labibi Documentation

Release 0.8.2

C. Titus Brown

August 22, 2013

CONTENTS

version 0.8.2 (beta, unreleased)

This is a set of protocols for doing genomic data analysis – specifically, de novo mRNAseq assembly and de novo metagenome assembly – in the cloud.

The latest released version of these protocols is v0.8.2; please use the following URL in citations and discussions:

https://khmer-protocols.readthedocs.org/en/v0.8.2/

CHAPTER

ONE

PROTOCOLS:

1.1 The Eel Pond mRNAseq Tutorial

author Titus Brown, Chris Welcher, and Leigh Sheneman.

Special thanks to Dr. Joshua Rosenthal for his help in testing this!

The tutorial:

1.1.1 0. Downloading and Saving Your Initial Data

We're going to run transcriptome assembly completely in the cloud, because that way (a) you don't need to buy a big computer, and (b) I don't have to figure out all the special details of your own computer system.

This does mean that the first thing you need to do is get your data over to the cloud. I tend to just store it there in the first place, because...

The basics

...Amazon is happy to rent disk space to you, in addition to compute time. They'll rent you disk space in a few different ways, but the way that's most useful for us is through what's called Elastic Block Store. This is essentially a hard-disk rental service.

There are two basic concepts – "volume" and "snapshot". A "volume" can be thought of as a pluggable-in hard drive: you create an empty volume of a given size, attach it to a running instance, and voila! You have extra hard disk space. Volume-based hard disks have two problems, however: first, they cannot be used outside of the "availability zone" they've been created in, which means that you need to be careful to put them in the same zone that your instance is running in; and they can't be shared amongst people.

Snapshots, the second concept, are the solution to transporting and sharing the data on volumes. A "snapshot" is essentially a frozen copy of your volume; you can copy a volume into a snapshot, and a snapshot into a volume.

Getting started

Run through `saving-data-persistently` once, to get the hang of the mechanics. Essentially you create a disk; attach it; format it; and then copy things to and from it.

Downloading and saving your data to a volume

There are *many* different ways of getting big sequence files to and from Amazon. The two that I mostly use are 'curl', which downloads files from a Web site URL; and 'ncftp', which is a robust FTP client that let's you get files from an FTP site. Sequencing centers almost always make their data available in one of these two ways.

Note: To use ncftp on your Amazon instance, you may need to install it:

```
apt-get -y install ncftp
```

For example, to retrieve a file from an FTP site, you would do something like:

```
cd /mnt
ncftp -u <username> ftp://path/to/FTP/site
```

use 'cd' to find the right directory, and then:

```
>> mget *
```

to download the files. Then type 'quit'. You can also use 'curl' to download files one at a time from Web or FTP sites.

Once you have the files, figure out their size using 'du -sk' (e.g. after the above, 'du -sk /mnt' will tell you how much data you have saved under /mnt), and go create and attach a volume (see `saving-data-persistently`).

This data is now something that will *stick around* when you shut down your instance. It's a good rule of thumb to do "savepoints" – whenever you complete a big chunk of work, think about saving the data at that point. I've broken the mRNAseq tutorial down into chunks of work where you can do this – after each Web page, basically.

Some test data

To get started with multfile analysis and assembly, I've provided some test mRNAseq data from embryonic stages of *Nematostella vectensis*; the source is this excellent paper by Tulin et al., "A quantitative reference transcriptome for Nematostella vectensis". The data is on snapshot 'snap-f5a9dea7', so go create a volume from that and mount it as '/data' to get started.

Next: *1. Quality Trimming and Filtering Your Sequences*

1.1.2 1. Quality Trimming and Filtering Your Sequences

Boot up an m1.xlarge machine from Amazon Web Services; this has about 15 GB of RAM, and 2 CPUs, and will be enough to complete the assembly of the Nematostella data set.

Note: This follows the NGS 2013 tutorial, Short-read quality evaluation, but for multiple files.

Note: The end results of this tutorial are available as public snapshot snap-8b155fd9 on EC2/EBS.

Install software

Install screed:

```
cd /usr/local/share
git clone https://github.com/ged-lab/screed.git
cd screed
python setup.py install
```

Install the bleeding-edge version of khmer:

```
cd /usr/local/share
git clone https://github.com/ged-lab/khmer.git -b bleeding-edge
cd khmer
make

echo 'export PYTHONPATH=/usr/local/share/khmer/python' >> ~/.bashrc
source ~/.bashrc
```

Install Trimmomatic:

```
cd /root
curl -O http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.27.zip
unzip Trimmomatic-0.27.zip
cp Trimmomatic-0.27/trimmomatic-0.27.jar /usr/local/bin
```

Install libgtextutils and fastx:

```
cd /root
curl -O http://hannonlab.cshl.edu/fastx_toolkit/libgtextutils-0.6.1.tar.bz2
tar xjf libgtextutils-0.6.1.tar.bz2
cd libgtextutils-0.6.1/
./configure && make && make install

cd /root
curl -O http://hannonlab.cshl.edu/fastx_toolkit/fastx_toolkit-0.0.13.2.tar.bz2
tar xjf fastx_toolkit-0.0.13.2.tar.bz2
cd fastx_toolkit-0.0.13.2/
./configure && make && make install
```

In each of these cases, we're downloading the software – you can use google to figure out what each package is and does if we don't discuss it below. We're then unpacking it, sometimes compiling it (which we can discuss later), and then installing it for general use.

Find your data

Either load in your own data (as in *0. Downloading and Saving Your Initial Data*) or create a volume from snapshot snap-f5a9dea7 and mount it as `/data` (again, this is the data from Tulin et al., 2013).

Check:

```
ls /data
```

If you see all the files you think you should, good! Otherwise, debug.

If you're using the Tulin et al. data provided in the snapshot above, you should see a bunch of files like:

```
/data/0Hour_ATCACG_L002_R1_001.fastq.gz
```

Link your data into a working directory

Rather than *copying* the files into the working directory, let's just *link* them in – this creates a reference so that UNIX knows where to find them but doesn't need to actually move them around.

```
cd /mnt
mkdir work
cd work

ln -fs /data/*.fastq.gz .
```

(The 'ln' command is what does the linking.)

Now, do an 'ls' to list the files. If you see only one entry, `*.fastq.gz`, then the ln command above didn't work properly. One possibility is that your files aren't in /data; another is that they're not named *.fastq.gz.

Download the Illumina adapters

In the working directory,

```
curl -O https://s3.amazonaws.com/public.ged.msu.edu/illuminaClipping.fa
```

Note: You'll need to make sure these are the right adapters for your data. If they are, you should see that some of them are trimmed off, below; if they're not, you shouldn't see anything get trimmed.

Adapter trim each pair of files

(From this point on, you may want to be running things inside of screen, so that you detach and log out while it's running; see `using-screen` for more information.)

If you're following along using the Nematostella data, you should have a bunch of files that look like this (use 'ls' to show them):

```
24HourB_GCCAAT_L002_R1_001.fastq.gz
                      ^^
```

Each file with an R1 in its name should have a matching file with an R2 – these are the paired ends.

Note: You'll need to replace <R1 FILE> and <R2 FILE>, below, with the names of your actual R1 and R2 files. You'll also need to replace <SAMPLE NAME> with something that's unique to each pair of files. It doesn't really matter what, but you need to make sure it's different for each pair of files.

For *each* of these pairs, run the following:

```
# make a temp directory
mkdir trim
cd trim

# run trimmomatic
java -jar /usr/local/bin/trimmomatic-0.27.jar PE <R1 FILE> <R2 FILE> s1_pe s1_se s2_pe s2_se ILLUMINA

# interleave the remaining paired-end files
/usr/local/share/khmer/scripts/interleave-reads.py s1_pe s2_pe | gzip -9c > ../<SAMPLE NAME>.pe.fq.gz

# combine the single-ended files
```

```
cat s1_se s2_se | gzip -9c > ../<SAMPLE NAME>.se.fq.gz

# go back up to the working directory and remove the temp directory
cd ..
rm -r trim

# make it hard to delete the files you just created
chmod u-w *.pe.fq.gz *.se.fq.gz
```

To get a basic idea of what's going on, please read the '#' comments above, but, briefly, this set of commands:

- creates a temporary directory, 'trim/'
- runs 'Trimmomatic' in that directory to trim off the adapters, and then puts remaining pairs (most of them!) in s1_pe and s2_pe, and any orphaned singletons in s1_se and s2_se.
- interleaves the paired ends and puts them back in the working directory
- combines the orphaned reads and puts them back in the working directory

At the end of this you will have new files ending in '.pe.fq.gz' and '.se.fq.gz', representing the paired and orphaned quality trimmed reads, respectively.

Automating things a bit

OK, once you've done this once or twice, it gets kind of tedious, doesn't it? I've written a script to write these commands out automatically. Run it like so:

```
cd /mnt/work
python /usr/local/share/khmer/sandbox/write-trimmomatic.py > trim.sh
```

Run this, and then look at 'trim.sh' using the 'more' command –

```
more trim.sh
```

If it looks like it contains the right commands, you can run it by doing:

```
bash trim.sh
```

Note: This is a prime example of scripting to make your life much easier and less error prone. Take a look at this file sometime – 'more /usr/local/share/khmer/sandbox/write-trimmomatic.py' – to get some idea of how this works.

Quality trim each pair of files

After you run this, you should have a bunch of '.pe.fq.gz' files and a bunch of '.se.fq.gz' files. The former are files that contain paired, interleaved sequences; the latter contain single-ended, non-interleaved sequences.

Next, for each of these files, run:

```
gunzip -c <filename> | fastq_quality_filter -Q33 -q 30 -p 50 | gzip -9c > <filename>.qc.fq.gz
```

This uncompresses each file, removes poor-quality sequences, and then recompresses it. Note that (following Short-read quality evaluation) you can also trim to a specific length by putting in a 'fastx_trimmer -Q33 -l 70 |' into the mix.

If fastq_quality_filter complains about invalid quality scores, try removing the -Q33 in the command; Illumina has blessed us with multiple quality score encodings.

Automating this step

This step can be automated with a 'for' loop at the shell prompt. Try:

```
for i in *.pe.fq.gz *.se.fq.gz
do
     echo working with $i
     newfile="$(basename $i .fq.gz)"
     gunzip -c $i | fastq_quality_filter -Q33 -q 30 -p 50 | gzip -9c > "${newfile}.qc.fq.gz"
done
```

What this loop does is:

- for every file ending in pe.fq.gz and se.fq.gz,
- print out a message with the filename,
- construct a name 'newfile' that omits the trailing .fq.gz
- uncompresses the original file, passes it through fastq, recompresses it, and saves it as 'newfile'.qc.fq.gz

Extracting paired ends from the interleaved files

The fastx utilities that we're using to do quality trimming aren't paired-end aware; they're removing individual sequences. Because the pe files are interleaved, this means that there may now be some orphaned sequences in there. Downstream, we will want to pay special attention to the remaining paired sequences, so we want to separate out the pe and se files. How do we go about that? Another script, of course!

The khmer script 'strip-and-split-for-assembly.py' does exactly that. You run it on an interleaved file that may have some orphans, and it produces .pe and .se files afterwards, containing pairs and orphans respectively.

To run it on all of the pe qc files, do:

```
for i in *.pe.qc.fq.gz
do
   /usr/local/share/khmer/scripts/extract-paired-reads.py $i
done
```

Finishing up

You should now have a whole mess of files. For example, in the Nematostella data, for *each* of the original input files, you'll have:

```
24HourB_GCCAAT_L002_R1_001.fastq.gz              - the original data
24HourB_GCCAAT_L002_R2_001.fastq.gz
24HourB_GCCAAT_L002_R1_001.pe.fq.gz              - adapter trimmed pe
24HourB_GCCAAT_L002_R1_001.pe.qc.fq.gz           - FASTX filtered
24HourB_GCCAAT_L002_R1_001.pe.qc.fq.gz.pe        - FASTX filtered PE
24HourB_GCCAAT_L002_R1_001.pe.qc.fq.gz.se        - FASTX filtered SE
24HourB_GCCAAT_L002_R1_001.se.fq.gz              - adapter trimmed orphans
24HourB_GCCAAT_L002_R1_001.se.qc.fq.gz           - FASTX filtered orphans
```

Yikes! What to do?

Well, first, you can get rid of the original data. You already have it on a disk somewhere, right?

```
rm *.fastq.gz
```

Next, you can get rid of the 'pe.fq.gz' and 'se.fq.gz' files, since you only want the QC files. So:

```
rm *.pe.fq.gz *.se.fq.gz
```

And, finally, you can toss the pe.fq.gz files, because you've turned *those* into .pe and .se files.

```
rm *.pe.qc.fq.gz
```

So now you should be left with only three files for each sample:

```
24HourB_GCCAAT_L002_R1_001.pe.qc.fq.gz.pe   - FASTX filtered PE
24HourB_GCCAAT_L002_R1_001.pe.qc.fq.gz.se   - FASTX filtered SE
24HourB_GCCAAT_L002_R1_001.se.qc.fq.gz      - FASTX filtered orphans
```

Things to think about

Note that the filenames, while ugly, are conveniently structured with the history of what you've done. This is a good idea.

Also note that we've conveniently named the files so that we can remove the unwanted ones en masse. This is a good idea, too.

Renaming files

I'm a fan of keeping the files named somewhat sensibly, and keeping them compressed. Let's do some mass renaming:

```
for i in *.pe.qc.fq.gz.pe
do
   newfile="$(basename $i .pe.qc.fq.gz.pe).pe.qc.fq"
   mv $i $newfile
   gzip $newfile
done
```

and also some mass combining:

```
for i in *.pe.qc.fq.gz.se
do
  otherfile="$(basename $i .pe.qc.fq.gz.se).se.qc.fq.gz"
  gunzip -c $otherfile > combine
  cat $i >> combine
  gzip -c combine > $otherfile
  rm $i
done
```

and finally, make the end product files read-only:

```
chmod u-w *.qc.fq.gz
```

to make sure you don't accidentally delete something.

Saving the files

At this point, you should save these files, which will be used in two ways: first, for assembly; and second, for mapping, to do quantitation and ultimately comparative expression analysis. You can save them by doing this:

```
mkdir save
mv *.qc.fq.gz save
du -sk save
```

This puts the data you want to save into a subdirectory named 'save', and calculates the size.

Now, create a volume of the given size – divide by a thousand to get gigabytes, multiply by 1.1 to make sure you have enough room, and then follow the instructions in `saving-data-persistently`. Once you've mounted it properly (I would suggest mounting it on /save instead of /data!), then do

```
rsync -av save /save
```

which will copy all of the files over from the ./save directory onto the '/save' disk. Then 'umount /save' and voila, you've got a copy of the files!

Next stop: *2. Applying Digital Normalization*.

1.1.3 2. Applying Digital Normalization

Note: You can start this tutorial with the contents of EC2/EBS snapshot snap-126cc847.

Note: You'll need ~15 GB of RAM for this, or more if you have a LOT of data.

Link in your data

Make sure your data is in /mnt/work/. If you've loaded it onto /data, you can do:

```
cd /mnt
mkdir work
cd /mnt/work
ln -fs /data/*.qc.fq.gz .
```

Run digital normalization

Apply digital normalization to the paired-end reads:

```
/usr/local/share/khmer/scripts/normalize-by-median.py -p -k 20 -C 20 -N 4 -x 3e9 --savehash normC20k
```

and then to the single-end reads:

```
/usr/local/share/khmer/scripts/normalize-by-median.py -C 20 --loadhash normC20k20.kh --savehash norm
```

Note the '-p' in the first normalize-by-median command – when run on PE data, that ensures that no paired ends are orphaned. However, it will complain on single-ended data, so you have to give the data to it separately.

Also note the '-N' and '-x' parameters. These specify how much memory diginorm should use. The product of these should be less than the memory size of the machine you selected. The maximum needed for *any* transcriptome should be in the ~60 GB range, e.g. -N 4 -x 15e9; for only a few hundred million reads, 16 GB should be plenty. (See choosing hash sizes for khmer for more information.)

Trim off likely erroneous k-mers

Now, run through all the reads and trim off low-abundance parts of high-coverage reads:

```
/usr/local/share/khmer/scripts/filter-abund.py -V normC20k20.kh *.keep
```

This will turn some reads into orphans, but that's ok – their partner read was bad.

Rename files

You'll have a bunch of 'keep.abundfilt' files – let's make things prettier.

First, let's break out the orphaned and still-paired reads:

```
for i in *.pe.*.abundfilt;
do
   /usr/local/share/khmer/scripts/extract-paired-reads.py $i
done
```

We can combine the orphaned reads into a single file:

```
for i in *.se.qc.fq.gz.keep.abundfilt
do
   pe_orphans=$(basename $i .se.qc.fq.gz.keep.abundfilt).pe.qc.fq.gz.keep.abundfilt.se
   newfile=$(basename $i .se.qc.fq.gz.keep.abundfilt).se.qc.keep.abundfilt.fq.gz
   cat $i $pe_orphans | gzip -c > $newfile
done
```

We can also rename the remaining PE reads & compress those files:

```
for i in *.abundfilt.pe
do
   newfile=$(basename $i .fq.gz.keep.abundfilt.pe).keep.abundfilt.fq
   mv $i $newfile
   gzip $newfile
done
```

This leaves you with a whole passel o' files, most of which you want to go away!

```
6Hour_CGATGT_L002_R1_005.pe.qc.fq.gz
6Hour_CGATGT_L002_R1_005.pe.qc.fq.gz.keep
6Hour_CGATGT_L002_R1_005.pe.qc.fq.gz.keep.abundfilt
6Hour_CGATGT_L002_R1_005.pe.qc.fq.gz.keep.abundfilt.se
6Hour_CGATGT_L002_R1_005.se.qc.fq.gz
6Hour_CGATGT_L002_R1_005.se.qc.fq.gz.keep
6Hour_CGATGT_L002_R1_005.se.qc.fq.gz.keep.abundfilt
```

So, finally, let's get rid of a lot of the old files

```
rm *.se.qc.fq.gz.keep.abundfilt
rm *.pe.qc.fq.gz.keep.abundfilt.se
rm *.keep
rm *.abundfilt
rm *.qc.fq.gz
```

Gut check

You should now have:

```
6Hour_CGATGT_L002_R1_005.pe.qc.keep.abundfilt.fq.gz
6Hour_CGATGT_L002_R1_005.se.qc.keep.abundfilt.fq.gz
```

These files are, respectively, the paired (pe) quality-filtered (qc) digitally normalized (keep) abundance-trimmed (abundfilt) FASTQ (fq) gzipped (gz) sequences, and the orphaned (se) quality-filtered (qc) digitally normalized (keep) abundance-trimmed (abundfilt) FASTQ (fq) gzipped (gz) sequences.

Save all these files to a new volume, and get ready to assemble!

Next: *3. Running the Actual Assembly*.

1.1.4 3. Running the Actual Assembly

All of the below should be run in screen, probably... You will want at least 15 GB of RAM, maybe more.

(If you start up a new machine, you'll need to go to *1. Quality Trimming and Filtering Your Sequences* and install khmer and screed.)

Note: You can start this tutorial with the contents of EC2/EBS snapshot snap-7b0b872e.

Installing Trinity

To install Trinity:

```
cd /root

curl -L http://sourceforge.net/projects/trinityrnaseq/files/latest/download?source=files > trinity.ta

tar xzf trinity.tar.gz
cd trinityrnaseq*
export FORCE_UNSAFE_CONFIGURE=1
make
```

Install bowtie

Download and install bowtie:

```
cd /root
curl -O -L http://sourceforge.net/projects/bowtie-bio/files/bowtie/0.12.7/bowtie-0.12.7-linux-x86_64
unzip bowtie-0.12.7-linux-x86_64.zip
cd bowtie-0.12.7
cp bowtie bowtie-build bowtie-inspect /usr/local/bin
```

Build the files to assemble

For paired-end data, Trinity expects two files, 'left' and 'right'; there can be orphan sequences present, however. So, below, we split all of our interleaved pair files in two, and then add the single-ended seqs to one of 'em.

```
cd /mnt/work
for i in *.pe.qc.keep.abundfilt.fq.gz
do
     python /usr/local/share/khmer/scripts/split-paired-reads.py $i
done

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

gunzip -c *.se.qc.keep.abundfilt.fq.gz >> left.fq
```

Assembling with Trinity

Run the assembler!

```
/root/trinityrnaseq_r2013-02-25/Trinity.pl --left left.fq --right right.fq --seqType fq -JM 15G
```

Note that this last bit (15G) is the maximum amount of memory to use. You can increase (or decrease) it based on what machine you rented. This size works for the m1.xlarge machines.

Once this completes (on the Nematostella data it might take about 12 hours), you'll have an assembled transcriptome in trinity_out_dir/Trinity.fasta.

You can now copy it over via Dropbox, or set it up for BLAST (see *BLASTing your assembled data*).

Next: *5. Building transcript families and annotating the sequences*.

1.1.5 BLASTing your assembled data

One thing everyone wants to do is BLAST sequence data, right? Here's a simple way to set up a stylish little BLAST server that lets you search your newly assembled sequences.

Installing blastkit

Installing some prerequisites:

```
pip install pygr
pip install whoosh
pip install git+https://github.com/ctb/pygr-draw.git
pip install git+https://github.com/ged-lab/screed.git
apt-get -y install lighttpd
```

and configure them:

```
cd /etc/lighttpd/conf-enabled
ln -fs ../conf-available/10-cgi.conf ./
echo 'cgi.assign = ( ".cgi" => "" )' >> 10-cgi.conf
echo 'index-file.names += ( "index.cgi" ) ' >> 10-cgi.conf

/etc/init.d/lighttpd restart
```

Next, install BLAST:

```
cd /root

curl -O ftp://ftp.ncbi.nih.gov/blast/executables/release/2.2.24/blast-2.2.24-x64-linux.tar.gz
tar xzf blast-2.2.24-x64-linux.tar.gz
cp blast-2.2.24/bin/* /usr/local/bin
cp -r blast-2.2.24/data /usr/local/blast-data
```

And put in blastkit:

```
cd /root
git clone https://github.com/ctb/blastkit.git -b ec2
cd blastkit/www
ln -fs $PWD /var/www/blastkit

mkdir files
```

```
chmod a+rxwt files
chmod +x /root
```

and run check.py:

```
cd /root/blastkit
python ./check.py
```

It should say everything is OK.

Adding the data

If you've just finished a transcriptome assembly (*3. Running the Actual Assembly*) then you can do this to copy your newly generated assembly into the right place:

```
cp trinity_out_dir/Trinity.fasta /root/blastkit/db/db.fa
```

Alternatively, you can grab my version of the assembly (from running this tutorial):

```
cd /root/blastkit
curl -O https://s3.amazonaws.com/public.ged.msu.edu/trinity-nematostella-raw.fa.gz
gunzip trinity-nematostella-raw.fa.gz
mv trinity-nematostella-raw.fa db/db.fa
```

Formatting the database

After you've done either of the above, format and install the database for blastkit:

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

Done!

Note: You can install any file of DNA sequences you want this way; just copy it into /root/blastkit/db/db.fa and run the indexing commands, above.

Running blastkit

Figure out what your machine name is (ec2-???-???-???-???.compute-1.amazonaws.com) and go to:

```
http://machine-name/blastkit/
```

Make sure you have enabled port 80 in your security settings on Amazon.

(If you're using the Nematostella data set, try this sequence:

```
CAGCCTTTAGAAGGAAACAGTGGCAATATATAATTCTAGATGAAGCTCAGAATATCAAAA
ATTTTAAAAGTCAAAGGTGGCAGTTGCTGTTGAATTTTTCAAGTCAGAGGAGACTTTTGT
TGACTGGAACACCTTTGCAGAACAATTTGATGGAGCTGTGGTCGCTTATGCATTTCCTCA
TGCCATCAATGTTTGCTTCTCATAAAGATTTTAGGGAGTGGTTTTCTAACCCTGTTACAG
GGATGATTGAAGGGAATTCAG
```

It should match something in your assembly.)

1.1.6 5. Building transcript families and annotating the sequences

Install khmer, screed, and BLAST. (See *1. Quality Trimming and Filtering Your Sequences* and *BLASTing your assembled data*). I would suggest using an m1.large or m1.xlarge machine.

You'll also need to install some eel-pond scripts:

```
git clone https://github.com/ctb/eel-pond.git /usr/local/share/eel-pond
```

Copy in your data

You need to get ahold of your assembled transcriptome (from e.g. *3. Running the Actual Assembly*). Put it in /mnt.

For the purposes of your first run through, I suggest just grabbing my copy of the Nematostella assembly:

```
cd /mnt
curl -O https://s3.amazonaws.com/public.ged.msu.edu/trinity-nematostella-raw.fa.gz
```

Run khmer partitioning

Partitioning runs a de Bruijn graph-based clustering algorithm that will cluster your transcripts by transitive sequence overlap. That is, it will build transcript families :).

```
/usr/local/share/khmer/scripts/do-partition.py -x 1e9 -N 4 --threads 4 nema trinity-nematostella-raw
```

This should take about 15 minutes, and outputs a file ending in '.part' that contains the partition assignments. Now, group and rename the sequences:

```
python /usr/local/share/eel-pond/rename-with-partitions.py nema trinity-nematostella-raw.fa.gz.part
mv trinity-nematostella-raw.fa.gz.part.renamed.fasta.gz trinity-nematostella.renamed.fa.gz
```

Looking at the renamed sequences

Let's look at the renamed sequences:

```
gunzip -c trinity-nematostella.renamed.fa.gz | head
```

You'll see that each sequence name looks like this:

```
>nema.id1.tr16001 1_of_1_in_tr16001 len=261 id=1 tr=16001
```

Some explanation:

- 'nema' is the prefix that you gave the rename script, above; modify accordingly for your own organism. It's best to change it each time you do an assembly, just to keep things straight.
- **'idN' is the unique ID for this sequence; it will never be repeated in this** file.
- 'trN' is the transcript family, which may contain one or more transcripts.
- '1_of_1_in_tr16001' tells you that this transcript family has only one transcript in it (this one!) Other transcript families may (will) have more.
- 'len' is the sequence length.

Doing a preliminary annotation against mouse

Now let's assign putative homology & orthology to these transcripts, by doing BLASTs & reciprocal best hit analysis. First, uncompress your transcripts file:

```
gunzip trinity-nematostella.renamed.fa.gz
```

Now, grab the latest mouse RefSeq:

```
curl -O ftp://ftp.ncbi.nih.gov/refseq/M_musculus/mRNA_Prot/mouse.protein.faa.gz
gunzip mouse.protein.faa.gz
```

Format both as BLAST databases:

```
formatdb -i mouse.protein.faa -o T -p T
formatdb -i trinity-nematostella.renamed.fa -o T -p F
```

And, now, run BLAST in both directions. Note, this may take ~24 hours or longer; you probably want to run it in screen:

```
blastall -i trinity-nematostella.renamed.fa -d mouse.protein.faa -e 1e-3 -p blastx -o nema.x.mouse -a
blastall -i mouse.protein.faa -d trinity-nematostella.renamed.fa -e 1e-3 -p tblastn -o mouse.x.nema -
```

Note: These BLASTs will take a *long* time, like 24-36 hours. If you want to work with canned BLASTs, do:

```
curl -O http://athyra.idyll.org/~t/nema.x.mouse.gz
curl -O http://athyra.idyll.org/~t/mouse.x.nema.gz
gunzip nema.x.mouse.gz
gunzip mouse.x.nema.gz
```

Assigning names to sequences

Now, calculate putative homology (best BLAST hit) and orthology (reciprocal best hits):

```
python /usr/local/share/eel-pond/make-uni-best-hits.py nema.x.mouse nema.x.mouse.homol
python /usr/local/share/eel-pond/make-reciprocal-best-hits.py nema.x.mouse mouse.x.nema nema.x.mouse
```

Prepare some of the mouse info:

```
python /usr/local/share/eel-pond/make-namedb.py mouse.protein.faa mouse.namedb
python -m screed.fadbm mouse.protein.faa
```

And, finally, annotate the sequences:

```
python /usr/local/share/eel-pond/annotate-seqs.py trinity-nematostella.renamed.fa nema.x.mouse.ortho
```

This will produce a file 'trinity-nematostella.renamed.fa.annot', which will have sequences that look like this:

```
>nematostella.id1.tr115222 h=43% => suppressor of tumorigenicity 7 protein isoform 2 [Mus musculus] 1
```

I suggest renaming this file to 'nematostella.fa' and using it for BLASTs (see *BLASTing your assembled data*).

1.2 The Kalamazoo Metagenome Assembly Tutorial

author Adina Howe and Titus Brown

1.2.1 1. Quality Trimming and Filtering Your Sequences

Boot up an m1.xlarge machine from Amazon Web Services; this has about 15 GB of RAM, and 2 CPUs, and will be enough to complete the assembly of the example data set.

Note: This follows the NGS 2013 tutorial, Short-read quality evaluation, but for multiple files.

Note: The end results of this tutorial are available as public snapshot XXX on EC2/EBS.

Also see: `../mrnaseq/using-screen`.

Install software

Install screed:

```
pip install git+https://github.com/ged-lab/screed.git
```

Install the bleeding-edge version of khmer:

```
cd /usr/local/share
git clone https://github.com/ged-lab/khmer.git -b bleeding-edge
cd khmer
make

echo 'export PYTHONPATH=/usr/local/share/khmer/python' >> ~/.bashrc
source ~/.bashrc
```

Install Trimmomatic:

```
cd /root
curl -O http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.27.zip
unzip Trimmomatic-0.27.zip
cp Trimmomatic-0.27/trimmomatic-0.27.jar /usr/local/bin
```

Install libgtextutils and fastx:

```
cd /root
curl -O http://hannonlab.cshl.edu/fastx_toolkit/libgtextutils-0.6.1.tar.bz2
tar xjf libgtextutils-0.6.1.tar.bz2
cd libgtextutils-0.6.1/
./configure && make && make install

cd /root
curl -O http://hannonlab.cshl.edu/fastx_toolkit/fastx_toolkit-0.0.13.2.tar.bz2
tar xjf fastx_toolkit-0.0.13.2.tar.bz2
cd fastx_toolkit-0.0.13.2/
./configure && make && make install
```

In each of these cases, we're downloading the software – you can use google to figure out what each package is and does if we don't discuss it below. We're then unpacking it, sometimes compiling it (which we can discuss later), and then installing it for general use.

Create a working directory

Let's create a place to work:

```
cd /mnt
mkdir assembly
cd assembly
```

Link in the data

Trim and quality filter

Grab some Illumina adapters:

```
curl -O https://s3.amazonaws.com/public.ged.msu.edu/illuminaClipping.fa
```

Trim the first data set (~20 minutes):

```
mkdir trim
cd trim

java -jar /usr/local/bin/trimmomatic-0.27.jar PE ../SRR492065_?.fastq.gz s1_pe s1_se s2_pe s2_se ILLU

/usr/local/share/khmer/scripts/interleave-reads.py s?_pe > combined.fq

fastq_quality_filter -Q33 -q 30 -p 50 -i combined.fq > combined-trim.fq
fastq_quality_filter -Q33 -q 30 -p 50 -i s1_se > s1_se.trim
/usr/local/share/khmer/scripts/extract-paired-reads.py combined-trim.fq

gzip -9c combined-trim.fq.pe > ../SRR492065.pe.qc.fq.gz
gzip -9c combined-trim.fq.se s1_se > ../SRR492065.se.qc.fq.gz

cd ../
rm -fr trim
```

Trim the second data set (~20 minutes):

```
mkdir trim
cd trim

java -jar /usr/local/bin/trimmomatic-0.27.jar PE ../SRR492066_?.fastq.gz s1_pe s1_se s2_pe s2_se ILLU

/usr/local/share/khmer/scripts/interleave-reads.py s?_pe > combined.fq

fastq_quality_filter -Q33 -q 30 -p 50 -i combined.fq > combined-trim.fq
fastq_quality_filter -Q33 -q 30 -p 50 -i s1_se > s1_se.trim
/usr/local/share/khmer/scripts/extract-paired-reads.py combined-trim.fq

gzip -9c combined-trim.fq.pe > ../SRR492066.pe.qc.fq.gz
gzip -9c combined-trim.fq.se s1_se > ../SRR492066.se.qc.fq.gz

cd ../
rm -fr trim
```

Done! Now you have four files: SRR492065.pe.qc.fq.gz, SRR492065.se.qc.fq.gz, SRR492066.pe.qc.fq.gz, and SRR492066.se.qc.fq.gz.

The '.pe' files are interleaved paired-end; you can take a look at them like so:

```
gunzip -c SRR492065.pe.qc.fq.gz | head
```

The other two are single-ended files, where the reads have been orphaned because we discarded stuff.

All four files are in FASTQ format.

Next: *2. Running digital normalization*

1.2.2 2. Running digital normalization

Note: Make sure you're running in screen!

Start with the QC'ed files from *1. Quality Trimming and Filtering Your Sequences* or copy them into a working directory.

Run a first round of digital normalization

Normalize everything to a coverage of 20, starting with the (more valuable) PE reads; keep pairs using '-p':

```
/usr/local/share/khmer/scripts/normalize-by-median.py -k 20 -C 20 -N 4 -x 5e8 -p --savehash normC20k2
```

...and continuing into the (less valuable but maybe still useful) SE reads:

```
/usr/local/share/khmer/scripts/normalize-by-median.py -C 20 --savehash normC20k20.kh --loadhash normC
```

This produces a set of '.keep' files, as well as a normC20k20.kh database file.

Error-trim your data

Use 'filter-abund' to trim off any k-mers that are abundance-1 in high-coverage reads (-V option, for variable coverage):

```
/usr/local/share/khmer/scripts/filter-abund.py -V normC20k20.kh *.keep
```

This produces .abundfilt files.

The process of error trimming could have orphaned reads, so split the PE file into still-interleaved and non-interleaved reads:

```
for i in *.pe.qc.fq.gz.keep.abundfilt
do
   /usr/local/share/khmer/scripts/extract-paired-reads.py $i
done
```

This leaves you with PE files (.pe.qc.fq.gz.keep.abundfilt.pe :) and two sets of SE files (.se.qc.fq.gz.keep.abundfilt and .pe.qc.fq.gz.keep.abundfilt.se). (Yes, I did indeed devise this naming scheme. It makes sense. Trust me.)

Normalize down to C=5

Now that we've eliminated many more erroneous k-mers, let's ditch some more high-coverage data. First, normalize the paired-end reads:

```
/usr/local/share/khmer/scripts/normalize-by-median.py -C 5 -k 20 -N 4 -x 5e8 --savehash normC5k20.kh
```

and then do the remaining single-ended reads:

```
/usr/local/share/khmer/scripts/normalize-by-median.py -C 5 --savehash normC5k20.kh --loadhash normC5
```

Compress and combine the files

Now let's tidy things up. Here are the paired files (kak = keep/abundfilt/keep):

```
gzip -9c SRR492065.pe.qc.fq.gz.keep.abundfilt.pe.keep > SRR492065.pe.kak.qc.fq.gz
gzip -9c SRR492066.pe.qc.fq.gz.keep.abundfilt.pe.keep > SRR492066.pe.kak.qc.fq.gz
```

and the single-ended files:

```
gzip -9c SRR492066.pe.qc.fq.gz.keep.abundfilt.se.keep SRR492066.se.qc.fq.gz.keep.abundfilt.keep > SRR
gzip -9c SRR492065.pe.qc.fq.gz.keep.abundfilt.se.keep SRR492065.se.qc.fq.gz.keep.abundfilt.keep > SRR
```

You can now remove all of these various files:

```
SRR492066.pe.qc.fq.gz.keep
SRR492066.pe.qc.fq.gz.keep.abundfilt
SRR492066.pe.qc.fq.gz.keep.abundfilt.pe
SRR492066.pe.qc.fq.gz.keep.abundfilt.pe.keep
SRR492066.pe.qc.fq.gz.keep.abundfilt.se
SRR492066.pe.qc.fq.gz.keep.abundfilt.se.keep
```

by typing:

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

You may also want to remove the k-mer hash tables:

```
rm *.kh
```

Read stats

Try running:

```
/usr/local/share/khmer/sandbox/readstats.py *.kak.qc.fq.gz *.?e.qc.fq.gz
```

after a long wait, you'll see

```
---------------
861769600 bp / 8617696 seqs; 100.0 average length -- SRR492065.pe.qc.fq.gz
79586148 bp / 802158 seqs; 99.2 average length -- SRR492065.se.qc.fq.gz
531691400 bp / 5316914 seqs; 100.0 average length -- SRR492066.pe.qc.fq.gz
89903689 bp / 904157 seqs; 99.4 average length -- SRR492066.se.qc.fq.gz

173748898 bp / 1830478 seqs; 94.9 average length -- SRR492065.pe.kak.qc.fq.gz
8825611 bp / 92997 seqs; 94.9 average length -- SRR492065.se.kak.qc.fq.gz
52345833 bp / 550900 seqs; 95.0 average length -- SRR492066.pe.kak.qc.fq.gz
10280721 bp / 105478 seqs; 97.5 average length -- SRR492066.se.kak.qc.fq.gz

---------------
```

This shows you how many sequences were in the original QC files, and how many are left in the 'kak' files. Not bad – considerably more than 80% of the reads were eliminated in the kak!

Next: *3. Partitioning*

1.2.3 3. Partitioning

Note: Make sure you're running in screen!

Start with the QC'ed files from *2. Running digital normalization* or copy them into a working directory.

Simple partitioning

Partitioning is a rather complex process – nowhere near as nice and simple as digital normalization. However, we do have a simple script to run the basic stuff; if this script is too slow, or doesn't work well for big chunks of data, we might have remedies. But for the meantime, try the simple script:

```
/usr/local/share/khmer/scripts/do-partition.py -k 32 -x 1e9 --threads 4 kak *.kak.qc.fq.gz
```

This should take about 15 minutes, and will produce '.part' files. These are now FASTA files that contain partition annotations. For example, check out:

```
head SRR492065.pe.kak.qc.fq.gz.part
```

Extracting the partitions into groups

Generally there are *lots* of partitions, and for convenience sake we group them into group files that can be assembled in small chunks. To do this,

```
/usr/local/share/khmer/scripts/extract-partitions.py -X 100000 kak *.part
```

This will leave you with a bunch of 'kak.group*.fa', as well as a '.dist' file containing the distribution of partition sizes (how many sequences are in a given partition).

Here, the '-X' sets the number of sequences stuck into a group file. By default the -X parameter is 1 million, which would put all of the sequences into a single file for this data set.

Occasionally (OK, rather frequently) you'll find that almost all of your sequences coalesce into one partition, which we unaffectionately call the 'lump'. There are many possible reasons for this, and we have a series of increasingly large hammers that can be used on the lump.

For now, simply observe that:

```
tail kak.dist
```

reports that about 2/3 of the sequences are in a single partition:

```
1674746 1 112164 2539252
```

Reinflating partitions (optional)

At this point it's worth noting that the partitions are *normalized*, that is, diginormed. That makes it hard to use them for abundance calculations, and some assemblers prefer to have the original abundances in there.

So, ran you recover the abundances? Of course you can! However, you do have to combine all of the raw (unpartitioned) reads into a single file, because the script to reinflate the partitions takes only single file. Sorry :(.

```
gunzip -c /class/cbrown/data/SRR49206?.?e.qc.fq.gz > all.fq
python /usr/local/share/khmer/sandbox/sweep-reads3.py -x 1e8 kak.group*.fa all.fq
```

Next: `4-assemble`

CHAPTER

TWO

ADDITIONAL INFORMATION

Need help? Either post comments on the bottom of each page, OR sign up for the mailing list.

Have you used these protocols in a scientific publication? We'll have citation instructions up soon.

CHAPTER

THREE

FUNDING

khmer-protocols development has largely been supported by AFRI Competitive Grant no. 2010-65205-20361 from the USDA NIFA, and Award Number R25HG006243 from the National Institutes of Health, both to C. Titus Brown. We now have continuing support from the National Human Genome Research Institute of the National Institutes of Health under Award Number R01HG007513, also to C. Titus Brown.

CTB's work on the Eel Pond mRNAseq tutorial was enabled by his 2013 summer research work at the Marine Biological Laboratory, funded by the Burr and Susie Steinbach Award and the Laura and Arthur Colwin Endowed Summer Research Fellowship Fund

CHAPTER

FOUR

TODO:

- remove/transition stuff from the angus site.
- add sfg/stanford: http://sfg.stanford.edu/
- send to biostar-ninjas