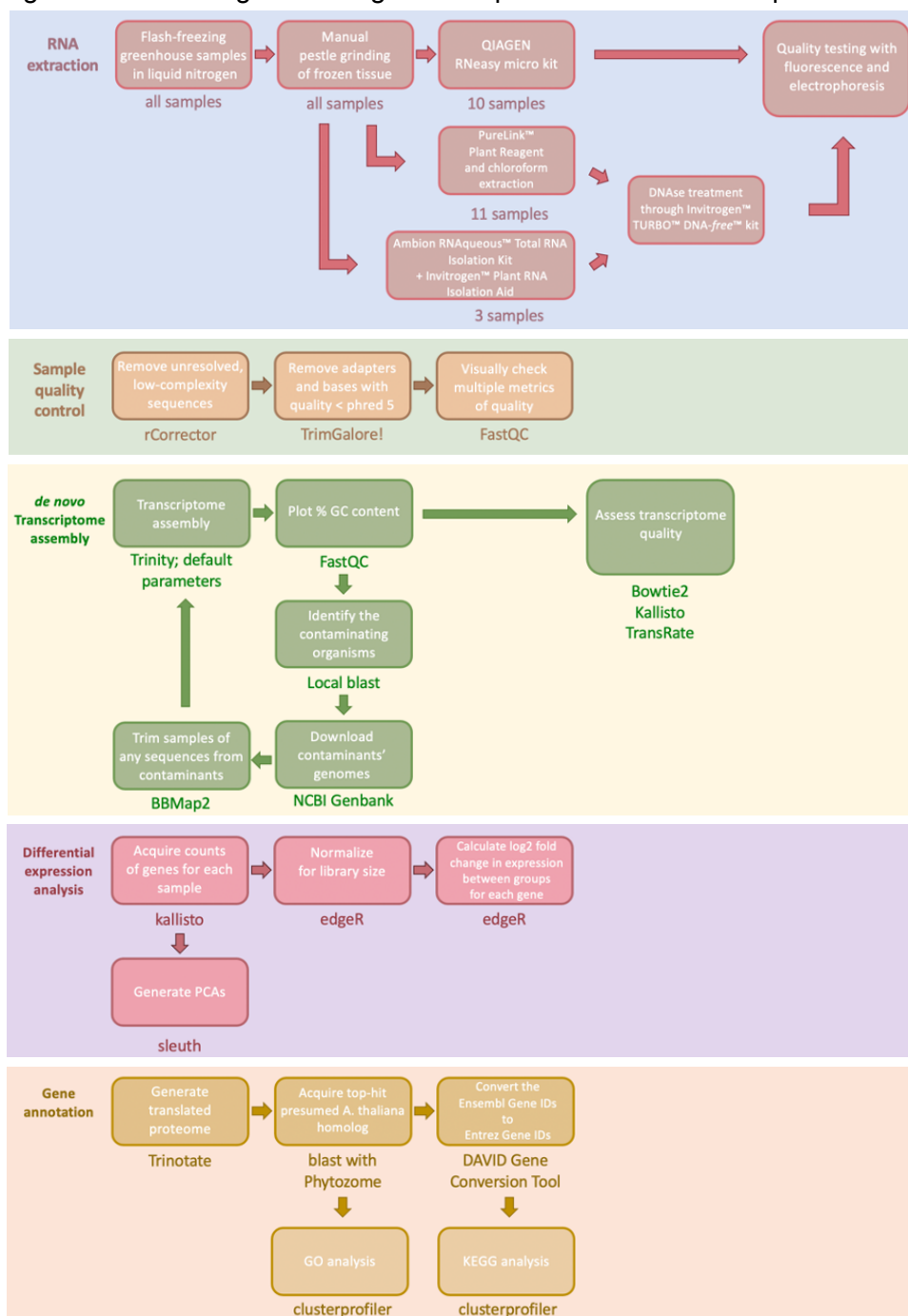
RNA sequence quality guide + primer on the Cluster

Abigail Burrus

February 10, 2022

This brief guide is for people new to the cluster (or very rusty, like I was!) who have received RNA sequences back from the Bauer Core, Novogene, or other company.

This guide covers the green/orange-boxed portion of this RNASeq workflow:



If you are assessing RNA sequences that you intend to align to a reference genome, I would skip to “Trimming Adapters and Low-Quality Sequences.”

If you’re confident with Cluster use and intend to perform *de novo* transcriptome assembly, I would skip to “Assessing quality of sequences: FASTQC” or straight to “What People Look For in FASTQC.”

If you are assessing RNA sequences that you intend to use in *de novo* transcriptome assembly, this full guide can probably work for you. Be aware that more trimming of your data in this guide may ultimately lead to rare genes getting edited out. (The RC group here at Harvard probably has more updated workflows than this one!)

PLEASE KEEP A RUNNING SCRIPT OF WHAT YOU DO AND WHERE YOUR FILES ARE GOING. You need to revisit certain places where certain files are, so if you keep track of that from the get-go in something like text file or an R script (I just use my Notes app, as you can see on the left), you can say “ahh those files must be in _ because I moved to that area before I ran this job,” etc.

As an overview, treating your RNA sequences at this point involves steps such as looking at its quality metrics, trimming out its adapters, and removing unhelpful fragments of your samples that would only slow down your downstream processes.

Getting Your RNA Sequences

I’m not much help at this part. The company/group from whom you got your sequences (for example, I sent my samples to Novogene and got my sequences back from them) should include instructions on how to download your sequences. But here’s a few pointers on it anyway, since part of it will come in handy with the next step:

File transfers: The company/group is likely hosting your files on a remote server somewhere, and you need to connect to that location through FTP transfer.

FTP, File Transfer Protocol, is basically a communication system that will let you, as a client, access files/information on a server (the host) in a secure way.

I recommend using FileZilla for your FTP needs - it’s totally free, and it’s highly successful in use with Harvard’s cluster system. Download the standalone desktop version as “FileZilla Client” [here](https://filezilla-project.org). (<https://filezilla-project.org>)

Assessing Quality of Sequences: FASTQC

The .fq format of your sequences denotes that they are in FASTA format (which is typically .fa or .fasta) and have quality scores attached to each read (making them “fasta with quality scoring”, or .fq, or FASTQ).

(Most computers don’t innately know what to do with .fq or .fa file extensions for the first time—direct your computer to open these files with a text editor such as SublimeText, Notepad++, or TextEdit if you ever have to look at them.)

Fasta files can be recognized when you open them by looking similar to this:

```
>TRINITY_DN176378_c0_g1_i1 len=211 path=[0:0-210]
CTCAGTGGTTGAGCATTGACTGCAAAATATTCTCAAATCTACCAGTCTGCACCACTATCATCATGAGTGGTAACAATAATGTCTAGTTTTGTGTTA
TCTTTATTTTAAGAAATCAGATAGGTATTCTTAGGTCTCCACCCTTCAGTCA
>TRINITY_DN176351_c0_g1_i1 len=271 path=[0:0-270]
CTTTTCCTTTGGTACTTTCTTCTAACTGATTCCAGTTTAGACAAGTCAGATTTTGCAGGCTCCGTCTGAGAGACCTGAAGTTTCTGGTCCAGGGCAGA
GACCTTTCCCTCTTTCTTTCACTCGAGTCAAACACGCAAGGCATGAGTTCCAGGCGGTTTTTCGCTATTCCTCTTTGTCAGCTTTCTCTCTGCT
>TRINITY_DN176361_c0_g1_i1 len=257 path=[0:0-256]
GCCACTTCTGTCACTGCCACTCCACGACGCCCTCGGGCCTGCACCCCTATCACTCTGGAGCCACCAGCTCCAGGGCTGAAAAGGGGCCGGGAGGGC
AGCACAGGAGGGCCTCTGTGTCAGGGCCCTTCCCGGACCCCAAGGCTTGTCTTCAGCAGCGGGTCCAGGGAGCTGCTGGGAGCAGAAGTACGCAC
>TRINITY_DN176355_c0_g1_i1 len=257 path=[0:0-256]
AATTTAAGTGACACAGACATTTTGTGTGAGACTTAAACATGGTGCCTATATCTGAGAGACCTCTGTGACTTACCGAGAAGATGTGAACAGCTCCCTTC
AGAATGGATGTTTTTCTTTATATGGTTGTCTGGATTTTTTTTAAAGTAATTCATCAGTTCAGTTTATCCACCTCATCATTAATAAATGAAGGATGC
```

There’s a > followed by some identifying information, and on the next line is the string of your nucleotides that’s the actual read.

That’s from my fasta above, but a fastq itself has single-byte encoding of its quality score put in like a substitution cipher after its sequence. Here’s an example from the Wiki:

A FASTQ file containing a single sequence might look like this:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAAATAGTAAATCCATTGTGTTCAACTCAGCTTT
+
!''*((((***+))%%%+)(%%)).1**+!*'')**55CCF>>>>>CCCCCCC65
```

The byte representing quality runs from 0x21 (lowest quality; '!' in ASCII) to 0x7e (highest quality; '~' in ASCII). Here are the quality value characters in left-to-right increasing order of quality (ASCII):

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

What looks like nonsense of !''*((etc. is really the quality scoring. Don’t think you have to memorize any of this - I just wanted what you’re looking at to make more sense to you.

Beyond it being useful to assess what type of file you’re looking at when you examine some data for the first time (especially when some helpful bioinformaticians send along files with no extension and you have to figure out yourself what extension that file needs to make sense and open properly in downstream applications), it’s great that we have FASTQ files like this because we can use the program FASTQC (like “fasta quality control”) to look at those quality scores and many other quality metrics of our samples.

To use fastqc within the cluster, we submit a job to the cluster that runs this task for us. We command this job by feeding a specific “bash” format script to the server. In the de novo guide I base this off of, [here](#), you can see the format of the scripts like this:

Use [fastqc](#) to examine quality metrics from your reads. An example slurm submission script is as follows:

```
#!/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/informat
ics/trinityrnaseq/trinityrnaseq.v2.12.0.simg'

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

The #SBATCH information along the top lines are all parameter specifications for your job. You're telling the cluster what partition to submit your job to, how many computing nodes to use, how many cores of computing to use, the MAX runtime you're allowing your job to have, The MAX memory in MB of your job, Your job's name, And what various customized names you want to give the job's standard output file, its error file (that's very useful!), etc. You don't HAVE to have it email you in these two:

```
#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
```

But if you want to keep a close eye on the job progress without having command-line open or you want to keep an eye on it while being away from keyboard, these email options are useful.

Everything below the #SBATCH parameter lines,

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

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

Which is THIS for the fastqc script, is the actual scripting specific to the job.

You don't have to generate this off the top of your head at any point in this guide. You just need to know how to put this information into your file in the cluster, and how to use its \$1, \$2, \$3 mentions in various files to feed it specific samples. (You'll see shortly, don't stress.)

So how do we get this script running for our samples in our cluster space? If the module/program you want to use is already in the cluster space (you can check if the software already implemented by searching [here](#)), then it's pretty simple. Fastqc is an example of the many modules already available on the cluster.

Making a Script/File on the Cluster

A fun trick in the cluster is that you can use a command to edit a file and, if that file doesn't exist yet, it will *create* that file too!

Just like there are multiple text editing programs for download around the internet, like SublimeText or Notepad++, there are a number of useful text editors in the cluster space.

My go-to program is called "nano". It's the easiest to pick up. (If you ever need to look at higher depth at your scripts, like trying to check if the designated classes of parts of your script are correct, I'd use vim instead of nano. But I'm not well-versed in vim and you'll need to look around RC's webpages some to figure that out. I DO know that if you open a file with vim and need to close that file, just type in :q and hit enter.)

If I want to edit a file named "mysequences" in my cluster space, I would just type in:

nano \$filename

For example:

```
HUIT-FAS-Abagails-MacBook-Pro:~ abagailburrus$ nano mysequences.fa
```

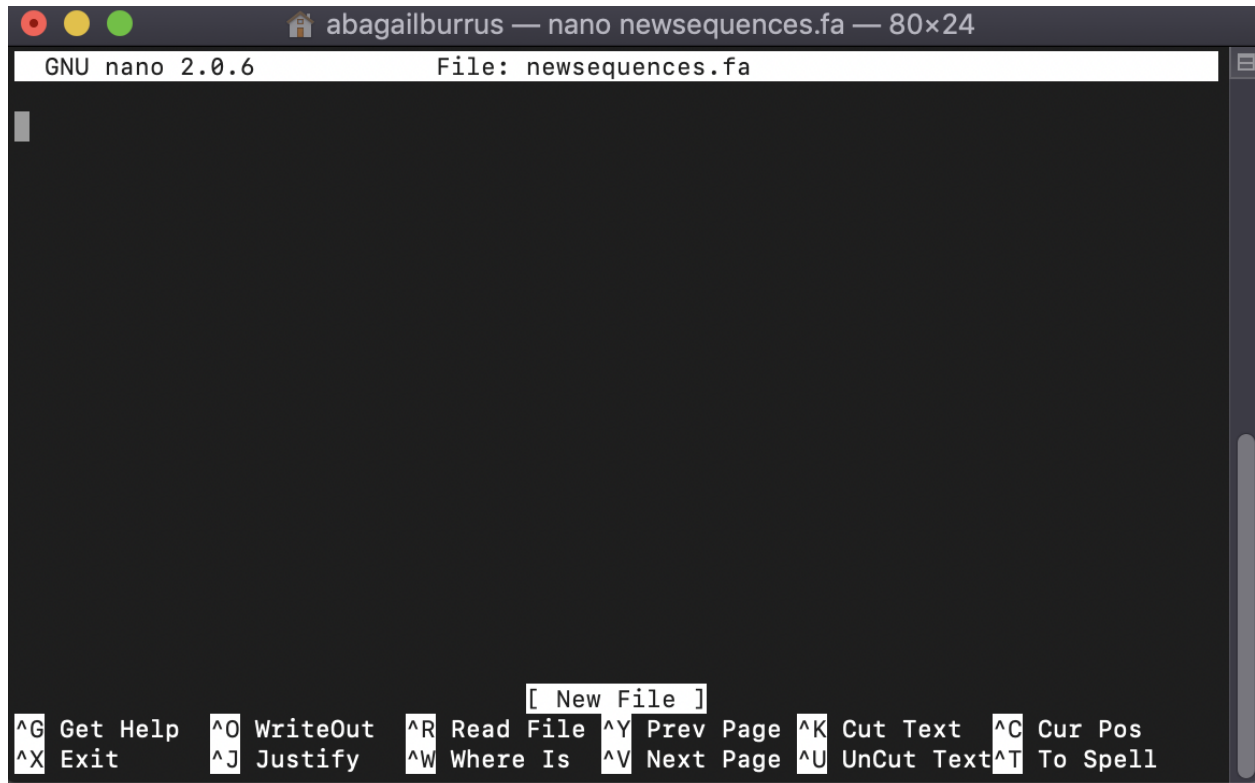
I would hit enter, and then the file would open within the terminal window I've been entering my code.

If I wanted to MAKE a file named "newsequences" in my cluster space, I would just type in:

```
:~ abagailburrus$ nano newsequences.fa
```

The NEW file I make (by opening it there) will be empty.

If you use nano to open a file you already have there, and it shows up empty like this:



The screenshot shows a terminal window with the nano text editor open. The title bar at the top reads "abagailburrus — nano newsequences.fa — 80x24". The editor's status bar at the top left says "GNU nano 2.0.6" and the top right says "File: newsequences.fa". The main editing area is empty. At the bottom, the nano help menu is displayed, showing various keyboard shortcuts for navigation and editing. A "[New File]" prompt is visible in the center of the help menu.

```

[ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell

```

You probably have one of the three problems:

- (1) A typo in your nano \$filename command, like you typed in "nano mysqeunces.fa" by mistake
- (2) Your file is actually in another directory - you need to use the cd command and navigate to that file's directory in order to open it with "nano"
- (3) You're looking for a job's output file that hasn't been made yet because something went wrong in the job or it isn't finished yet

Okay, let's finally make a script file for using Fastqc.

Type in “nano whateveryouwanttonamethatfile.fa” and hit enter.

Paste in the text I isolated under step 2 on that de novo guide I linked above. (You can copy + paste from here too:)

```
#!/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
```

```
module load fastqc
```

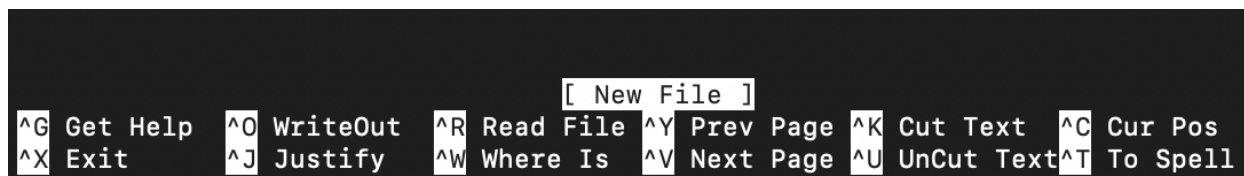
```
readonly SINGULARITY_IMAGE='singularity exec --cleanenv
/n/singularity_images/informatics/trinityrnaseq/trinityrnaseq.v2.12.0.simg'
```

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

ALWAYS HAVE `#!/bin/bash` at the top of your slurm files to submit (any .sh file). This is important formatting information for the cluster.

The %A you see in the SBATCH parameters -o and -e mean that the filename will have your jobid incorporated in the output and error files. This lets you easily identify which error file you want to look at to diagnose what went wrong with which job. (example: FASTQC.6300194.err is the error file for jobid 6300194)

Once you have that pasted in, you can save your file by following the “write out” command at the bottom of the text editor.



^ means holding the control key. So you save your file by holding down Ctrl and hit O.

You also get a prompt to save any edited file when you go to exit, so I just do Ctrl+X and save when prompted.

Whether you save first or upon exiting, use ^X to exit and be returned to your command-line.

Pasted directly from that guide,

The above script uses a command line argument, specified by \$1, which would be the name of the fastq file. Thus, job submission would look something like:

```
$ sbatch myfastqcscript.sh mypurpleunicorn_1.fastq
```

Outputs will include an html format report, and text files summarizing various quality metric information. Note, in the above script and those below, we use the practice of first loading new-modules to create access to the current module set, then purging any modules that might have gotten loaded from your .bash_profile or .bashrc file to avoid any conflicts, then loading any required modules for the current analysis.

The \$1 they mention is at the end of that pasted script:

```
${SLURM_CPUS_ON_NODE} --outdir `pwd` $1
```

Because you're currently in the directory that has both your script (the .sh file) and your .fq files, you can directly enter:

```
sbatch fastqcscript.sh sequence.fq
```

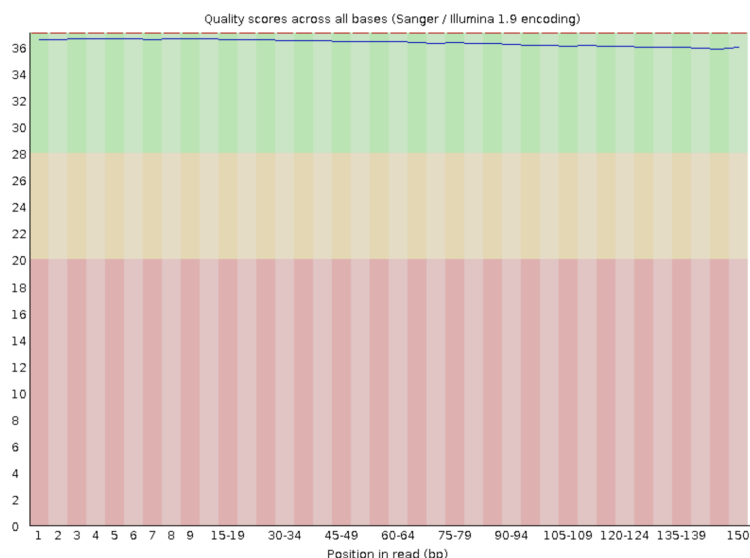
The "sbatch" you type in here is a command saying you're submitting a batch script for the cluster to interpret.

Upon hitting enter, it should tell you it submitted the job, and give you a string of numbers. This string is the jobid; you can use this jobid to get more information about the job as it runs/prepares to run.

What People Look For in FASTQC

In my experience, I only initially look at FASTQC for MAJOR issues. The following screens I check for the following information:

✓ Per base sequence quality

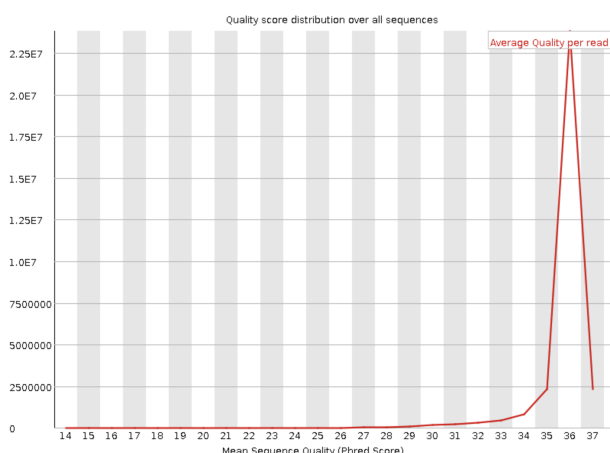


Green is good base quality - high confidence we've noted nucleotides correctly. It's normal for the end on the right to taper a bit lower. But staying all above 36 on the vertical axis is ideal. The quality across position is indicated by the blue line (near the top there).

The vertical axis is mean sequence quality score. I've heard that anything above 20 is "acceptable" because that denotes 99% confidence in the specific

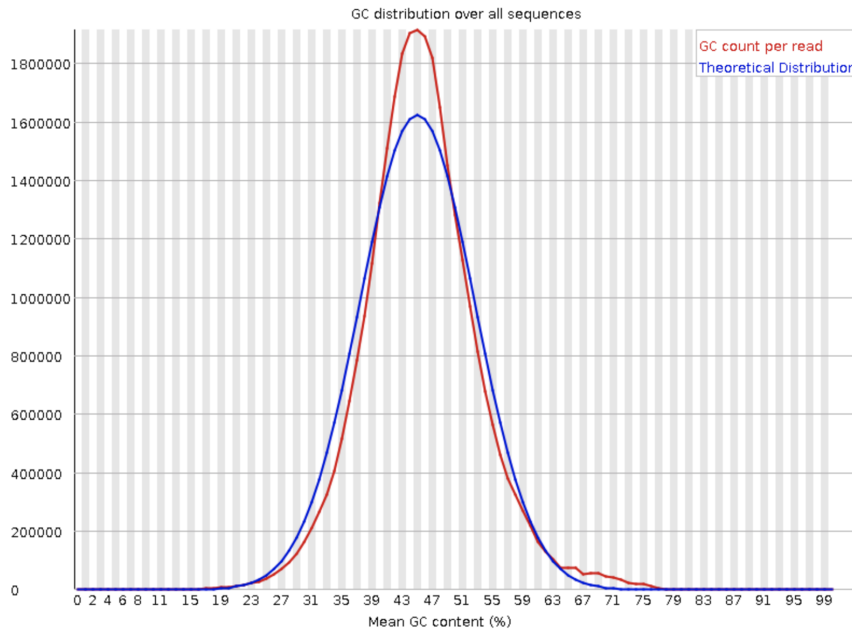
nucleotide for a given position being correct (that's where the red and yellow regions meet), but let's always hope for as high as possible.

✓ Per sequence quality scores



This just gives similar information. A second/lower peak is less desirable.

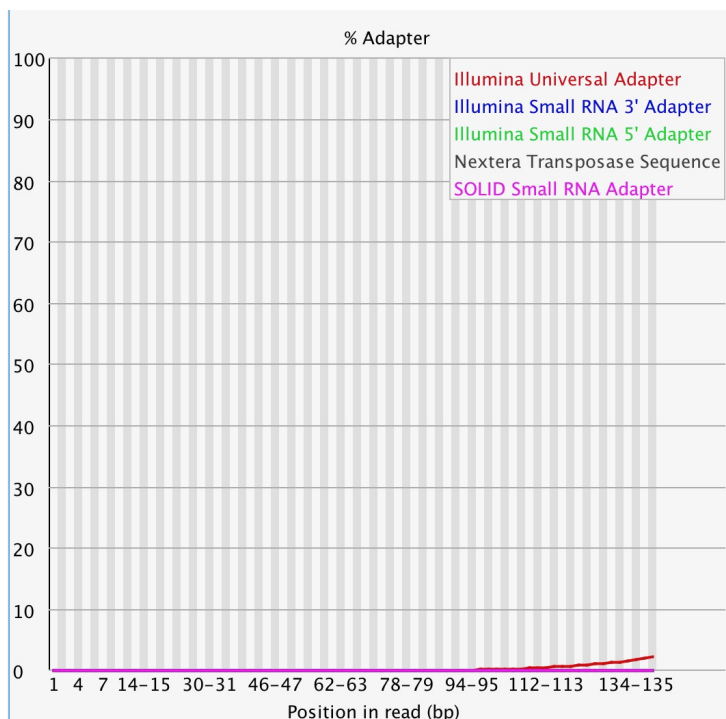
✓ Per sequence GC content



You want to see a strong peak at 40% GC content- that's what we expect in plants! Red is your sample, and blue is theoretical.

The wiggly red tail on the right end of my sample here may be some contamination. Hard to say just from this.

✓ Adapter Content



(I stole this picture from Grace)
A colored line showing above 0 here is indicative of adapter content. Here there's a red tail above 0, meaning there's some Illumina Universal Adapter present in the samples.

In some protocols, you use this identification of the adapter to find its sequence via google, then trim that specific sequence out of each of your samples.

We don't have to worry about that in this protocol.

Removing Rare Kmers Using RCorrector

Rare Kmers are most likely due to sequence errors (or, in my data, they were by chance often fragments of contaminants - you don't have to worry about contaminants in your samples when you align to a reference genome, but you do in *de novo* transcriptomics!).

Anyway, because these are likely errors anyway, it pays off to remove these early, as doing so lets your downstream processes be faster and more efficient.

Rcorrector's a package that lets us do that, and it's already in the cluster.

Using the above "a couple more helpful commands," go ahead and make a new directory in the scratch space that lets you run Rcorrector on copies of your samples, rather than your raw samples.

Then use the "cd" command to enter that new directory.

Then use the "nano \$whateveryouwanttonameyourrcorrectorscript.sh" trick to create what'll be your rcorrector file, and paste in:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 12
#SBATCH -p general,shared
#SBATCH -e rcorrect_%A.e
#SBATCH -o rcorrect_%A.o
#SBATCH -J rcorrect
#SBATCH --mem=24000
#SBATCH --time=36:00:00

module purge
module load Rcorrector/20180919-fasrc01

perl /n/helmod/apps/centos7/Core/Rcorrector/20180919-fasrc01/bin/run_rcorrector.pl
-t 12 -1 $1 -2 $2
```

At the bottom of that script, the \$1 and \$2 there are indicators that when you submit your script for a job, you'll specify your comma-separated sequence files.

For now, use ^X to save and exit out of the now-created script.

I'd use another program (like I type it up in my Notes app on my mac) to type out the command line you'll submit for this. It could be a large body of text if you have a LOT of samples!

If you have a lot of samples, I'd break up the sequences into a few different batches. Here's an example:

I have 24 samples, and I submit a couple at a time through rcorrector as so:

```
sbatch rcorrectorscript.sh NES11_1.fq.gz,NES12_1.fq.gz,NES13_1.fq.gz NES11_2.fq.gz,NES12_2.fq.gz,NES13_2.fq.gz
```

I made it small so it all fit onto one line for this page.

NES11_1.fq.gz and NES11_2.fq.gz are respectively the left (R1) and right (R2) series of my paired-end reads, making together my NES11 sample. (This is “nectary eglandular stage 1 sample 1” in my labeling lingo- don’t worry about the specifics there.)

For my three samples NES11, NES12, and NES13, I separated the R1 and R2 files, which is why it reads NES11_1.fq.gz,NES12_1.fq.gz,NES13_1.fq.gz with NO SPACES separating them - the commas are the separating delimitations. The only space around the samples goes BETWEEN the _1 set and the _2 set.

This produces:

```
sbatch script.sh sample1_1.fq.gz,sample2_1.fq.gz,sample3_1.fq.gz sample1_2.fq.gz,sample2_2.fq.gz,sample3_2.fq.gz
```

When you submit that customized to your script and your samples, it’ll run.

Use sacct to check its status.

When you check your directory after the jobs are finished, using ls, you should see the “corrected” files with “.cor” inserted into your file names, such as my example of:

NES11_1.cor.fq.gz

NES11_2.cor.fq.gz

Etc.

Discarding Unfixable Reads (ones rCorrector couldn’t help)

Paraphrased from the de novo guide:

Because some sequences have lots of Ns or of low complexity, they’re not useful at all, and they just slow things down. Sometimes one read in a read pair is like this, but we should get rid of both reads in that pair, since an unpaired read won’t help us at all either.

This won’t be a huge amount of your data, so don’t sweat getting rid of this stuff.

The guide tells you to clone the github repository of TranscriptomeAssemblyTools. Go to [this link](#), then click on FilterUncorrectablePEfastq.py

This will take you to this page:

harvardinformatics / TranscriptomeAssemblyTools (Public)

Notifications Fork 19 Star 27

<> Code Issues 2 Pull requests 1 Actions Projects Wiki Security Insights

master TranscriptomeAssemblyTools / FilterUncorrectablePEfastq.py / <> Jump to Go to file ...

adamfreedman rm prefix arg Latest commit e2df226 on Feb 19, 2019 History

1 contributor

107 lines (84 sloc) 4.04 KB Raw Blame

```

1  """
2  author: adam h freedman
3  afreedman405 at gmail.com
4  data: Fri Aug 26 10:55:18 EDT 2016
5
6  This script takes as an input Rcorrector error corrected Illumina paired-reads
7  in fastq format and:
8
9  1. Removes any reads that Rcorrector identifies as containing an error,
10 but can't be corrected, typically low complexity sequences. For these,
11 the header contains 'unfixable'.
12
13 2. Strips the 'cor' from headers of reads that Rcorrector fixed, to avoid
14 issues created by certain header formats for downstream tools.
15
16 3. Write a log with counts of (a) read pairs that were removed because one end
17 was unfixable, (b) corrected left and right reads, (c) total number of
18 read pairs containing at least one corrected read.

```

You can click the ellipsis button on the top right, download the file, and move it into your directories on the cluster via FileZilla.

OR you can copy all the blue text in the box there, and paste it into a new cluster file you name FilterUncorrectablePEfastq.py ... Faster and easier.

You now want to make another new script file, like you did for the rcorrector job.

This is what you put in that file (I named mine "trimuncorrectable.sh", for example):

```

#!/bin/bash
#SBATCH -J rmunfix_$1          # Job name
#SBATCH -n 1                  # Use 1 core for the job
#SBATCH -t 03:00:00           # Runtime in HH:MM:SS
#SBATCH -p serial_requeue     # Partition to submit to
#SBATCH --mem=2000             # Memory per node in MB
#SBATCH -o rmunfix_%A.o        # File to which STDOUT will be written
#SBATCH -e rmunfix_%A.e        # File to which STDERR will be written
#SBATCH --mail-type=ALL        # Type of email notification- BEGIN,END,FAIL,ALL
#SBATCH --mail-user=name@harvard.edu # Email to which notifications will be sent

```

```
module purge
```

```
module load python/2.7.8-fasrc01
```

```
python /PATH/TO/FilterUncorrectablePEfastq.py -1 $1 -2 $2 -s $3
```

\$1 and \$2 are the same input prompts as we saw in rcorrector. \$3 here is a new file label you want to add to a log summarizing how many reads were removed. (If anything went bizarrely wrong in your sample, being able to look at this might be useful!)

The example of how I ran this for each of my 24 samples (48 total samples if you consider that they're paired-end reads) is:

```
sbatch trimuncorrectable.sh NES11_1.cor.fq.gz NES11_2.cor.fq.gz sampleremovedNES11
```

This has the call to run a bash script, the bash script itself, the R1 and then R2 halves of my paired-end reads for sample NES11, and then the label "sampleremovedNES11" to the log if I need it later.

You submit this, and it runs. Repeat for each of your samples and let them all run at once.

Trimming Adapters and Low Quality Bases

This is the last main step of this quality work!

Even if your overall sequence quality is very good, there are likely some bases that the sequencer wasn't very confident in labeling one nucleotide or the other.

You likely also have your adapter sequences still captured in some portion of your samples.

We'll use trimgalore to remove those.

Another way to get information from github, where trimgalore is found, is to directly get it from the github repository from within the cluster command-line.

First, navigate to your home directory.

To get there, type in: `cd ~`

This changes your directory to your home (~) state.

Use the `ls` command and see if you have an 'apps' folder there, where you can put your software, etc.

If you don't yet, use `mkdir` to make an "~/apps/" directory.

Then enter `cd ~/apps/` to enter that new directory.

Now paste in:

```
wget https://github.com/FelixKrueger/TrimGalore/archive/0.6.0.zip
unzip 0.6.0.zip
```

This grabs and unzips the program for you.

If you check that directory now with `ls`, you should see this as part of it:

```
[(base) [aburru@holylogin03 apps]$ ls
0.6.0.zip blobtools perl R R_4.0.2 src TrimGalore-0.6.0
```

Trimgalore is over on the right there.

Next, let's go ahead and use the `cp` command to make copies of your corrected reads in a new directory.

First, make that new directory with `mkdir`. I made one in the `holyscratch01` space with:

```
mkdir /n/holyscratch01/davis_lab/aburru/testdata/corrected
```

Now, `cd` to where your `rcorrector`'s output files are (where all the files are that have `.cor` in the file names).

Now, `cp` those files to your new directory.

```
cp *.cor.fq /n/holyscratch01/davis_lab/aburru/testdata/corrected
```

**If you skipped to trimgalore from the top of the document, your file format is probably `.fq.gz`:*

Another example: `cp *.fq.gz /directory/for/data/trimmed`

Now `cd` into that new directory

This part isn't in the guide I've linked, but I found I needed to do it:

Install `cutadapt` via `conda` in the cluster, by running this line:

```
python3 -m pip install --user --upgrade cutadapt
```

Now also specifically put this work onto a specific working path with running:

```
export PATH=/n/home06/$Rcusername/.local/bin:$PATH
```

For me, this is:

```
export PATH=/n/home06/aburru/.local/bin:$PATH
```

You may need to run the installation of `cutadapt` in your home directory (going to `~/apps/` or just into `~` maybe)? But I didn't make note of that.

Now, in your "corrected" directory we just made, use the `nano` trick to make a new file for your `trimgalore` script (mine is "trimgalore.sh") and paste in:

```
#!/bin/bash
#SBATCH -J trimgalore
#SBATCH -n 1 # Use 1 cores for the job
#SBATCH -t 0-6:00 # Runtime in D-HH:MM
#SBATCH -p serial_requeue # Partition to submit to
#SBATCH --mem=3000 # Memory pool for all cores (see also --mem-per-cpu)
#SBATCH -o trimgalore_PE.%A.out # File to which STDOUT will be written
#SBATCH -e trimgalore_PE.%A.err # File to which STDERR will be written
#SBATCH --mail-type=ALL # Type of email notification- BEGIN,END,FAIL,ALL
#SBATCH --mail-user=name@harvard.edu # Email to send notifications to
```

```
module purge
```

```
# $1 = R1 reads
```

```
# $2 = R2 reads
```

```
/your/path/to/trim_galore --paired --retain_unpaired --phred33 --output_dir
trimmed_reads --length 36 -q 5 --stringency 1 -e 0.1 $1 $2
```

“-q 5” in there specifically is to remove bases with a phred value below 5, which should allow for removing REALLY bad quality bases but also not be too stringent such that you lose any valuable information.

Save and exit out of that file.

I actually couldn't figure out how to properly run a job with trimgalore without getting errors. So INSTEAD I ran trimgalore on the foreground (rather than submitting a job) by just using the following script straight in the cluster's command line:

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
/n/holyscratch01/davis_lab/aburru/testdata/corrected/output --length 36 -q 5 --stringency 1 -e 0.1
unfixrm_NES11_1.cor.fq unfixrm_NES11_2.cor.fq
```

Generalized, this is:

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
$newdirectoryyouwanttomakeforoutputtogoto --length 36 -q 5 --stringency 1 -e 0.1$sample1_1.cor.fq
$sample1_2.cor.fq
```

**If you skipped to trimgalore from the top of the document, your file format is probably .fq.gz:*

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
$newdirectoryyouwanttomakeforoutputtogoto --length 36 -q 5 --stringency 1 -e 0.1$.fq.gz $sample1_2.fq.gz
```

And I repeated that for each of my samples, exemplified as below

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
/n/holyscratch01/davis_lab/aburru/testdata/corrected/output --length 36 -q 5 --stringency 1 -e 0.1
unfixrm_NES33_1.cor.fq unfixrm_NES33_2.cor.fq
```

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
/n/holyscratch01/davis_lab/aburru/testdata/corrected/output --length 36 -q 5 --stringency 1 -e 0.1
unfixrm_NGS11_1.cor.fq unfixrm_NGS11_2.cor.fq
```

```
~/apps/TrimGalore-0.6.0/trim_galore --paired --retain_unpaired --phred33 --output_dir
/n/holyscratch01/davis_lab/aburru/testdata/corrected/output --length 36 -q 5 --stringency 1 -e 0.1
unfixrm_NGS12_1.cor.fq unfixrm_NGS12_2.cor.fq
```

Use sacct or email notifications to look at how it runs, check err files if it says FAILED on sacct, and with any luck it'll all work.

The output of these files can be looked at again in fastqc.

```
sbatch trimgalore.sh filename1.fq.gz filename2.fq.gz
```