labibi Documentation

Release 1.0

C. Titus Brown

August 20, 2016

Contents

Parts of this workshop were given on August 18th and 19th, 2016, by C. Titus Brown and Phillip T. Brooks, at the University of Puerto Rico - Rio Pedras campus.

For more information, please contact Titus directly.

We have an EtherPad for sharing text and asking questions.

Tutorials:

CHAPTER 1

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;
- introduction to philosophy and perspective of data analysis in science;

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 at UC Davis in the School of Vet Med.

Phil Brooks - postdoc at UC Davis.

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;

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.**

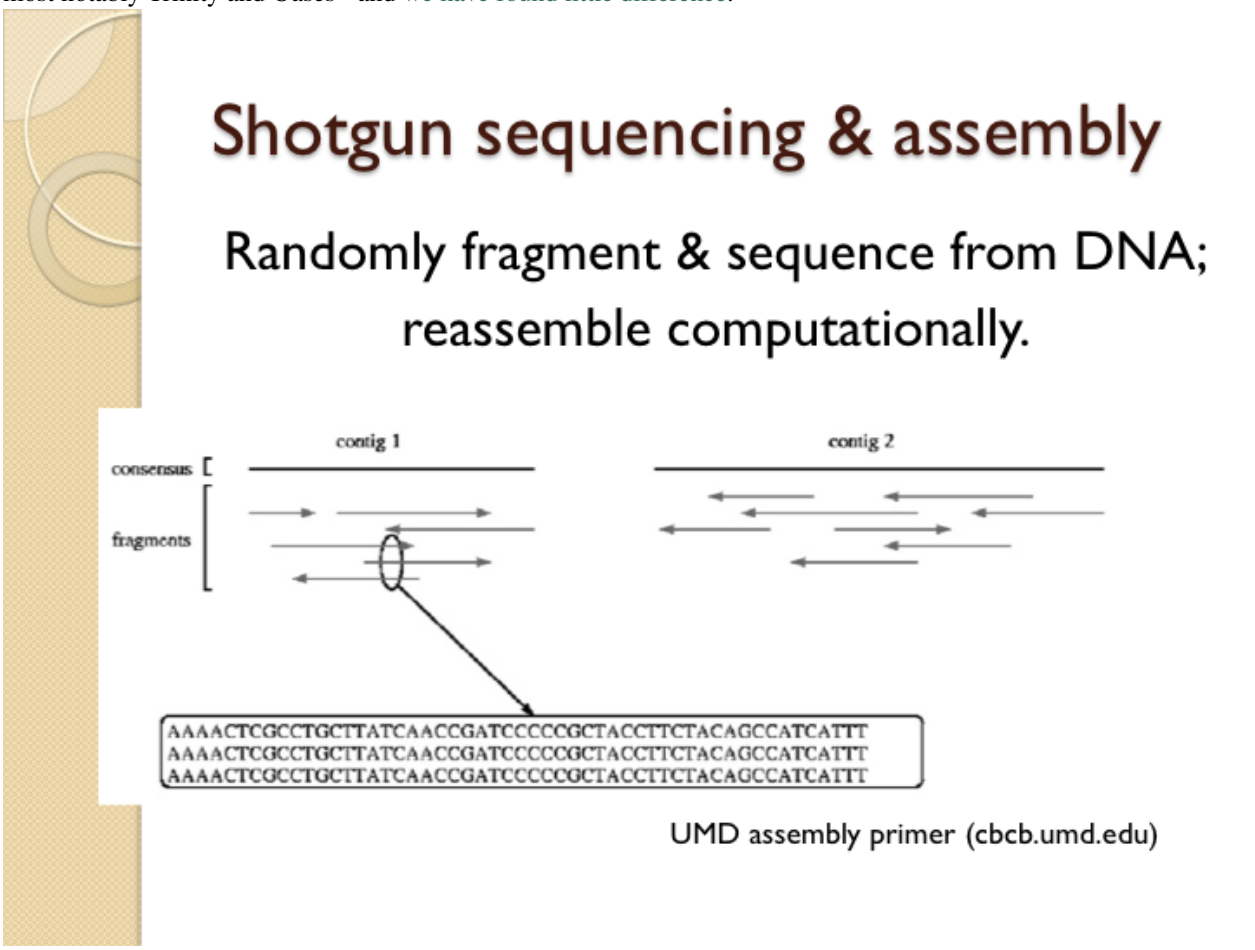
—

Next: Non-model organisms and RNAseq

CHAPTER 2

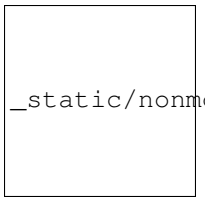
Non-model organisms and RNAseq

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.



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



- 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 (`annotate`)
- Quantify your transcripts and examine differential expression (Quantification and Differential Expression of RNAseq with salmon)
- BLAST your transcripts individually (`n-blast`)

Next: Logging in to your Amazon instance.

CHAPTER 3

Logging in to your Amazon instance.

We've spun up cloud machines and installed a bunch of software for you (see Installation of base image for what, exactly). All you need to do is log in!

So,

1. Find and start the terminal.
2. Copy your hostname from the list that we sent around.
3. Type 'ssh ubuntu@HOSTNAME', pasting in the hostname (set of four . separated numbers) in place of HOSTNAME.
4. Type in the super-secret password that we'll give you.

Done!

Next: Short read quality and trimming

CHAPTER 4

Short read quality and trimming

First, Log into your computer.

OK, you should now be logged into your Amazon computer! You should see something like this:

```
ubuntu@ip-172-30-1-252:~$
```

this is the command prompt.

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.

Note: /mnt is the location we're going to use on Amazon computers, but if you're working on a local cluster, it will have a different location. Talk to your local sysadmin and ask them where they recommend putting lots of short-term working files, i.e. the "scratch" space.

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 for you.

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_
```

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 installed for you.

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

CHAPTER 5

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, 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

CHAPTER 6

Running the actual assembly

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

We've already installed the prerequisites (see Installation of base image); now, install Trinity v2.2.0 itself:

```
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 5G --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

CHAPTER 7

Assembly statistics and evaluation

Note: If you are starting at this point, you'll need a copy of the assembly we just performed (Running the actual assembly). You can set that up by doing:

```
sudo chmod a+rwxt /mnt
cd /mnt
mkdir work
cd work
curl -L -O https://github.com/ngs-docs/2016-aug-nonmodel-rnaseq/raw/master/files/assembly-and-reads.t
tar xzf assembly-and-reads.tar.gz
```

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-1.0.3-linux-x86_64.tar.gz
tar xzf transrate-1.0.3-linux-x86_64.tar.gz
export PATH=~/transrate-1.0.3-linux-x86_64:$PATH
```

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

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

This should give you output like this:

```
[ INFO] 2016-08-20 10:25:51 : n seqs                                60
[ INFO] 2016-08-20 10:25:51 : smallest                             206
[ INFO] 2016-08-20 10:25:51 : largest                             4441
```

```
[ INFO] 2016-08-20 10:25:51 : n bases                      73627
[ INFO] 2016-08-20 10:25:51 : mean len                   1227.12
[ INFO] 2016-08-20 10:25:51 : n under 200                      0
[ INFO] 2016-08-20 10:25:51 : n over 1k                       34
[ INFO] 2016-08-20 10:25:51 : n over 10k                       0
[ INFO] 2016-08-20 10:25:51 : n with orf                      38
[ INFO] 2016-08-20 10:25:51 : mean orf percent             67.01
[ INFO] 2016-08-20 10:25:51 : n90                            816
[ INFO] 2016-08-20 10:25:51 : n70                           1562
[ INFO] 2016-08-20 10:25:51 : n50                           1573
[ INFO] 2016-08-20 10:25:51 : n30                           2017
[ INFO] 2016-08-20 10:25:51 : n10                           3417
[ INFO] 2016-08-20 10:25:51 : gc                            0.47
[ INFO] 2016-08-20 10:25:51 : bases n                          0
[ INFO] 2016-08-20 10:25:51 : proportion n                   0.0
```

...which is pretty useful basic stats.

I'd suggest paying attention to:

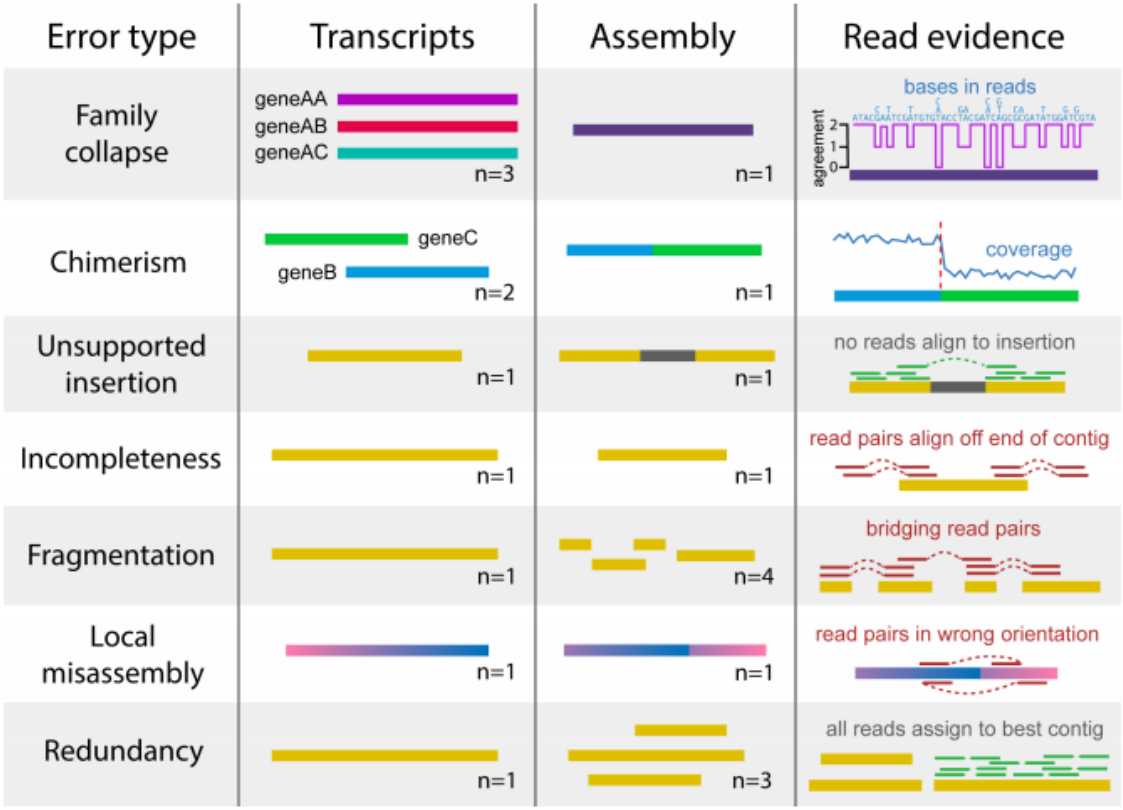
- n seqs
- largest
- mean orf percent

and more or less ignoring the rest on a first pass.

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.



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

```
transrate --assembly=rna-assembly.fa --left=left.fq --right=right.fq --output=transrate_reads
```

The relevant output is here:

```
[ INFO] 2016-08-20 10:27:47 : Read mapping metrics:
[ INFO] 2016-08-20 10:27:47 : ------------------------------------
[ INFO] 2016-08-20 10:27:47 : fragments                    57158
[ INFO] 2016-08-20 10:27:47 : fragments mapped             50840
[ INFO] 2016-08-20 10:27:47 : p fragments mapped            0.89
[ INFO] 2016-08-20 10:27:47 : good mappings                8276
[ INFO] 2016-08-20 10:27:47 : p good mapping               0.14
[ INFO] 2016-08-20 10:27:47 : bad mappings                42564
[ INFO] 2016-08-20 10:27:47 : potential bridges              18
[ INFO] 2016-08-20 10:27:47 : bases uncovered              12751
[ INFO] 2016-08-20 10:27:47 : p bases uncovered             0.17
[ INFO] 2016-08-20 10:27:47 : contigs uncovbase               47
[ INFO] 2016-08-20 10:27:47 : p contigs uncovbase           0.78
[ INFO] 2016-08-20 10:27:47 : contigs uncovered               14
[ INFO] 2016-08-20 10:27:47 : p contigs uncovered           0.23
[ INFO] 2016-08-20 10:27:47 : contigs lowcovered              25
[ INFO] 2016-08-20 10:27:47 : p contigs lowcovered          0.42
[ INFO] 2016-08-20 10:27:47 : contigs segmented               14
[ INFO] 2016-08-20 10:27:47 : p contigs segmented           0.23
```

Here, the percent of good mappings is probably the first number to look at - this is mappings where both members of the pair are aligned in the correct orientation on the same contig, without overlapping either end. (See transrate metrics for more documentation.)

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 one we did with the same starting reads, but without using digital normalization:

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

Compare in both directions:

```
transrate --assembly=rna-assembly.fa --reference=rna-assembly-nodn.fa --output=assembly-compare1
```

and

```
transrate --reference=rna-assembly.fa --assembly=rna-assembly-nodn.fa --output=assembly-compare2
```

First results:

```
[ INFO] 2016-08-20 10:35:35 : Comparative metrics:
[ INFO] 2016-08-20 10:35:35 : -----------------------------------
[ INFO] 2016-08-20 10:35:35 : CRBB hits                          54
[ INFO] 2016-08-20 10:35:35 : n contigs with CRBB                54
[ INFO] 2016-08-20 10:35:35 : p contigs with CRBB               0.9
[ INFO] 2016-08-20 10:35:35 : rbh per reference                 0.9
[ INFO] 2016-08-20 10:35:35 : n refs with CRBB                   32
[ INFO] 2016-08-20 10:35:35 : p refs with CRBB                 0.53
[ INFO] 2016-08-20 10:35:35 : cov25                              18
[ INFO] 2016-08-20 10:35:35 : p cov25                           0.3
[ INFO] 2016-08-20 10:35:35 : cov50                              18
[ INFO] 2016-08-20 10:35:35 : p cov50                           0.3
[ INFO] 2016-08-20 10:35:35 : cov75                              18
[ INFO] 2016-08-20 10:35:35 : p cov75                           0.3
[ INFO] 2016-08-20 10:35:35 : cov85                              16
[ INFO] 2016-08-20 10:35:35 : p cov85                          0.27
[ INFO] 2016-08-20 10:35:35 : cov95                              14
[ INFO] 2016-08-20 10:35:35 : p cov95                          0.23
[ INFO] 2016-08-20 10:35:35 : reference coverage               0.24
```

Second results:

```
[ INFO] 2016-08-20 10:36:45 : Comparative metrics:
[ INFO] 2016-08-20 10:36:45 : -----------------------------------
[ INFO] 2016-08-20 10:36:45 : CRBB hits                          45
[ INFO] 2016-08-20 10:36:45 : n contigs with CRBB                45
[ INFO] 2016-08-20 10:36:45 : p contigs with CRBB              0.75
[ INFO] 2016-08-20 10:36:45 : rbh per reference                0.75
[ INFO] 2016-08-20 10:36:45 : n refs with CRBB                   31
[ INFO] 2016-08-20 10:36:45 : p refs with CRBB                 0.52
[ INFO] 2016-08-20 10:36:45 : cov25                              17
[ INFO] 2016-08-20 10:36:45 : p cov25                          0.28
[ INFO] 2016-08-20 10:36:45 : cov50                              17
[ INFO] 2016-08-20 10:36:45 : p cov50                          0.28
[ INFO] 2016-08-20 10:36:45 : cov75                              16
```

```
[ INFO] 2016-08-20 10:36:45 : p cov75                          0.27
[ INFO] 2016-08-20 10:36:45 : cov85                              15
[ INFO] 2016-08-20 10:36:45 : p cov85                          0.25
[ INFO] 2016-08-20 10:36:45 : cov95                              15
[ INFO] 2016-08-20 10:36:45 : p cov95                          0.25
[ INFO] 2016-08-20 10:36:45 : reference coverage               0.14
```

In this case you can see that our first assembly "covers" more of the other assembly than the other assembly does ours (rbh per reference, and reference coverage). However, you can also see that the assemblies differ quite a bit (for reasons that I haven't tracked down).

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

...although for our assemblies here it doesn't really improve them.

Back to index: 2016 / Aug / mRNAseq on non-model organisms

CHAPTER 8

Quantification and Differential Expression of RNAseq with salmon

Salmon is one of a breed of new, very fast RNAseq counting packages. Like Kallisto and Sailfish, Salmon counts fragments without doing up-front read mapping. Salmon can be used with edgeR and others to do differential expression analysis.

Salmon preprint: http://biorxiv.org/content/early/2015/06/27/021592

Salmon Web site: https://combine-lab.github.io/salmon/

Intro blog post: http://robpatro.com/blog/?p=248

A (very recent) blog post evaluating and comparing methods: https://cgatoxford.wordpress.com/2016/08/17/why-you-should-stop-using-featurecounts-htseq-or-cufflinks2-and-start-using-kallisto-salmon-or-sailfish/

8.1 Installation

Download and unpack salmon, and add it to your path:

```
cd
curl -L -O https://github.com/COMBINE-lab/salmon/releases/download/v0.7.0/Salmon-0.7.0_linux_x86_64.t
tar xzf Salmon-0.7.0_linux_x86_64.tar.gz
export PATH=$PATH:$HOME/SalmonBeta-0.7.0_linux_x86_64/bin
```

8.2 Getting the data

Do:

```
sudo chmod a+rwxt /mnt
mkdir /mnt/data
cd /mnt/data/
curl -O https://s3.amazonaws.com/public.ged.msu.edu/nema-subset.tar.gz
tar xzf nema-subset.tar.gz
```

(This is data from Tulin et al., 2013 that was processed and assembled with the khmer protocols steps 1-3 – basically, what we did in Short read quality and trimming, but for the entire data set.)

Make a directory to work in:

```
mkdir /mnt/quant
```

Copy in the transcriptome from the snapshot:

```
cd /mnt/quant
cp /mnt/data/nema.fa .
```

Index it with salmon:

```
salmon index --index nema_index --transcripts nema.fa --type quasi
```

Link the reads in that we downloaded:

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

Now, quantify the reads against the reference using Salmon:

```
for i in *.1.fq
do
   BASE=$(basename $i .1.fq)
   salmon quant -i nema_index --libType IU \
          -1 $BASE.1.fq -2 $BASE.2.fq -o $BASE.quant;
done
```

(Note that `--libType` must come *before* the read files!)

This will create a bunch of directories named something like `0Hour_ATCACG_L002001.quant`, containing a bunch of files. Take a look at what files there are:

```
find 0Hour_ATCACG_L002001.quant -type f
```

You should see:

```
0Hour_ATCACG_L002001.quant/lib_format_counts.json
0Hour_ATCACG_L002001.quant/cmd_info.json
0Hour_ATCACG_L002001.quant/libParams/flenDist.txt
0Hour_ATCACG_L002001.quant/aux_info/observed_bias.gz
0Hour_ATCACG_L002001.quant/aux_info/observed_bias_3p.gz
0Hour_ATCACG_L002001.quant/aux_info/expected_bias.gz
0Hour_ATCACG_L002001.quant/aux_info/fld.gz
0Hour_ATCACG_L002001.quant/aux_info/meta_info.json
0Hour_ATCACG_L002001.quant/logs/salmon_quant.log
0Hour_ATCACG_L002001.quant/quant.sf
```

The two most interesting files are `lib_format_counts.json` and `quant.sf`. The latter contains the counts; the former contains the log information from running things. Take a look at the counts metadata –

```
less 0Hour_ATCACG_L002001.quant/lib_format_counts.json
```

and see what you think it means... (Use 'q' to quit out of less.)

You might also be interested in the log file –

```
less 0Hour_ATCACG_L002001.quant/logs/salmon_quant.log
```

A few notes –

- what number should you be looking at?
- check out the LIBTYPE docs

So, what should you pay attention to here? Let's list them out...

8.3 Working with the counts

Now, the `quant.sf` files actually contain the relevant information about expression – take a look:

```
head -20 0Hour_ATCACG_L002001.quant/quant.sf
```

The first column contains the transcript names, and the fifth column is what edgeR etc will want - the "raw counts". However, they're not in a convenient location / format for edgeR to use; let's fix that.

Download the `gather-counts.py` script:

```
curl -L -O https://github.com/ngs-docs/2016-aug-nonmodel-rnaseq/raw/master/files/gather-counts.py
```

and run it:

```
python ./gather-counts.py
```

This will give you a bunch of .counts files, processed from the quant.sf files and named for the directory they are in.

Now, run an edgeR script (nema.salmon.R) that loads all this in and calculates a few plots –

```
curl -O -L https://raw.githubusercontent.com/ngs-docs/2016-aug-nonmodel-rnaseq/master/files/nema.salm
Rscript nema.salmon.R
```

These will produce two plots, nema-edgeR-MDS.pdf and nema-edgeR-MA-plot.pdf.

You can see the plot outputs for the whole data set (all the reads) here:

- nema-edgeR-MDS.pdf
- nema-edgeR-MA-plot.pdf (0 vs 6 hour)

8.4 A challenge exercise

How would we create an MA plot comparing 6 Hour vs 12 Hour?

2016 / Aug / mRNAseq on non-model organisms

CHAPTER 9

Miscellaneous advice

9.1 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.

9.2 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!

9.3 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

CHAPTER 10

More resources

10.1 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.

10.2 Places to share data, scripts, and results files

Figshare.

CHAPTER 11

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`.

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.

CHAPTER 12

Tips and Tricks for working with Remote Computers

12.1 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.)

12.2 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.

12.3 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.

12.4 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.

CHAPTER 13

Installation of base image

1. Boot a recent Ubuntu on Amazon (wily 15.10 image or later)
2. Log in as user ‘ubuntu’.
3. Run:

   sudo apt-get -y update && sudo apt-get -y install r-base python3-matplotlib libzmq3-dev python3.5-dev

   texlive-latex-extra texlive-latex-recommended python3-virtualenv trimmomatic fastqc python-pip python-dev bowtie samtools zlib1g-dev ncurses-dev

   sudo pip install -U setuptools khmer==2.0 jupyter jupyter_client ipython pandas

and done!

CHAPTER 14

Technical information

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