
labibi Documentation

Release 1.0

C. Titus Brown

March 30, 2016

1	Welcome!	3
1.1	1. Learning goals	3
1.2	2. Safe space and code of conduct	3
1.3	3. Instructor introductions	3
1.4	4. Amazon and cloud computing - why?!	3
1.5	5. Sticky notes and how they work... + Minute Cards	4
2	Non-model organisms and RNAseq	5
2.1	The overall process	6
3	Using Amazon Web Services	7
4	Short read quality and trimming	9
4.1	Prepping the computer	9
4.2	Data source	9
4.3	1. Copying in some data to work with.	9
4.4	1. Copying data into a working location	10
4.5	2. FastQC	10
4.6	3. Trimmomatic	11
4.7	4. FastQC again	12
4.8	5. Trim the rest of the sequences	12
4.9	6. Interleave the sequences	14
5	Running digital normalization	15
6	Running the actual assembly	17
7	Assembly statistics and evaluation	19
7.1	Applying transrate	19
7.2	Evaluating read mapping	20
7.3	Using transrate to compare two transcriptomes	21
7.4	Merging two (or more) assemblies	22
8	BLASTing your assembled data	23
9	Annotation of denovo transcriptome	25
9.1	Identify the Gene/Transcript relationships:	25
9.2	Generate the longest-ORF peptide candidates from the Trinity Assembly:	25
9.3	Capturing BLAST Homologies	26

9.4	Characterization of functional annotation features	26
9.5	Integration of all annotations into one database	27
9.6	Output an Annotation Report	28
10	Quantification and Differential Expression	29
10.1	Download Express	29
10.2	Align Reads with Bowtie	29
10.3	Quantify Expression using eXpress	30
10.4	Differential Expression	30
11	Remapping your reads to your assembled transcriptome	33
12	Miscellaneous advice	35
12.1	Sequencing depth and number of samples	35
12.2	Downloading your data	35
12.3	Developing your own pipeline	36
13	More resources	37
13.1	Informational resources	37
13.2	Places to share data, scripts, and results files	37
14	Miscellaneous questions	39
15	Tips and Tricks for working with Remote Computers	41
15.1	Use screen to run things that take a long time.	41
15.2	Use CyberDuck to transfer files	41
15.3	Subsetting data	41
15.4	Running full analyses on Amazon Web Services	42
16	Technical information	43

Part of this workshop was given on Mar 30th, 2016, by C. Titus Brown. See [the workshop organization page](#) for more information, or [contact Titus directly](#).

We have an [EtherPad](#) for sharing text and asking questions

Tutorials:

Welcome!

1.1 1. Learning goals

For you:

- get a first (or second) look at tools;
- gain some experience in the basic command line;
- get 80% of way to a complete analysis of some data;

For us:

- what are the on campus needs? who are the on-campus people?

1.2 2. Safe space and code of conduct

This is intended to be a safe and friendly place for learning!

Please see the Software Carpentry workshop Code of Conduct: <http://software-carpentry.org/conduct.html>

In particular, please ask questions, because I guarantee you that your question will help others!

1.3 3. Instructor introductions

Titus Brown - prof here at UC Davis in the School of Vet Med.

1.4 4. Amazon and cloud computing - why?!

- simplifies software installation;
- can be used for bigger analyses quite easily;
- good for “burst” capacity (just got a data set!)
- accessible everywhere;
- they give us \$100 gift certificates...

1.5 5. Sticky notes and how they work... + Minute Cards

Basic rules:

- no sticky note - “working on it”
- green sticky note - “all is well”
- red sticky note - “need help!”

Place the sticky notes where we can see them from the back of the room – e.g. on the back of your laptop.

At the end of each session (coffee break, lunch, end of day) please write down on an index card **one thing you learned** and **one thing you’re still confused about**.

—

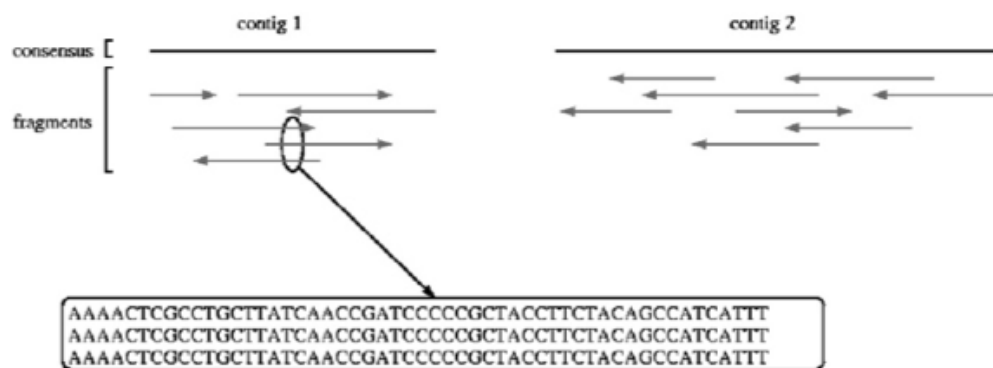
Next: [Non-model organisms and RNAseq](#)

Non-model organisms and RNAseq

With non-model systems, where there is neither a good genome nor a lot of mRNAseq data, you have to build your own transcriptome from scratch – so-called “de novo transcriptome assembly”. There are a few programs to do this – most notably Trinity and Oases – and [we have found little difference](#).

Shotgun sequencing & assembly

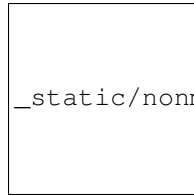
Randomly fragment & sequence from DNA;
reassemble computationally.



UMD assembly primer (cbcb.umd.edu)

The main problem you’ll run into with non-model mRNAseq is that the output is fairly noisy with respect to splice variants. Our experience has been that many of these splice variants are probably “real” – in the sense of actually present – but may be biological “noise”, in the sense that they are not actually functional (See [this excellent paper by Pickrell and Pritchard making the case](#)). Regardless, there’s little that you can do about this, although we will talk about it a bit on the second day.

2.1 The overall process



_static/nonmodel-rnaseq-pipeline.png

- Copy over your RNAseq data (from two or more samples);
- Trim primers and junk from sequence ([Short read quality and trimming](#))
- Do abundance normalization ([Running digital normalization](#))
- Assemble everything together ([Running the actual assembly](#))

This gives you an assembled transcriptome, consisting of many transcripts and transcript families.

At this point you can do one or more of the following:

- Annotate your transcripts ([Annotation of denovo transcriptome](#))
- Quantify your transcripts and examine differential expression ([Quantification and Differential Expression](#))
- BLAST your transcripts individually ([BLASTing your assembled data](#))

Next: [Using Amazon Web Services](#)

Using Amazon Web Services

We're going to be using computers rented from Amazon Web Services today. Please see [the February workshop](#) for instructions - get to the part where you're logged into your Amazon instance.

Note: please use an m3.xlarge, rather than an m4.xlarge.

Next: [Short read quality and trimming](#)

Short read quality and trimming

Note: There's a more thorough discussion of trimming in our 2016 short-read trimming workshop, [here](#).

OK, you should now be logged into your Amazon computer! How exciting!

4.1 Prepping the computer

Before we do anything else, we need to set up a place to work and install a few things.

First, let's set up a place to work:

```
sudo chmod a+rwxt /mnt
```

This makes '/mnt' a place where we can put data and working files.

Next, let's install a few things:

```
sudo apt-get update
sudo apt-get install -y trimmomatic fastqc python-pip python-dev
```

These are the Trimmomatic and FastQC programs, which we'll use below, along with some software prerequisites that we'll need for other things below.

4.2 Data source

We're going to be using a subset of data from [Tulin et al., 2013](#), a paper looking at early transcription in the organism *Nematostella vectensis*, the sea anemone.

4.3 1. Copying in some data to work with.

We've loaded subsets of the data onto an Amazon location for you, to make everything faster for today's work. We're going to put the files on your computer locally under the directory /mnt/data:

```
mkdir /mnt/data
```

Next, let's grab part of the data set:

```
cd /mnt/data
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
```

Now if you type:

```
ls -l
```

you should see something like:

```
-r--r--r-- 1 ubuntu ubuntu 7874107 Dec 14 2013 0Hour_ATCACG_L002_R1_001.extract.fastq.gz
-r--r--r-- 1 ubuntu ubuntu 7972058 Dec 14 2013 0Hour_ATCACG_L002_R1_002.extract.fastq.gz
...
```

These are subsets of the original data, where we selected for reads that belong to a few particular transcripts.

One problem with these files is that they are writeable - by default, UNIX makes things writeable by the file owner. Let's fix that before we go on any further:

```
chmod u-w *
```

We'll talk about what these files are below.

4.4 1. Copying data into a working location

First, make a working directory; this will be a place where you can futz around with a copy of the data without messing up your primary data:

```
mkdir /mnt/work
cd /mnt/work
```

Now, make a "virtual copy" of the data in your working directory by linking it in –

```
ln -fs /mnt/data/* .
```

These are FASTQ files – let's take a look at them:

```
less 0Hour_ATCACG_L002_R1_001.extract.fastq.gz
```

(use the spacebar to scroll down, and type 'q' to exit 'less')

Question:

- why do the files have DNA in the name?
- why are there R1 and R2 in the file names?
- why don't we combine all the files?

Links:

- [FASTQ Format](#)

4.5 2. FastQC

We're going to use [FastQC](#) to summarize the data. We already installed 'fastqc' on our computer - that's what the 'apt-get install' did, above.

Now, run FastQC on two files:

```
fastqc 0Hour_ATCACG_L002_R1_001.extract.fastq.gz
fastqc 0Hour_ATCACG_L002_R2_001.extract.fastq.gz
```

Now type 'ls':

```
ls -d *fastqc*
```

to list the files, and you should see:

```
0Hour_ATCACG_L002_R1_001.extract_fastqc
0Hour_ATCACG_L002_R1_001.extract_fastqc.zip
0Hour_ATCACG_L002_R2_001.extract_fastqc
0Hour_ATCACG_L002_R2_001.extract_fastqc.zip
```

We are *not* going to show you how to look at these files right now - you need to copy them to your local computer to do that. We'll show you that tomorrow. But! we can show you what they look like, because I've made copies of them for you:

- [0Hour_ATCACG_L002_R1_001.extract_fastqc/fastqc_report.html](#)
- [0Hour_ATCACG_L002_R2_001.extract_fastqc/fastqc_report.html](#)

Questions:

- What should you pay attention to in the FastQC report?
- Which is “better”, R1 or R2? And why?

Links:

- [FastQC](#)
- [FastQC tutorial video](#)

4.6 3. Trimmomatic

Now we're going to do some trimming! We'll be using [Trimmomatic](#), which (as with fastqc) we've already installed via apt-get.

The first thing we'll need are the adapters to trim off:

```
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-semi-2015-03-04/TruSeq2-PE.fa
```

Now, to run Trimmomatic:

```
TrimmomaticPE 0Hour_ATCACG_L002_R1_001.extract.fastq.gz \
               0Hour_ATCACG_L002_R2_001.extract.fastq.gz \
0Hour_ATCACG_L002_R1_001.qc.fq.gz s1_se \
0Hour_ATCACG_L002_R2_001.qc.fq.gz s2_se \
ILLUMINACLIP:TruSeq2-PE.fa:2:40:15 \
LEADING:2 TRAILING:2 \
SLIDINGWINDOW:4:2 \
MINLEN:25
```

You should see output that looks like this:

```
...
Quality encoding detected as phred33
Input Read Pairs: 140557 Both Surviving: 138775 (98.73%) Forward Only Surviving: 1776 (1.26%) Reverse
TrimmomaticPE: Completed successfully ...
```

Questions:

- How do you figure out what the parameters mean?
- How do you figure out what parameters to use?
- What adapters do you use?
- What version of Trimmomatic are we using here? (And FastQC?)
- Do you think parameters are different for RNAseq and genomic data sets?
- What's with these annoyingly long and complicated filenames?
- why are we running R1 and R2 together?

For a discussion of optimal RNAseq trimming strategies, see [MacManes, 2014](#).

Links:

- [Trimmomatic](#)

4.7 4. FastQC again

Run FastQC again on the trimmed files:

```
fastqc 0Hour_ATCACG_L002_R1_001.qc.fq.gz
fastqc 0Hour_ATCACG_L002_R2_001.qc.fq.gz
```

And now view my copies of these files:

- [0Hour_ATCACG_L002_R1_001.qc.fq_fastqc/fastqc_report.html](#)
- [0Hour_ATCACG_L002_R2_001.qc.fq_fastqc/fastqc_report.html](#)

Let's take a look at the output files:

```
less 0Hour_ATCACG_L002_R1_001.qc.fq.gz
```

(again, use spacebar to scroll, 'q' to exit less).

Questions:

- is the quality trimmed data “better” than before?
- Does it matter that you still have adapters!?

4.8 5. Trim the rest of the sequences

First download the rest of the data:

```
cd /mnt/data
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/0Hour_ATCACG_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
```



```
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
curl -O -L http://dib-training.ucdavis.edu.s3.amazonaws.com/mRNAseq-non-2015-05-04/6Hour_CGATGT_L002_
```

And link it in:

```
cd /mnt/work
ln -fs /mnt/data/*.fastq.gz .
```

Now we have a lot of files – and we really don’t want to trim each and every one of them by typing in a command for each pair! Here we’ll make use of a great feature of the UNIX command line – the ability to automate such tasks.

Here’s a for loop that you can run - we’ll walk through what it does while it’s running:

```
rm -f orphans.fq

for filename in *_R1_*.extract.fastq.gz
do
    # first, make the base by removing .extract.fastq.gz
    base=$(basename $filename .extract.fastq.gz)
    echo $base

    # now, construct the R2 filename by replacing R1 with R2
    baseR2=${base/_R1/_R2_}
    echo $baseR2

    # finally, run Trimmomatic
    TrimmomaticPE ${base}.extract.fastq.gz ${baseR2}.extract.fastq.gz \
        ${base}.qc.fq.gz s1_se \
        ${baseR2}.qc.fq.gz s2_se \
        ILLUMINACLIP:TruSeq2-PE.fa:2:40:15 \
        LEADING:2 TRAILING:2 \
        SLIDINGWINDOW:4:2 \
        MINLEN:25

    # save the orphans
    cat s1_se s2_se >> orphans.fq
done
```

Things to mention –

- # are comments;
- anywhere you see a ‘\$’ is replaced by the value of the variable after it, so e.g. \$filename is replaced by each of the files matching `_R1_.extract.fastq.gz`, once for each time through the loop;
- we have to do complicated things to the filenames to get this to work, which is what the `${base/_R1/_R2_}` stuff is about.
- what’s with ‘orphans.fq’??

Questions:

- **how do you figure out if it’s working?**
 - copy/paste it from Word

- put in lots of echo
- edit one line at a time
- how on earth do you figure out how to do this?!

4.9 6. Interleave the sequences

Next, we need to take these R1 and R2 sequences and convert them into interleaved form ,for the next step. To do this, we'll use scripts from the [khmer package](#), which we need to install:

```
sudo pip install -U setuptools
sudo pip install khmer==2.0
```

Now let's use a for loop again - you might notice this is only a minor modification of the previous for loop...

```
for filename in *_R1_*.qc.fq.gz
do
    # first, make the base by removing .extract.fastq.gz
    base=$(basename $filename .qc.fq.gz)
    echo $base

    # now, construct the R2 filename by replacing R1 with R2
    baseR2=${base/_R1/_R2_}
    echo $baseR2

    # construct the output filename
    output=${base/_R1_/}.pe.qc.fq.gz

    interleave-reads.py ${base}.qc.fq.gz ${baseR2}.qc.fq.gz | \
        gzip > $output
done

gzip orphans.fq
```

Next: [Running digital normalization](#)

Running digital normalization

Next, we’re going to apply abundance normalization to the data – known as “digital normalization”, this approach was developed by our lab to make it possible to assemble large data sets more quickly and easily. You can read more about it in [Brown et al., 2012](#), and also see some of its affects on transcriptome assembly in [Lowe et al., 2014](#).

Digital normalization works by eliminating high abundance reads that are unnecessary for assembly.

First, we’ll run it on the interleaved files we generated in the previous section:

```
cd /mnt/work
normalize-by-median.py -k 20 -C 20 -N 4 -x 2e8 -s normC20k20.ct *.pe.qc.fq.gz orphans.fq.gz
```

(These parameters should work for essentially all mRNAseq data sets; see [the khmer documentation](#) for more information.)

Do k-mer abundance trimming on the reads, which will eliminate the majority of the errors (thus further decreasing the memory requirements) –:

```
filter-abund.py -V normC20k20.ct *.keep
```

See our paper [Zhang et al., 2014](#) <<http://www.ncbi.nlm.nih.gov/pubmed/25062443>> ___, Table 3, for more information on k-mer trimming effects.

Now, take all of the paired-end files and split them into paired and orphaned reads:

```
for filename in *.pe.*.keep.abundfilt
do
    extract-paired-reads.py $filename
done
```

Put all the orphaned reads in one place:

```
cat *.se orphans.fq.gz.keep.abundfilt | gzip > orphans.dn.fq.gz
```

And now rename the paired-end files to something nice:

```
for filename in *.pe.qc.fq.gz.keep.abundfilt.pe
do
    base=$(basename $filename .pe.qc.fq.gz.keep.abundfilt.pe)
    output=${base}.dn.fq.gz
    gzip -c $filename > $output
done
```

Now, if you type:

```
ls *.dn.fq.gz
```

you'll see all of the files that you need to move on to the next step –

```
0Hour_ATCACG_L002001.dn.fq.gz  6Hour_CGATGT_L002002.dn.fq.gz
0Hour_ATCACG_L002002.dn.fq.gz  6Hour_CGATGT_L002003.dn.fq.gz
0Hour_ATCACG_L002003.dn.fq.gz  6Hour_CGATGT_L002004.dn.fq.gz
0Hour_ATCACG_L002004.dn.fq.gz  6Hour_CGATGT_L002005.dn.fq.gz
0Hour_ATCACG_L002005.dn.fq.gz  orphans.dn.fq.gz
6Hour_CGATGT_L002001.dn.fq.gz
```

Let's remove some of the detritus before moving on:

```
rm *.pe *.se *.abundfilt *.keep
rm normC20k20.ct
```

Next: [Running the actual assembly](#)

Running the actual assembly

Now we'll assemble all of these reads into a transcriptome, using [the Trinity de novo transcriptome assembler](#).

First, install some prerequisites for Trinity:

```
sudo apt-get -y install bowtie samtools zlib1g-dev ncurses-dev
```

Next, install Trinity v2.2.0:

```
cd
curl -L https://github.com/trinityrnaseq/trinityrnaseq/archive/v2.2.0.tar.gz > trinity.tar.gz
tar xzf trinity.tar.gz
mv trinityrnaseq* trinity/

cd trinity
make
```

Go into the work directory, and prepare the data:

```
cd /mnt/work
for i in *.dn.fq.gz
do
    split-paired-reads.py $i
done

cat *.1 > left.fq
cat *.2 > right.fq
```

Now, run the Trinity assembler:

```
~/trinity/Trinity --left left.fq --right right.fq --seqType fq --max_memory 10G --bypass_java_version
```

This will give you an output file `trinity_out_dir/Trinity.fasta`.

Let's copy that to a safe place, where we'll work with it moving forward:

```
cp trinity_out_dir/Trinity.fasta rna-assembly.fa
```

Next: [Assembly statistics and evaluation](#)

Assembly statistics and evaluation

So, we now have an assembly of our reads in `rna-assembly.fa`. Let's take a look at this file –

```
head rna-assembly.fa
```

This is a FASTA file with complex (and not, on the face of it, very informative!) sequence headers, and a bunch of sequences. There are three things you might want to do with this assembly - check its quality, annotate it, and search it. Below we're going to check its quality; other workshops do (will) cover annotation and search.

7.1 Applying transrate

`transrate` is a program for assessing RNAseq assemblies that will give you a bunch of assembly statistics, along with a few other outputs.

First, let's download and install it:

```
cd
curl -O -L https://bintray.com/artifact/download/blahah/generic/transrate-1.0.2-linux-x86_64.tar.gz
tar xzf transrate-1.0.2-linux-x86_64.tar.gz
export PATH=~/.transrate-1.0.2-linux-x86_64:$PATH
```

Now run `transrate` on the assembly to get some preliminary stats:

```
transrate --assembly=/mnt/work/rna-assembly.fa --output=/mnt/work/stats
```

This should give you output like this:

```
[ INFO] 2016-03-30 13:38:11 : n seqs                62
[ INFO] 2016-03-30 13:38:11 : smallest              206
[ INFO] 2016-03-30 13:38:11 : largest              4441
[ INFO] 2016-03-30 13:38:11 : n bases              81132
[ INFO] 2016-03-30 13:38:11 : mean len             1308.58
[ INFO] 2016-03-30 13:38:11 : n under 200           0
[ INFO] 2016-03-30 13:38:11 : n over 1k             37
[ INFO] 2016-03-30 13:38:11 : n over 10k            0
[ INFO] 2016-03-30 13:38:11 : n with orf            41
[ INFO] 2016-03-30 13:38:11 : mean orf percent      64.76
[ INFO] 2016-03-30 13:38:11 : n90                  841
[ INFO] 2016-03-30 13:38:11 : n70                 1563
[ INFO] 2016-03-30 13:38:11 : n50                 1606
[ INFO] 2016-03-30 13:38:11 : n30                 2070
[ INFO] 2016-03-30 13:38:11 : n10                 3417
[ INFO] 2016-03-30 13:38:11 : gc                   0.46
```

```
[ INFO] 2016-03-30 13:38:11 : gc skew          -0.04
[ INFO] 2016-03-30 13:38:11 : at skew          -0.05
[ INFO] 2016-03-30 13:38:11 : cpG ratio        1.88
[ INFO] 2016-03-30 13:38:11 : bases n         0
[ INFO] 2016-03-30 13:38:11 : proportion n    0.0
[ INFO] 2016-03-30 13:38:11 : linguistic complexity 0.23
```

...which is pretty useful basic stats.

I'd suggest paying attention to:






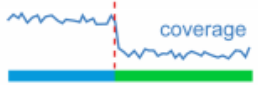
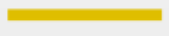
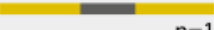













- n seqs
- largest
- mean orf percent

and more or less ignoring the rest ;).

Note: don't use n50 to characterize your transcriptome, as with transcripts you are not necessarily aiming for the longest contigs, and isoforms will mess up your statistics in any case.

7.2 Evaluating read mapping

You can also use transrate to assess read mapping; this will use read evidence to detect many kinds of errors.

Error type	Transcripts	Assembly	Read evidence
Family collapse	 geneAA geneAB geneAC n=3	 n=1	 bases in reads agreement
Chimerism	 geneC geneB n=2	 n=1	 coverage
Unsupported insertion	 n=1	 n=1	 no reads align to insertion
Incompleteness	 n=1	 n=1	 read pairs align off end of contig
Fragmentation	 n=1	 n=4	 bridging read pairs
Local misassembly	 n=1	 n=1	 read pairs in wrong orientation
Redundancy	 n=1	 n=3	 all reads assign to best contig

To do this, you have to supply transrate with the reads:


```
transrate --assembly=/mnt/work/rna-assembly.fa --left=/mnt/work/left.fq --right=/mnt/work/right.fq --
```

The relevant output is here:

```
[ INFO] 2016-03-30 15:38:55 : Read mapping metrics:
[ INFO] 2016-03-30 15:38:55 : -----
[ INFO] 2016-03-30 15:38:55 : fragments                      57178
[ INFO] 2016-03-30 15:38:55 : fragments mapped              50925
[ INFO] 2016-03-30 15:38:55 : p fragments mapped              0.89
[ INFO] 2016-03-30 15:38:55 : good mappings                  8297
[ INFO] 2016-03-30 15:38:55 : p good mapping                  0.15
[ INFO] 2016-03-30 15:38:55 : bad mappings                    42628
[ INFO] 2016-03-30 15:38:55 : potential bridges                22
[ INFO] 2016-03-30 15:38:55 : bases uncovered                 11877
[ INFO] 2016-03-30 15:38:55 : p bases uncovered               0.15
[ INFO] 2016-03-30 15:38:55 : contigs uncovbase                48
[ INFO] 2016-03-30 15:38:55 : p contigs uncovbase              0.77
[ INFO] 2016-03-30 15:38:55 : contigs uncovered                13
[ INFO] 2016-03-30 15:38:55 : p contigs uncovered              0.21
[ INFO] 2016-03-30 15:38:55 : contigs lowcovered               25
[ INFO] 2016-03-30 15:38:55 : p contigs lowcovered             0.4
[ INFO] 2016-03-30 15:38:55 : contigs segmented               14
[ INFO] 2016-03-30 15:38:55 : p contigs segmented              0.23
[ INFO] 2016-03-30 15:38:55 : Read metrics done in 7 seconds
[ INFO] 2016-03-30 15:38:55 : No reference provided, skipping comparative diagnostics
[ INFO] 2016-03-30 13:43:11 : -----
[ INFO] 2016-03-30 13:43:11 : TRANSRATE OPTIMAL SCORE      0.036
[ INFO] 2016-03-30 13:43:11 : TRANSRATE OPTIMAL CUTOFF      0.1589
[ INFO] 2016-03-30 13:43:11 : good contigs                    27
[ INFO] 2016-03-30 13:43:11 : p good contigs                   0.44
```

7.3 Using transrate to compare two transcriptomes

transrate can also compare an assembly to a “reference”. One nice thing about this is that you can compare two assemblies...

First, install the necessary software:

```
transrate --install-deps ref
```

Second, download a different assembly – this is done with the same starting reads, but without using digital normalization:

```
curl -O -L https://github.com/ngs-docs/2016-mar-nonmodel/raw/master/files/rna-assembly-nodn.fa.gz
gunzip rna-assembly-nodn.fa.gz
```

Compare in both directions:

```
transrate --assembly=/mnt/work/rna-assembly.fa --reference=/mnt/work/rna-assembly-nodn.fa --output=/mnt/work/
```

and

```
transrate --reference=/mnt/work/rna-assembly.fa --assembly=/mnt/work/rna-assembly-nodn.fa --output=/mnt/work/
```

In this case you can see that our assembly “covers” more of the other assembly than the other assembly does ours.

7.4 Merging two (or more) assemblies

Finally, you can also use transrate to merge contigs from multiple assemblies, if you’ve used read mapping –

```
transrate --assembly=/mnt/work/rna-assembly.fa \  
  --merge-assemblies=/mnt/work/rna-assembly-nodn.fa \  
  --left=/mnt/work/left.fq --right=/mnt/work/right.fq \  
  --output=/mnt/work/transrate-merge
```

and at the end you’ll see you have more “good” contigs –:

```
[ INFO] 2016-03-30 15:50:54 : p good contigs          0.52
```

Back to index: [2016 / Mar / mRNAseq on non-model organisms](#)

BLASTing your assembled data

First, install a few prerequisites:

```
sudo apt-get -y install lighttpd blast2 git-core zlib1g-dev
```

Next, grab things needed for the BLAST server:

```
sudo pip install pygr whoosh Pillow Jinja2 \
    git+https://github.com/ctb/pygr-draw.git screed
sudo ln -s /usr/bin/blastall /usr/local/bin/
```

Install the BLAST server and configure it:

```
cd
git clone https://github.com/ctb/blastkit.git -b 2015-may-nonmodel
sudo ./blastkit/configure-lighttpd.sh

cd blastkit/www
sudo ln -fs $PWD /var/www/blastkit

mkdir files
chmod a+rxwt files
chmod +x /home/ubuntu

cd /home/ubuntu/blastkit
python ./check.py
```

Now, copy in your newly created transcriptome:

```
cd /mnt/work
gunzip -c trinity-nematostella-raw.renamed.fasta.gz > /home/ubuntu/blastkit/db/db.fa

cd /home/ubuntu/blastkit
formatdb -i db/db.fa -o T -p F
python index-db.py db/db.fa
```

You can now access your BLAST server at <http://<amazon machine name>/blastkit/>.

Note that you will need to enable HTTP access on your Amazon firewall settings; see [../amazon/enable-http](#).

If you want to use something that will get you results with the test data set, try BLASTing [zebrafish lethal giant larvae homolog](#).

Annotation of denovo transcriptome

9.1 Identify the Gene/Transcript relationships:

we can generate this file like so:

```
cd /mnt/work
~/trinity/util/support_scripts/get_Trinity_gene_to_trans_map.pl trinity_out_dir/Trinity.fasta > Trin
```

Let's have a look on the map:

```
less Trinity.fasta.gene_trans_map
```

Components, genes and isoforms:

- The different (i's) that correspond to the same (g) represent isoforms
- The different (g's) could represent different genes (or parts of genes)
- The component (TRlc) often contain related genes (paralogs or gene fragments).

Check the [Trinityseq forum](#) for more details

9.2 Generate the longest-ORF peptide candidates from the Trinity Assembly:

We need to install Transdecoder to do this job:

```
cd
sudo cpan URI::Escape
```

Note: type yes for all interactive questions

```
curl -L https://github.com/TransDecoder/TransDecoder/archive/2.0.1.tar.gz > transdecoder.tar.gz
tar xzf transdecoder.tar.gz
mv TransDecoder* TransDecoder
cd TransDecoder
make
```

Now we can run the Transdecoder software to identify the longest-ORF peptide:

```
cd /mnt/work
~/TransDecoder/TransDecoder.LongOrfs -t trinity_out_dir/Trinity.fasta
```

Check the Transdecoder output:

```
less Trinity.fasta.transdecoder_dir/longest_orfs.pep
```

9.3 Capturing BLAST Homologies

Install BLAST+ (<http://www.ncbi.nlm.nih.gov/books/NBK52640/>):

```
sudo apt-get install -y ncbi-blast+
```

Get the required sequence databases and prepare local blast databases

1. SwissProt database: The UniProt Knowledgebase which include the Manually annotated proteins:

```
cd /mnt/work
wget ftp://ftp.broadinstitute.org/pub/Trinity/Trinotate_v2.0_RESOURCES/uniprot_sprot.trinotate_v
mv uniprot_sprot.trinotate_v2.0.pep.gz uniprot_sprot.trinotate.pep.gz
gunzip uniprot_sprot.trinotate.pep.gz
makeblastdb -in uniprot_sprot.trinotate.pep -dbtype prot
```

Run blast to find homologies.

- (a) search Trinity transcripts:

```
blastx -query trinity_out_dir/Trinity.fasta -db uniprot_sprot.trinotate.pep -num_threads 4 -
```

- (b) search Transdecoder-predicted proteins:

```
blastp -query Trinity.fasta.transdecoder_dir/longest_orfs.pep -db uniprot_sprot.trinotate.pep
```

2. Optional: Uniref90 which provides clustered sets of protein sequences in a way such that each cluster is composed of sequences that have at least 90% sequence identity to, and 80% overlap with, the longest sequence:

```
wget ftp://ftp.broadinstitute.org/pub/Trinity/Trinotate_v2.0_RESOURCES/uniprot_uniref90.trinotate
mv uniprot_uniref90.trinotate_v2.0.pep.gz uniprot_uniref90.trinotate.pep.gz
gunzip uniprot_uniref90.trinotate.pep.gz
makeblastdb -in uniprot_uniref90.trinotate.pep -dbtype prot
```

perform similar searches using uniref90 as the target database, rename output files accordingly:

```
blastx -query trinity_out_dir/Trinity.fasta -db uniprot_uniref90.trinotate.pep -num_threads 4 -m
blastp -query Trinity.fasta.transdecoder_dir/longest_orfs.pep -db uniprot_uniref90.trinotate.pep
```

I have ran them overnight already. You can download these files to save time:

```
wget https://github.com/ngs-docs/2015-may-nonmodel/blob/master/_static/uniref90.blastp.outfmt6
wget https://github.com/ngs-docs/2015-may-nonmodel/blob/master/_static/uniref90.blastx.outfmt6
```

9.4 Characterization of functional annotation features

1. identify protein domains: we need to install HMMER and download the Pfam domains database:

```
sudo apt-get install -y hmmer
```

Then we can run hmmer to identify the protein domains:

```
cd /mnt/work
wget ftp://ftp.broadinstitute.org/pub/Trinity/Trinotate_v2.0_RESOURCES/Pfam-A.hmm.gz
gunzip Pfam-A.hmm.gz
hmmcompress Pfam-A.hmm
hmmsearch --cpu 4 --domtblout TrinotatePFAM.out Pfam-A.hmm Trinity.fasta.transdecoder_dir/longest_orfs
```

2. We can predict other features like

- signal peptides: using signalP
- transmembrane regions: using tmHMM
- rRNA transcripts: using RNAMMER

9.5 Integration of all annotations into one database

install Trinotate:

```
cd
curl -L https://github.com/Trinotate/Trinotate/archive/v2.0.2.tar.gz > trinotate.tar.gz
tar xzf trinotate.tar.gz
mv Trinotate* Trinotate
```

install `sqlite`

```
sudo apt-get install sqlite3
```

We need also the DBI perl package:

```
sudo cpan DBI
sudo cpan DBD::SQLite
```

Retrieve the Trinotate Pre-generated Resource SQLite database A pregenerated sqlite database that contains Uniprot (swissprot and uniref90)-related annotation information is available from the Trinity ftp site:

```
cd /mnt/work
wget "ftp://ftp.broadinstitute.org/pub/Trinity/Trinotate_v2.0_RESOURCES/Trinotate.sprot_uniref90.20150428.sqlite.gz"
gunzip Trinotate.sqlite.gz
```

Load transcripts and coding regions. We have three data types:

1. Transcript sequences (de novo assembled transcripts or reference transcripts)
2. Protein sequences (currently as defined by TransDecoder)
3. Gene/Transcript relationships

```
~/Trinotate/Trinotate Trinotate.sqlite init --gene_trans_map Trinity.fasta.gene_trans_map --transcript_map Trinity.fasta.transdecoder_dir.transcript_map
```

Loading BLAST homologies:

```
~/Trinotate/Trinotate Trinotate.sqlite LOAD_swissprot_blastp blastp.outfmt6
~/Trinotate/Trinotate Trinotate.sqlite LOAD_swissprot_blastx blastx.outfmt6
```

Optional: load Uniref90 blast hits:

```
~/Trinotate/Trinotate Trinotate.sqlite LOAD_trembl_blastp uniref90.blastp.outfmt6
~/Trinotate/Trinotate Trinotate.sqlite LOAD_trembl_blastx uniref90.blastx.outfmt6
```

Optional: Loading functional annotation features:

```
~/Trinotate/Trinotate Trinotate.sqlite LOAD_pfam TrinotatePFAM.out
```

9.6 Output an Annotation Report

```
~/Trinotate/Trinotate Trinotate.sqlite report -E 0.0001 > trinotate_annotation_report.xls
```

There are 2 arguments that we can use to control the accuracy of annotation

-E <float> : maximum E-value for reporting best blast hit and associated annotations.

-pfam_cutoff <string>

1. 'DNC' : domain noise cutoff (default)
2. 'DGC' : domain gathering cutoff
3. 'DTC' : domain trusted cutoff
4. 'SNC' : sequence noise cutoff
5. 'SGC' : sequence gathering cutoff
6. 'STC' : sequence trusted cutoff

let us see the output. Open a new shell:

```
scp -i YOUR_SECURITY_KEY.pem ubuntu@YOUR_AMAZONE_INSTANCE_ADDRESS:/mnt/work/trinotate_annotation_report.xls
```


[illegible]

```
cd /mnt/work
ln -fs /mnt/data/*.fastq.gz .
```

```
cd
curl -L http://bio.math.berkeley.edu/eXpress/downloads/express-1.5.1/express-1.5.1-linux
tar xzf express.tar.gz
```

29

```
cd /mnt/work
gunzip -c trinity-nematostella-raw.renamed.fasta.gz > trinity-nematostella-raw.renamed.fasta
bowtie-build --offrate 1 trinity-nematostella-raw.renamed.fasta trinity-nematostella-raw.renamed
```

Using the index we built, we'll align the reads from a few of our samples back to our assembly:

```
bowtie -aS -X 800 --offrate 1 trinity-nematostella-raw.renamed \
  -1 <(zcat 0Hour_ATCACG_L002_R1_001.extract.fastq.gz) \
  -2 <(zcat 0Hour_ATCACG_L002_R2_001.extract.fastq.gz) \
  > 0Hour_ATCACG_L002_001.extract.sam

bowtie -aS -X 800 --offrate 1 trinity-nematostella-raw.renamed -1 <(zcat 0Hour_ATCACG_L002_R1_002.extract.fastq.gz) \
  -2 <(zcat 0Hour_ATCACG_L002_R2_002.extract.fastq.gz) \
  > 0Hour_ATCACG_L002_002.extract.sam

bowtie -aS -X 800 --offrate 1 trinity-nematostella-raw.renamed -1 <(zcat 6Hour_CGATGT_L002_R1_001.extract.fastq.gz) \
  -2 <(zcat 6Hour_CGATGT_L002_R2_001.extract.fastq.gz) \
  > 6Hour_CGATGT_L002_001.extract.sam

bowtie -aS -X 800 --offrate 1 trinity-nematostella-raw.renamed -1 <(zcat 6Hour_CGATGT_L002_R1_002.extract.fastq.gz) \
  -2 <(zcat 6Hour_CGATGT_L002_R2_002.extract.fastq.gz) \
  > 6Hour_CGATGT_L002_002.extract.sam
```

10.3 Quantify Expression using eXpress

Finally, using eXpress, we'll get abundance estimates for our transcripts. eXpress uses a probabilistic model to efficiently assign mapped reads to isoforms and estimate expression level (see [the website for additional details and relevant publications](#)):

```
~/express-1.5.1-linux_x86_64/express --no-bias-correct -o 0Hour_ATCACG_L002_001.extract.sam-express t
~/express-1.5.1-linux_x86_64/express --no-bias-correct -o 0Hour_ATCACG_L002_002.extract.sam-express t

~/express-1.5.1-linux_x86_64/express --no-bias-correct -o 6Hour_CGATGT_L002_001.extract.sam-express t
~/express-1.5.1-linux_x86_64/express --no-bias-correct -o 6Hour_CGATGT_L002_002.extract.sam-express t
```

This will put the results in a new set of folders named like `<condition>_<barcode>_L002_<replicate>.extract.sam-express`. Each contains a file called *results.xprs* with the results. We'll look at the first ten lines of one of the files using the *head* command:

```
head 0Hour_ATCACG_L002_001.extract.sam-express/results.xprs
```

You should see something like this:

bundle_id	target_id	length	eff_length	tot_counts	uniq_counts	est_counts	eff_counts	ambig_dis
1	nema.id7.tr4	269	0.000000	0	0	0.000000	0.000000	0.000000e+00
2	nema.id1.tr1	811	508.137307	1301	45	158.338092	252.711602	4.777128e+01
2	nema.id2.tr1	790	487.144836	1845	356	1218.927626	1976.727972	1.111471e+02
2	nema.id3.tr1	852	549.122606	1792	3	871.770849	1352.610064	5.493335e+01
2	nema.id4.tr1	675	372.190166	1005	20	88.963433	161.343106	2.836182e+01
3	nema.id62.tr13	2150	1846.657210	9921	9825	9919.902997	11549.404689	1.704940e+03
3	nema.id63.tr13	406	103.720396	360	270	271.097003	1061.173959	1.934732e+02
3	nema.id61.tr13	447	144.526787	6	0	0.000000	0.000000	2.246567e+04
4	nema.id21.tr8	2075	1771.684102	2782	58	958.636395	1122.756883	1.223148e+02

10.4 Differential Expression

First, install R and edgeR:

```
sudo apt-get install -y r-base-core r-bioc-edger csvtool
```

Now, we extract the columns we need from the eXpress outputs and convert it to the appropriate format:

```
csvtool namedcol -t TAB target_id,est_counts 0Hour_ATCACG_L002_001.extract.sam-express/results.xprs
csvtool namedcol -t TAB target_id,est_counts 0Hour_ATCACG_L002_002.extract.sam-express/results.xprs
csvtool namedcol -t TAB target_id,est_counts 6Hour_CGATGT_L002_001.extract.sam-express/results.xprs
csvtool namedcol -t TAB target_id,est_counts 6Hour_CGATGT_L002_002.extract.sam-express/results.xprs
```

We'll be using [edgeR](#) to do the basic differential expression analysis of our counts.

To run [edgeR](#), you need to write a data loading and manipulation script in R. In this case, I've provided one – [diff_exp.R](#). This script will load in two samples with two replicates, execute an MA plot, do an MDS analysis/plot, and provide a spreadsheet with differential expression information in it.

Links:

- [False Discovery Rate](#)
- [Learn R with Swirl](#)

So, download the script:

```
cd /mnt/work
curl -O http://2015-may-nonmodel.readthedocs.org/en/latest/_static/diff_exp.R
```

Now we run the differential expression script with:

```
Rscript diff_exp.R
```

This will produce three files, [nema-edgeR-MA-plot.pdf](#), [nema-edgeR-MDS.pdf](#), and [nema-edgeR.csv](#). The CSV file can be opened directly in Excel; you can also look at it [here](#). It consists of five columns: gene name, log fold change, P-value, and FDR-adjusted P-value.

You can also view more informative versions of these files generated from a different dataset: [chick-edgeR-MA-plot.pdf](#), and [chick-edgeR-MDS.pdf](#).

Remapping your reads to your assembled transcriptome

First, we'll need to make sure `bowtie2` is installed:

```
sudo apt-get install -y bowtie2
```

Now, create a bowtie2 index out of your transcriptome:

```
cd /mnt/work
gunzip -c trinity-nematostella-raw.renamed.fasta.gz > trinity-nematostella-raw.renamed.fasta
bowtie2-build trinity-nematostella-raw.renamed.fasta transcriptome
```

And then, finally, count the number of reads that map to your transcriptome:

```
zcat 0Hour_ATCACG_L002_R1_001.extract.fastq.gz | \
  head -400000 | \
  bowtie2 -U - -x transcriptome > /dev/null
```

You should get something like:

```
97.18% overall alignment rate
```

Miscellaneous advice

12.1 Sequencing depth and number of samples

Hart et al. (2013) provides a nice description and a set of tools for estimating your needed sequencing depth and number of samples. They provide an [Excel based calculator](#) for calculating number of samples. Their numbers are surprisingly large to me ;).

In a proposal for an exploratory effort to discover differentially expressed genes, I would suggest 3-5 biological replicates with 30-50 million reads each. More reads is usually cheaper than more replicates, so 50-100m reads may give you more power to resolve smaller fold changes.

12.2 Downloading your data

If you do your sequencing at the MSU Core Facility, you'll get an e-mail from them when you're samples are ready. The e-mail will give you an FTP site, a username, and a password, as well as a URL. You can use these to download your data. For example, if you get:

```
hostname:      titan.bch.msu.edu
username:      rnaseqmodel
password:      QecheJa6
URI:           ftp://rnaseqmodel:QecheJa6@titan.bch.msu.edu
```

you can go to <ftp://rnaseqmodel:QecheJa6@titan.bch.msu.edu> in your Web browser; that is, it lets you combine your username and password to open that link.

In this case, you will see a 'testdata' directory. If you click on that, you'll see a bunch of fastq.gz files. These are the files that you want to get onto the HPC.

To download these files onto the HPC, log into the HPC, go to the directory on the HPC you want to put the files in, and run a 'wget' – for example, on the HPC:

```
mkdir ~/testdata
cd ~/testdata

wget -r -np -nH ftp://rnaseqmodel:QecheJa6@titan.bch.msu.edu/testdata/
```

This will download `_all_` of the files in that directory. You can also do them one at a time, e.g. to get 'Ath_Mut_1_R1.fastq.gz', you would do

```
wget ftp://rnaseqmodel:QecheJa6@titan.bch.msu.edu/testdata/Ath_Mut_1_R1.fastq.gz
```

Tada!

12.3 Developing your own pipeline

Even if all you plan to do is change the filenames you're operating on, you'll need to develop your own analysis pipeline. Here are some tips.

1. Start with someone else's approach; don't design your own. There are lots of partly done examples that you can find on the Web, including in this tutorial.
2. Generate a data subset (the first few 100k reads, for example).
2. Run commands interactively on an HPC dev node until you get all of the commands basically working; track all of your commands in a Word document or some such.
3. Once you have a set of commands that seems to work on small data, write a script. Run the script on the small data again; make sure that works.
4. Turn it into a qsub script (making sure you're in the right directory, have the modules loaded, etc.)
5. Make sure the qsub script works on your same small data.
6. Scale up to a big test data set.
7. Once that's all working, **SAVE THE SCRIPT SOMEWHERE**. Then, edit it to work on all your data sets (you may want to make subsets again, as much as possible).
8. Provide your scripts and raw counts files as part of any publication or thesis, perhaps via [figshare](#).

Next: [More resources](#)

More resources

13.1 Informational resources

UT (Austin) Sequencing Core [prices](#) - costs and yields for sequencing.

ANGUS - summer NGS [course](#) - lots of resources and materials and book reference

Data Carpentry - intro to R, etc.

Software Carpentry - more scripting, Python, etc.

13.2 Places to share data, scripts, and results files

Figshare.

Miscellaneous questions

1. When should I use de novo assembly, and when should I use [reference-guided \(ab initio\) assembly](#)?

This is always a judgement call, and you can always try both (although there aren't good methods for comparing the results).

The short version is that if you have no nearby genomic sequence, you *must* use de novo assembly; if you have an incomplete genomic sequence you *may* want to use de novo assembly; and if you have a great genomic sequence, you *shouldn't* use de novo assembly.

The positives of using de novo assembly are that you do not depend in any way on the reference. So, if the reference genome is missing, incomplete, or incorrect, you will not have biased results from doing it.

The negatives are that you will get many more isoforms from de novo transcriptome assembly than you will from reference-based transcriptome assembly, and the process is probably a bit more computationally intensive (and certainly more subject to problems from bad data).

2. What are “transcript families”?

Transcript families and components are computational terms for “transcripts that may share exons”. The biological analogy to use is splice isoforms - but keep in mind that the computer can't necessarily tell the difference between transcripts that are “real” splice variants, noisy splicing, different allelic variants of transcripts, recent paralogs, etc. etc. - all the computer knows is that the transcripts share some amount of sequence.

So, transcript families are Trinity's best guess at transcripts that come from the same locus.

3. What should we look at in FastQC results for RNAseq data?

The main thing to pay attention to is the first graph, of quality scores vs position. If your average quality takes a big dip at a particular position, you might consider trimming at that position.

4. How do we transfer our data to Amazon (or any remote computer)?

There are two options –

If your data is on your local computer, you can use Cyberduck to transfer the data to Amazon. (see [Tips and Tricks for working with Remote Computers](#)).

If the data is on a remote computer (like your sequencing center) you can probably use ‘curl’ or ‘wget’ to copy the data directly from the sequencing center to your Amazon computer. You should ask them what the full URL (with username and password) is to each of your data sets, or find your local computer expert to help out.

5. How do we use Amazon to run full analyses?

See [Tips and Tricks for working with Remote Computers](#), “Running full analyses”.

6. Can we use XSEDE or iPlant or <insert other platform here> to run these analyses?

Yes, but you should omit all of the ‘apt-get’ and ‘pip install’ instructions - the sysadmins on those computers will need to install these programs for you.

7. How do we know if our reference transcriptome is “good enough”?

See [Remapping your reads to your assembled transcriptome](#).

8. How do I choose the set of tools to use?

Our recommendations, in order:

- (a) Find a tool that a nearby lab is using, and start there.
- (b) Look at tools and workflows that are used in published papers by groups working in your area.
- (c) Look for good tutorials online.

Tips and Tricks for working with Remote Computers

15.1 Use screen to run things that take a long time.

Often you want to run things that will take days or weeks to run. The ‘screen’ command will let you run programs and record the output, and then come back later and “reconnect”.

For example, try running the beginning bit of digital normalization ([Running digital normalization](#)) inside of screen:

```
screen
cd /mnt/work
normalize-by-median.py -k 20 -p -C 20 -N 4 -x 2e9 -s normC20k20.ct *.pe.qc.fq.gz
```

The normalize-by-median command will take a while, but now that it’s running in screen, you can “detach” from your remote computer and walk away for a bit. For example,

- close your terminal window;
- open up a new one and connect into your Amazon machine;
- type ‘screen -r’ to reconnect into your running screen.

(See [amazon/using-screen](#) for a more complete rundown on instructions.)

15.2 Use CyberDuck to transfer files

To transfer remote files to your local laptop, or local laptop files to the remote system, try using [CyberDuck](#). We’ll walk through it in class.

15.3 Subsetting data

If you want to generate a small subset of a FASTQ file for testing, you can do something like this:

```
gunzip -c /mnt/data/SRR534005_1.fastq.gz | head -400000 | gzip > sample.fq.gz
```

This will take 400,000 lines (or 100,000 FASTQ records) from the beginning of the `SRR534005_1.fastq.gz` file and put them in the `sample.fq.gz` file.

15.4 Running full analyses on Amazon Web Services

You need to do three things to run a full analysis on AWS (or really any cloud machine) –

1. you need to get your data onto that machine.
2. you need to be prepared to let things run for a long time.
3. you need to have a large disk to store all the intermediate files. A good rule of thumb is that every 200 million reads requires about a TB of intermediate disk space.

Getting your data onto the machine can be done by using the ‘curl’ command to download data from (e.g.) your sequencing core. This will be core specific and it’s something we can help you with when you need the help.

To let things run for a long time, you basically need to run them in screen (see above, “Use screen.”)

By default, Amazon doesn’t give you really big hard disks on your machine – you can use ‘df’ to take a look. On an m3.xlarge machine, you can ask about disk space on /mnt by using ‘df’ (disk free):

```
df -k /mnt
```

You should see something like this:

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/xvdb	38565344	20098736	16500940	55%	/mnt

which tells you that /mnt has 40 GB of disk space.

To add disk space to your Amazon instance, see this set of instructions:

<http://angus.readthedocs.org/en/2014/amazon/setting-up-an-ebs-volume.html>

The simplest advice is to make /mnt a 1 TB disk, which should hold a half dozen mRNAseq data sets and all the intermediate data.

Technical information

The github repository for this workshop is publicly available at <https://github.com/ngs-docs/2016-mar-nonmodel>.