labibi Documentation

Release 5.0

C. Titus Brown

Jun 02, 2017

Contents

This is the schedule for the 2016 MSU NGS course. All course communications will be organized around the Slack channel.

This workshop has a *Workshop Code of Conduct*, do read it!

Download all of these materials or visit the GitHub repository.

Meal Times: Breakfast 7:45-8:45, Lunch 12-1, Dinner 6-7 (Unless noted below). See HERE for the menu and other info.

Day	Schedule
Monday 8/8	• 1:30pm lecture: Welcome! (Meg and Matt) • 2:30pm: Getting Started with AWS *Getting started with Amazon EC2* (Matt) • 4pm: Intro to Linux setup and MacManes_UNIX and Online Tutorial (Matt) • 7pm tutorial, background PDF BLAST • *Running command-line BLAST* (Meg) • 8pm: Instructor and TA Research presentations, socialize
Tuesday 8/9	• Tutorial *Day 2 – Intro to Linux & Quality control* • 9:15am lecture: Sequencing basics (Matt) • 11:00am Assessment (Bob Drost) • 1:15pm tutorial, background PDF and MacManesTrimming (Matt) • 3:00pm Opionated guide to making a computational notebook. (Matt and other people) • Evening *firepit social*
Wed 8/10	• 9:15am lecture, mapping and variant calling lecture (Meg) • 10:00am practical Variant Calling, *Variant calling* (Meg) • 1:15pm LinuxBrew slides (Shaun) • 1:45pm practical, Installing Linuxbrew on AWS Ubuntu 14 linuxbrew_install (Torst) • 2:15pm tutorial, BASH for genomics, GenomicsShell (Amanda) • 7:00pm Mapping Quest MacManes_MapQuest (All) • 8:30pm student presentations (Part 1 ~3 minutes each)
Thursday 8/11	• 9:15am lecture, Intro to genome assembly (Torst) • 10:15am Genome assembly practical (Shaun) • 1:15pm Genome assembly practical (continued) • 2:15pm lecture, De Bruijn graph assembly (Shaun) • 7:15pm Assembly challenge MacManes_AssemblyQuest (Matt)
Friday 8/12	• 9:15am Debrief of Assembly Quest marathon (Matt) • 9:30am lecture, Bacterial genome annotation (Torst) • 10:00am practical Prokka prokka_genome_annotation (Torst) • 11:00am practical Species identification with Kraken kraken_species_identification (Torst) • 1:15pm tutorial, Intro to R Rintro (Amanda) • 6:00pm Thai food take out at Mcrary Hall • 7:30pm Mars Rover Challenge! (Shaun)
	• 8:30pm Finish up student presentations
Saturday 8/13	• 9:15am Population genetics lecture and practical (Sonal)

CHAPTER 1

Dramatis personae

Instructors:

- Meg Staton @HardwoodGenomic.
- Matt MacManes @macmanes.
- Torsten Seemann @torstenseemann.
- Shaun Jackman @sjackman.
- Rob Patro @nomad421.
- Ian Dworkin @IanDworkin.
- Sonal Singhal
- Amanda Charbonneau @procrastinomics.

TAs:

- Lisa Cohen @monsterbashseq.
- Will Pitchers @steeljawpanda.
- Ming Chen

She Who Drives Many Places:

- Kate MacManes

CHAPTER 2

Genome Assembly References

- Ben Langmead Resources
- How do we assemble genomes (videos)

CHAPTER 3

Papers and References

Books

- Bioinformatics Data Skills
- Practical Computing for Biologists

  These books, especially the 1st, are highly recommended book for people looking for a systematic presentation on shell scripting, programming, UNIX, etc.

RNAseq

- Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks, Trapnell et al., Nat. Protocols.

  One paper that outlines a pipeline with the tophat, cufflinks, cuffdiffs and some associated R scripts.
- Statistical design and analysis of RNA sequencing data., Auer and Doerge, Genetics, 2010.
- A comprehensive comparison of RNA-Seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: a case study in Saccharomyces cerevisiae. Nookaew et al., Nucleic Acids Res. 2012.
- Challenges and strategies in transcriptome assembly and differential gene expression quantification. A comprehensive in silico assessment of RNA-seq experiments Vijay et al., 2012.
- Computational methods for transcriptome annotation and quantification using RNA-seq, Garber et al., Nat. Methods, 2011.
- Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments., Bullard et al., 2010.
- A comparison of methods for differential expression analysis of RNA-seq data, Soneson and Delorenzi, BMC Bioinformatics, 2013.

- Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples., Wagner et al., Theory Biosci, 2012. Also see this blog post explaining the paper in detail.

Computing and Data

- A Quick Guide to Organizing Computational Biology Projects, Noble, PLoS Comp Biology, 2009.
- Willingness to Share Research Data Is Related to the Strength of the Evidence and the Quality of Reporting of Statistical Results, Wicherts et al., PLoS One, 2011.
- Got replicability?, McCullough, Economics in Practice, 2007.

Also see this great pair of blog posts on organizing projects and research workflow.

CHAPTER 4

Links

Humor

- Data Sharing and Management Snafu in 3 Short Acts

Resources

- Biostar

  A high quality question & answer Web site.
- SEQanswers

  A discussion and information site for next-generation sequencing.
- Software Carpentry lessons

  A large number of open and reusable tutorials on the shell, programming, version control, etc.

Blogs

- http://www.genomesunzipped.org/

  Genomes Unzipped.
- http://ivory.idyll.org/blog/

  Titus's blog.
- http://bcbio.wordpress.com/

  Blue Collar Bioinformatics

- http://massgenomics.org/

  Mass Genomics
- http://blog.nextgenetics.net/

  Next Genetics
- http://gettinggeneticsdone.blogspot.com/

  Getting Genetics Done
- http://omicsomics.blogspot.com/

  Omics! Omics!
- http://lab.loman.net/

  Nick Loman's lab notebook
- http://TheGenomeFactory.blogspot.com/

  The Genome Factory (Torsten Seemann)

CHAPTER 5

Material from previous years

Day 1 - Getting started with Amazon

We're going to start by getting you set up on Amazon Web Services. For the duration of the course, we'll be running analyses on computers we rent from Amazon; this has a number of benefits that we'll discuss in the lecture.

Getting started with Amazon EC2

Summary:

- Go to http://aws.amazon.com/, log in, then "EC2" (upper left);
- Select "Launch instance";
- Select "Ubuntu 14.04" from the list;
- Select "m3.xlarge" from the list (towards bottom of "General purpose");
- Click "Review and launch"
- Select "Launch";
- If your first time through, create a key pair; otherwise select existing;
- Click "launch instance"

Details!

Start up an EC2 instance

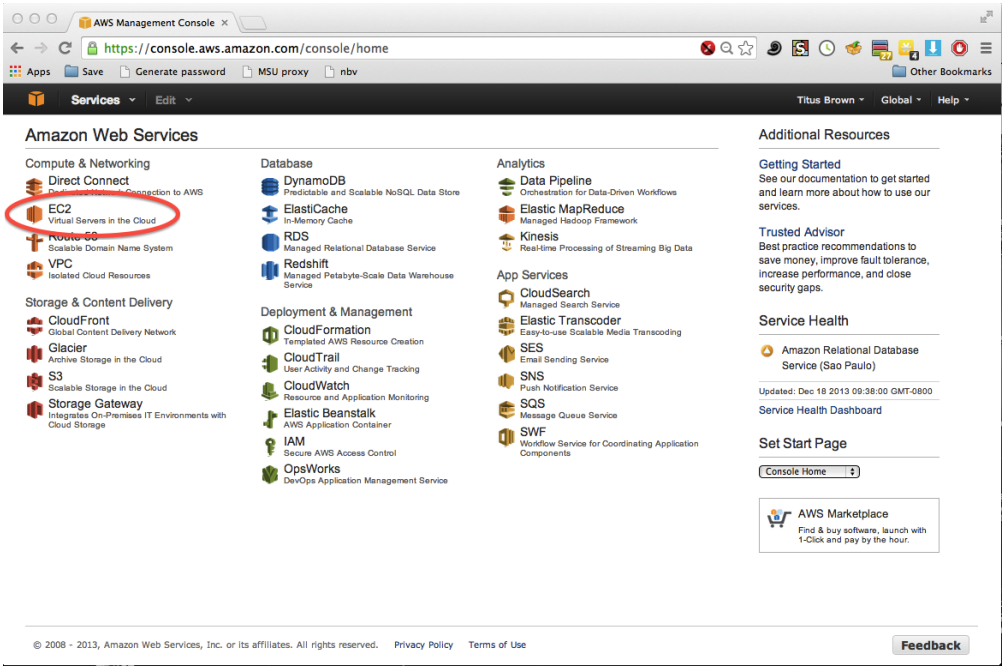
Log in

Go to 'https://aws.amazon.com' in a Web browser.

Select 'My Account/Console' menu option 'AWS Management Console."

Log in with your username & password.

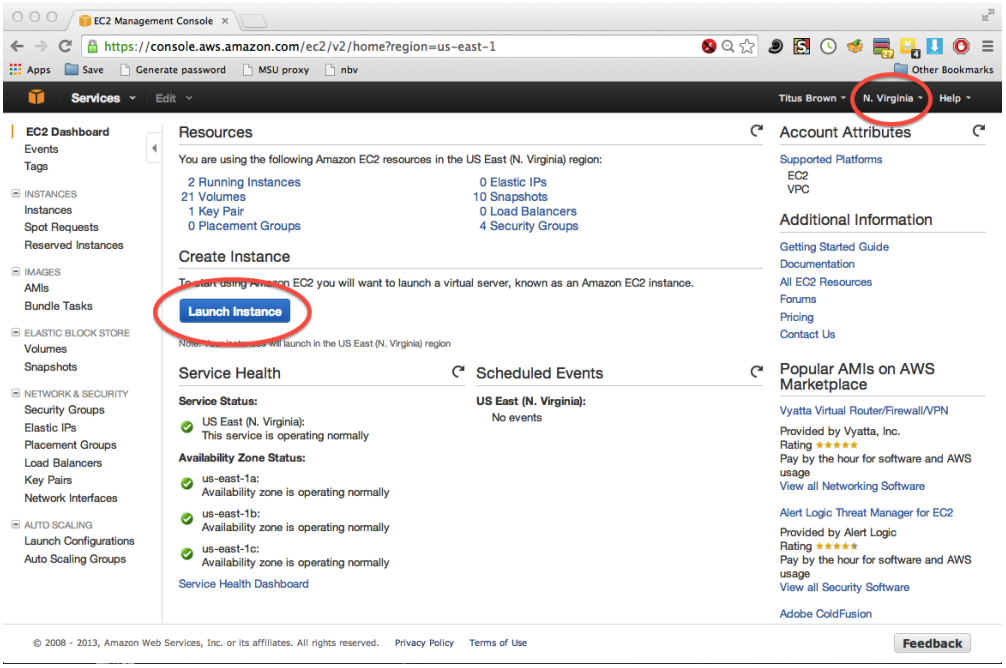
Click on EC2 (upper left).



Select your zone

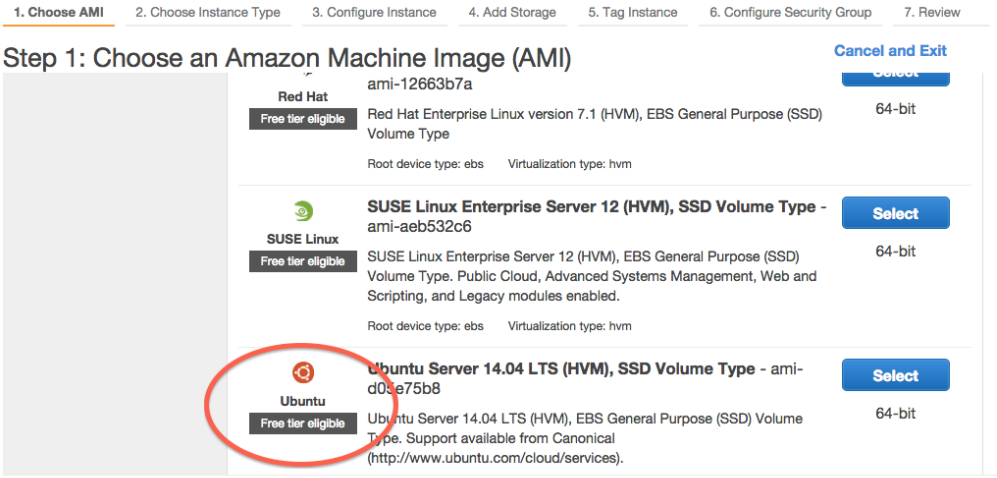
Many of the resources that we use are hosted by Amazon on the East coast. Make sure that your dashboard has 'N. Virginia' on the upper right.

Then click on Launch Instance.



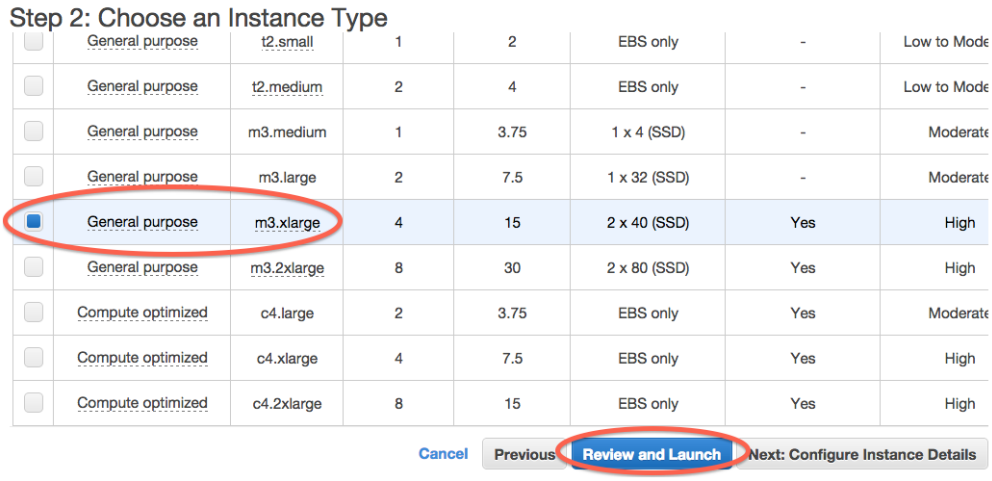
Select the machine operating system to boot

Find the "Ubuntu Server 14.04" image in the first list to show up.



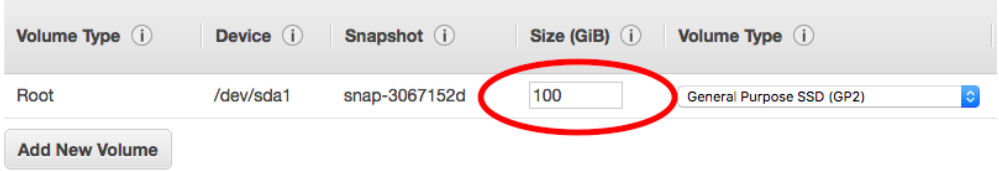
Choose the machine size

Select 'General purpose', 'c4.2xlarge', and then 'Review and Launch'.



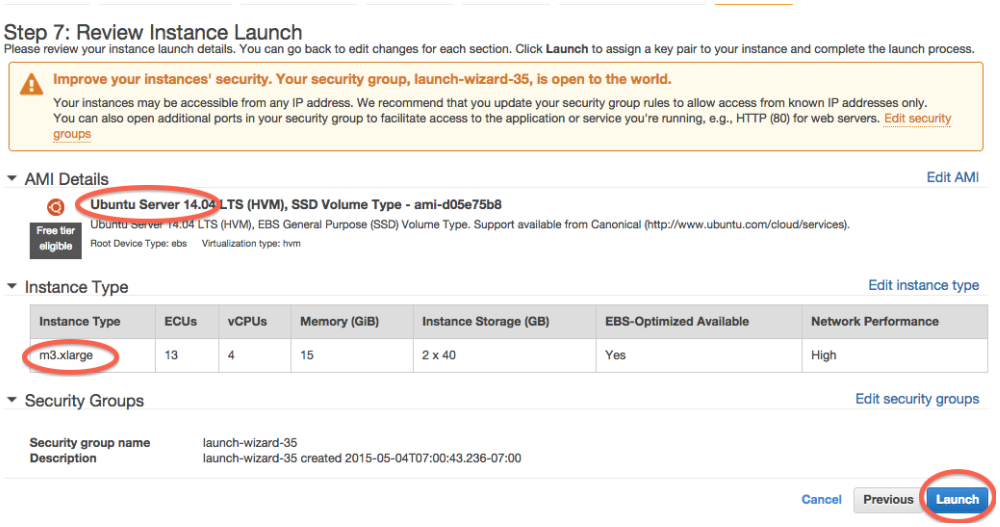
Change the size of the hard drive

Make the size of the hard drive to be 100GB (or whatever is appropriate for the lesson).



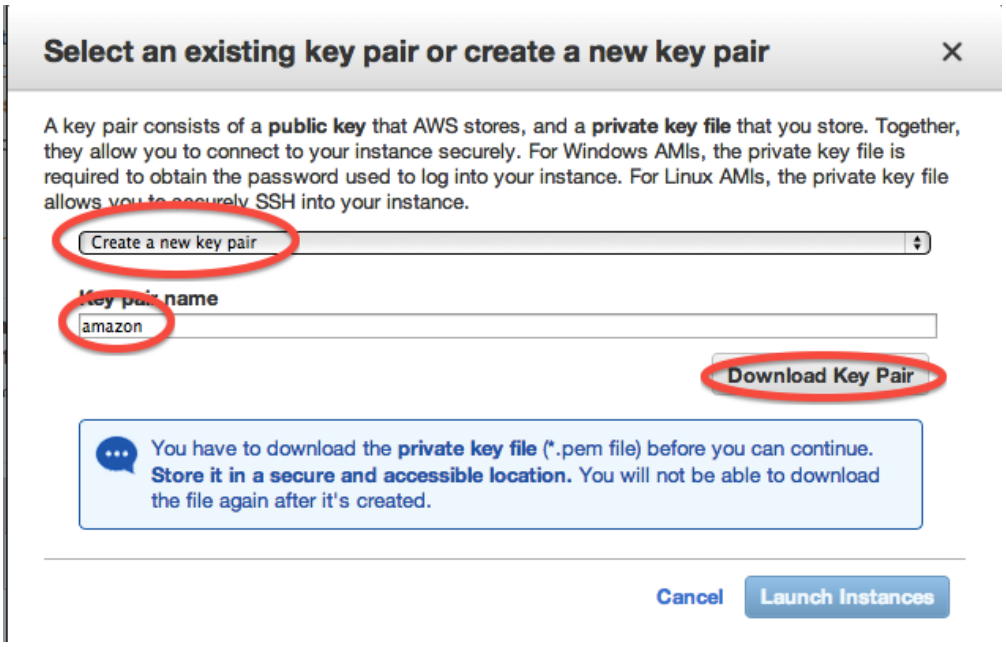
Confirm and launch

Review the details (ignore the warnings!) and click on Launch.



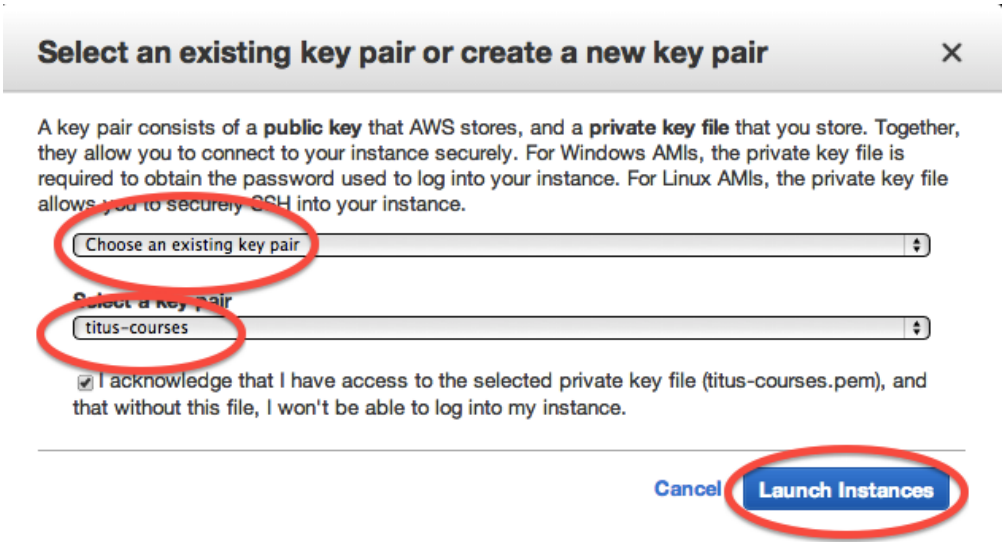
(First time through) generate a new key pair

If you don't have any key pairs, enter a key pair name and then download a key pair. Then click Launch Instance.

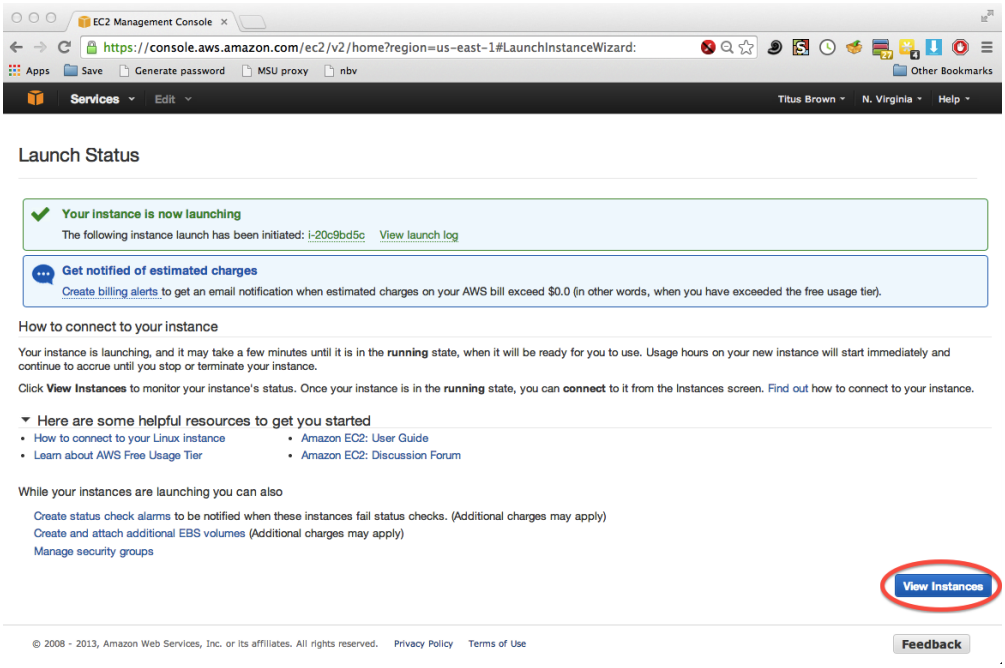


(Next times through) select an existing key pair

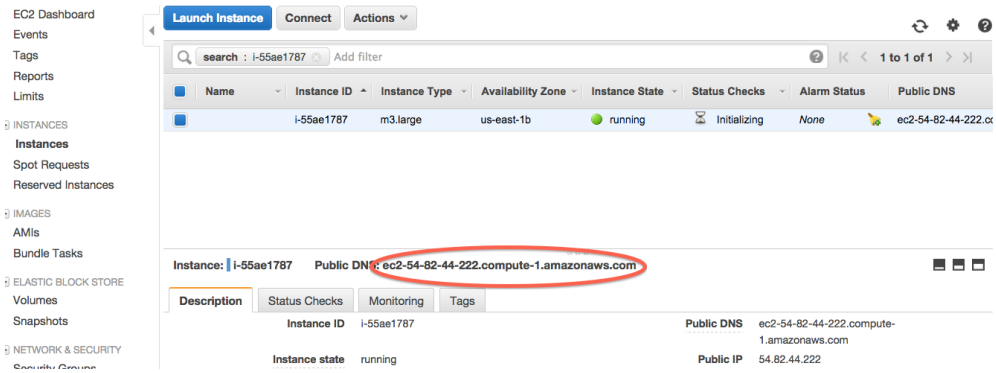
Select a key pair and click 'Launch'.



Click on View Instances



Select the public DNS name for later use



Next steps

Logging into your new instance "in the cloud" (Mac version) or log-in-with-ssh-win

Logging into your new instance "in the cloud" (Windows version)

OK, so you've created a running computer. How do you get to it?

The main thing you'll need is the network name of your new computer. To retrieve this, go to the instance view and click on the instance, and find the "Public DNS". This is the public name of your computer on the Internet.

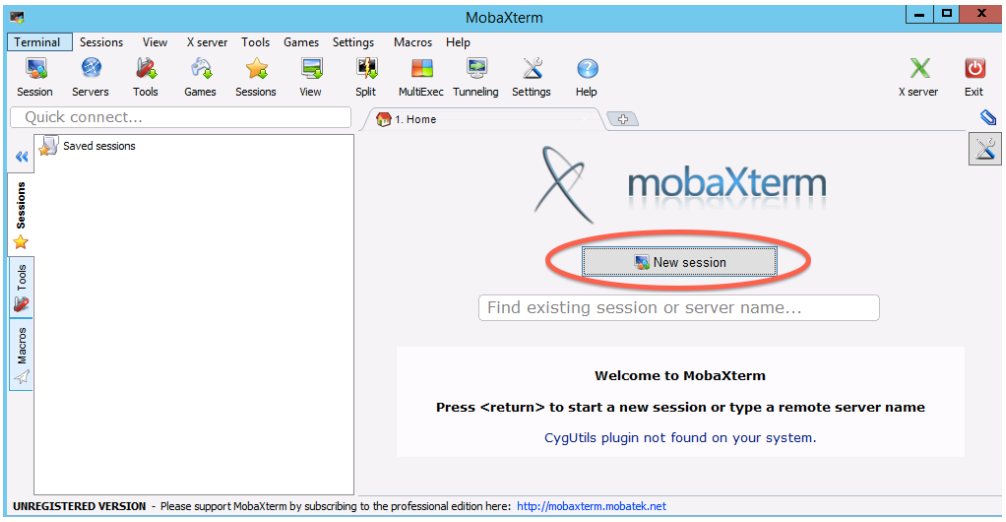
Copy this name, and connect to that computer with ssh under the username 'ubuntu', as follows.

—

Install mobaxterm

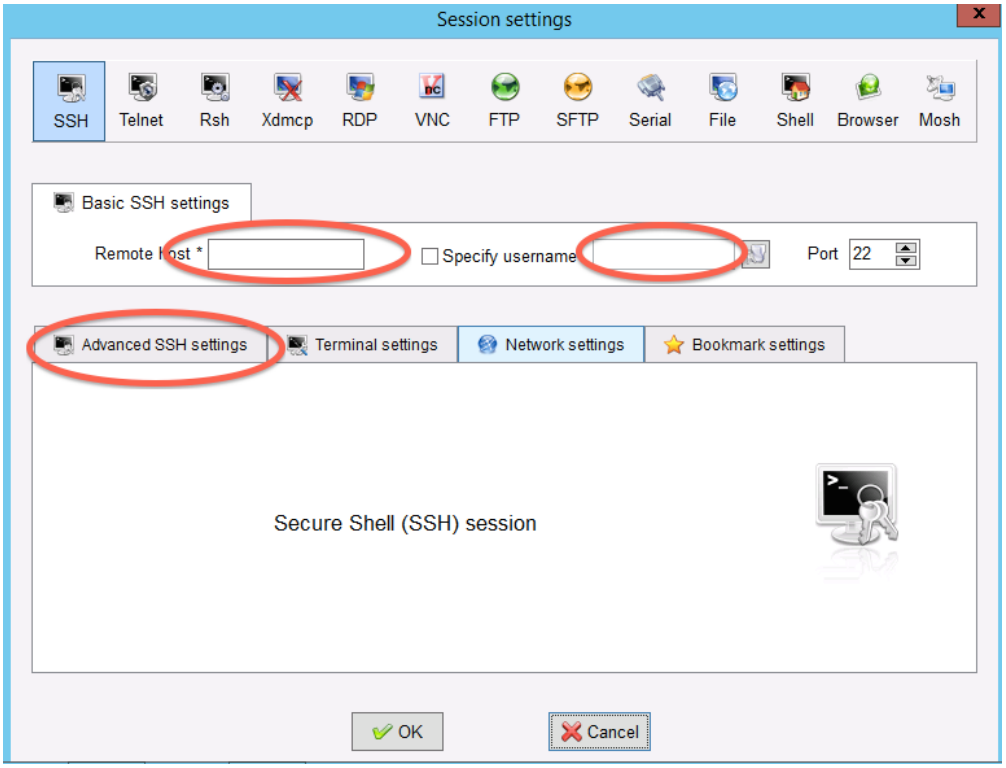
First, download mobaxterm and run it.

Start a new session



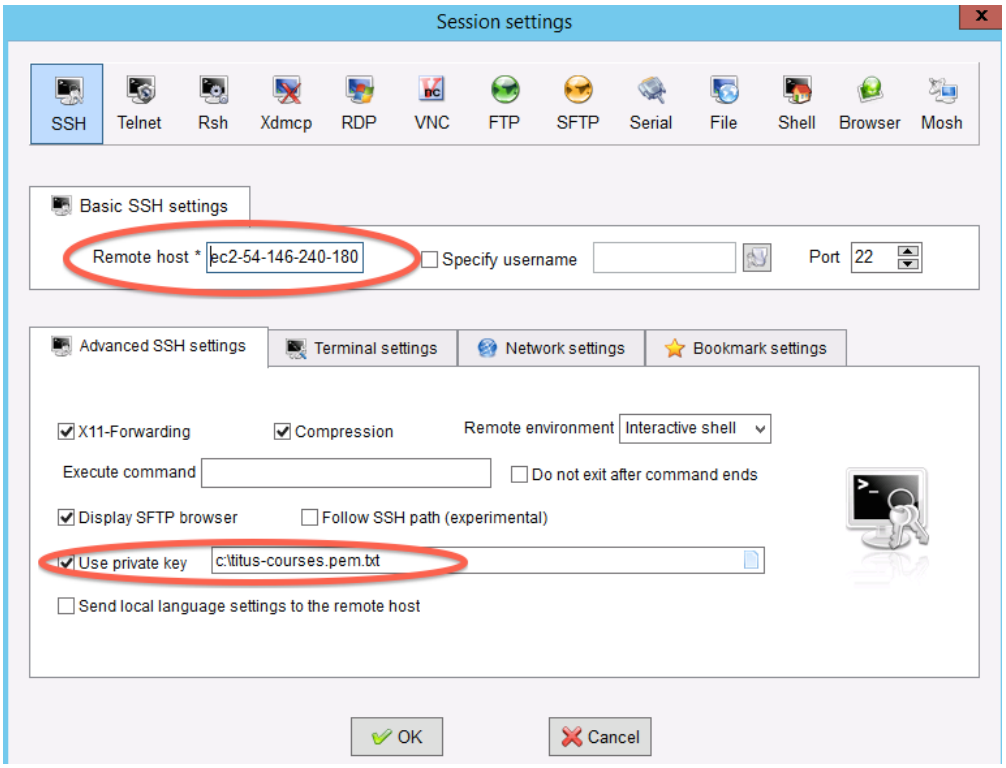
Fill in session settings

Put in your hostname (should be `ec2-XXX-YYY-ZZZ-AAA.compute-1.amazon.aws.com`), select 'specify username', and enter 'ubuntu'.



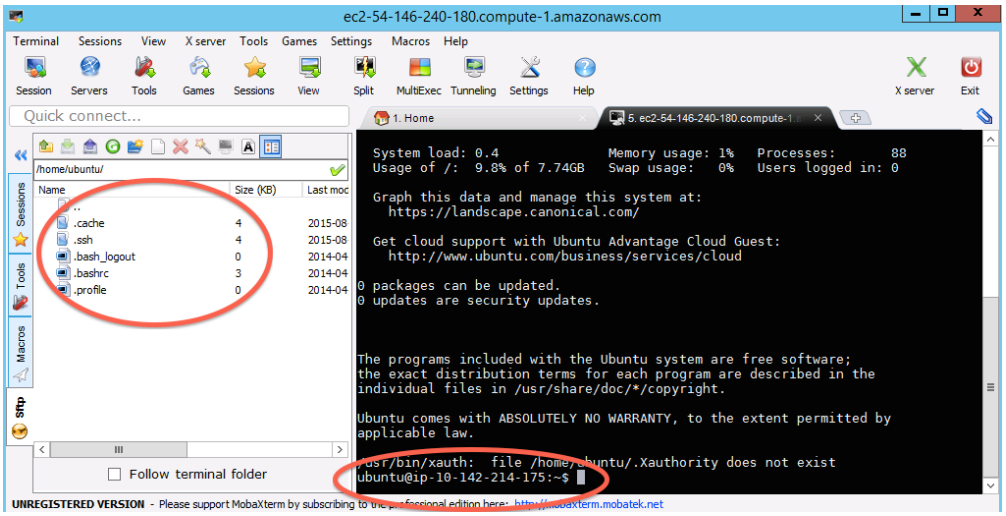
Specify the session key

Copy the downloaded .pem file onto your primary hard disk (generally C:) and the put in the full path to it.



Click OK

Victory! (?)



Return to index: *Getting started with Amazon EC2*

Logging into your new instance “in the cloud” (Mac version)

OK, so you’ve created a running computer. How do you get to it?

The main thing you'll need is the network name of your new computer. To retrieve this, go to the instance view and click on the instance, and find the "Public DNS". This is the public name of your computer on the Internet.

Copy this name, and connect to that computer with ssh under the username 'ubuntu', as follows.

First, find your private key file; it's the .pem file you downloaded when starting up your EC2 instance. It should be in your Downloads folder. Move it onto your desktop and rename it to 'amazon.pem'.

Next, start Terminal (in Applications... Utilities...) and type:

```
chmod og-rwx ~/Desktop/amazon.pem
```

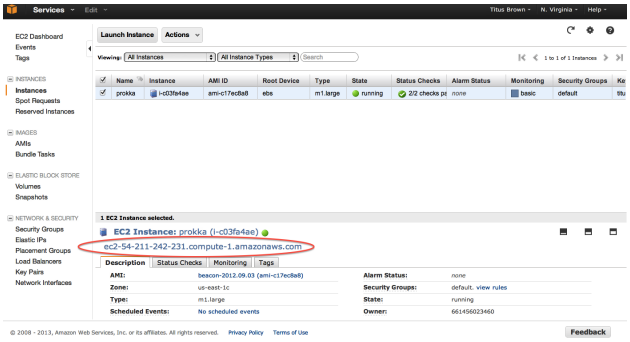
to set the permissions on the private key file to "closed to all evildoers".

Then type:

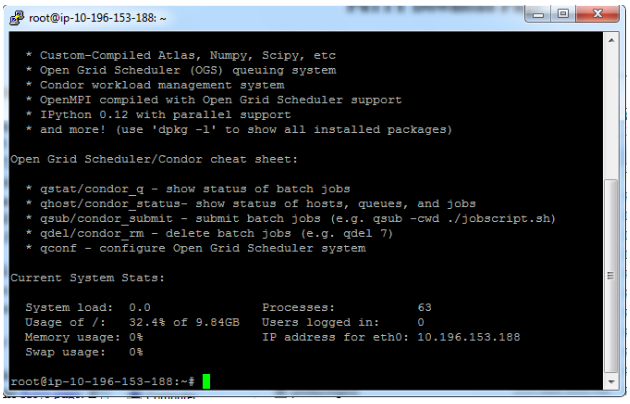
```
ssh -i ~/Desktop/amazon.pem ubuntu@ec2-???-???-???-???.compute-1.amazonaws.com
```

Here, you're logging in as user 'ubuntu' to the machine 'ec2-174-129-122-189.compute-1.amazonaws.com' using the authentication key located in 'amazon.pem' on your Desktop.

Note, you have to replace the stuff after the '@' sign with the name of the host; see the red circle in:



At the end you should see text and a prompt that look like this:

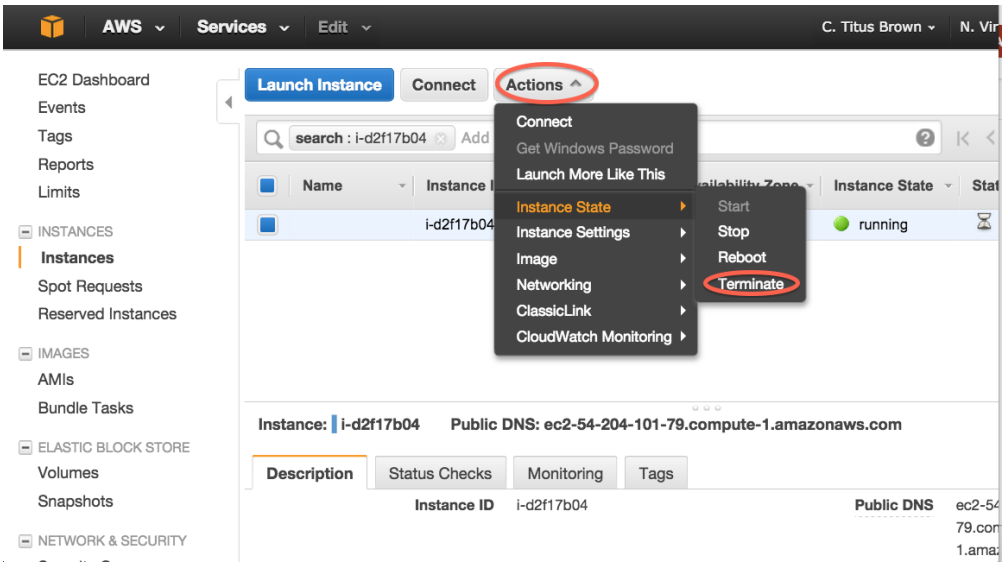


Return to index: *Getting started with Amazon EC2*

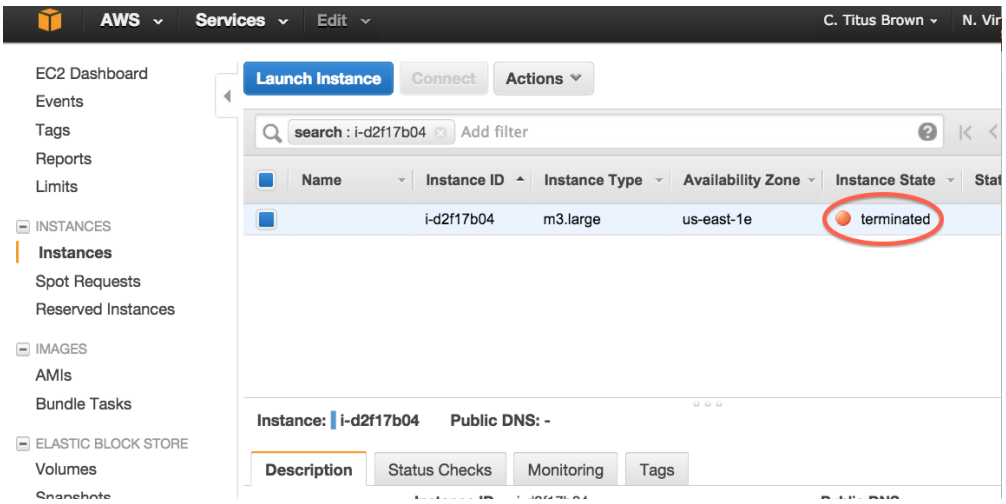
Terminating your instance

Be sure to terminate your instance(s) after transferring off any data that you want to keep!

To terminate your instance, select the instance you want to terminate, and then go to the 'Actions' menu and select 'Instance actions', 'terminate':



Wait a minute or two to be sure that the instance state changes to "terminated":



A final checklist:

- You have a green EC2 instance!
- You used ubuntu 14.04;
- You're in US East (Virginia);
- You didn't start a micro instance (m3.xlarge, or bigger);

Amazon Web Services reference material

Instance types

Instance costs

Running command-line BLAST

The goal of this tutorial is to run you through a demonstration of the command line, which you may not have seen or used much before.

Start up an m1.xlarge Amazon EC2 instance.

All of the commands below can copy/pasted.

Install software

Copy and paste the following commands

```
sudo apt-get update && sudo apt-get -y install python ncbi-blast+
```

This updates the software list and installs the Python programming language and NCBI BLAST+.

Get Data

Grab some data to play with. Grab some cow and human RefSeq proteins:

```
wget ftp://ftp.ncbi.nih.gov/refseq/B_taurus/mRNA_Prot/cow.1.protein.faa.gz
wget ftp://ftp.ncbi.nih.gov/refseq/H_sapiens/mRNA_Prot/human.1.protein.faa.gz
```

This is only the first part of the human and cow protein files - there are 24 files total for human.

The database files are both gzipped, so lets unzip them

```
gunzip *gz
ls
```

Take a look at the head of each file:

```
head cow.1.protein.faa
head human.1.protein.faa
```

These are protein sequences in FASTA format. FASTA format is something many of you have probably seen in one form or another – it's pretty ubiquitous. It's just a text file, containing records; each record starts with a line beginning with a '>', and then contains one or more lines of sequence text.

Note that the files are in fasta format, even though they end if ".faa" instead of the usual ".fasta". This NCBI's way of denoting that this is a fasta file with amino acids instead of nucleotides.

How many sequences are in each one?

```
grep -c '^>' cow.1.protein.faa
grep -c '^>' human.1.protein.faa
```

This grep command uses the c flag, which reports a count of lines with match to the pattern. In this case, the pattern is a regular expression, meaning match only lines that begin with a >.

This is a bit too big, lets take a smaller set for practice. Lets take the first two sequences of the cow proteins, which we can see are on the first 6 lines

```
head -6 cow.1.protein.faa > cow.small.faa
```

BLAST

Now we can blast these two cow sequences against the set of human sequences. First, we need to tell blast about our database. BLAST needs to do some pre-work on the database file prior to searching. This helps to make the software work a lot faster. Because you installed your own version of the sotware, you need to tell the shell where the software is located. Use the full path and the makeblastdb command:

```
makeblastdb -in human.1.protein.faa -dbtype prot
ls
```

Note that this makes a lot of extra files, with the same name as the database plus new extensions (.pin, .psq, etc). To make blast work, these files, called index files, must be in the same directory as the fasta file.

Now we can run the blast job. We will use blastp, which is appropriate for protein to protein comparisons.

```
blastp -query cow.small.faa -db human.1.protein.faa
```

This gives us a lot of information on the terminal screen. But this is difficult to save and use later - Blast also gives the option of saving the text to a file.

```
    blastp -query cow.small.faa -db human.1.protein.faa -out cow_vs_human_blast_
↪results.txt
ls
```

Take a look at the results using less. Note that there can be more than one match between the query and the same subject. These are referred to as high-scoring segment pairs (HSPs).

```
less cow_vs_human_blast_results.txt
```

So how do you know about all the options, such as the flag to create an output file? Lets also take a look at the help pages. Unfortunately there are no man pages (those are usually reserved for shell commands, but some software authors will provide them as well), but there is a text help output

```
blastp -help
```

To scroll through slowly

```
blastp -help | less
```

To quit the less screen, press the q key.

Parameters of interest include the -evalue (Default is 10?!?) and the -outfmt

Lets filter for more statistically significant matches with a different output format:

```
blastp \
-query cow.small.faa \
-db human.1.protein.faa \
-out cow_vs_human_blast_results.tab \
-evalue 1e-5 \
-outfmt 7
```

I broke the long single command into many lines with by “escaping” the newline. That forward slash tells the command line “Wait, I’m not done yet!”. So it waits for the next line of the command before executing.

Check out the results with less.

Lets try a medium sized data set next

```
head -199 cow.1.protein.faa > cow.medium.faa
```

What size is this db?

```
grep -c '^>' cow.medium.faa
```

Lets run the blast again, but this time lets return only the best hit for each query.

```
blastp \
-query cow.medium.faa \
-db human.1.protein.faa \
-out cow_vs_human_blast_results.tab \
-evalue 1e-5 \
-outfmt 6 \
-max_target_seqs 1
```

Summary

Review:

- command line programs such as blast use flags to get information about how and what to do
- blast options can be found by typing *blastp -help*
- break a command up over many lines by using ‘‘ to “escape” the new line

Reminder: shut down your instance!

Day 2 – Intro to Linux & Quality control

Before following the procedures below, go through the process of starting up an ec2 instance and logging in – see *Day 1 - Getting started with Amazon* for details.

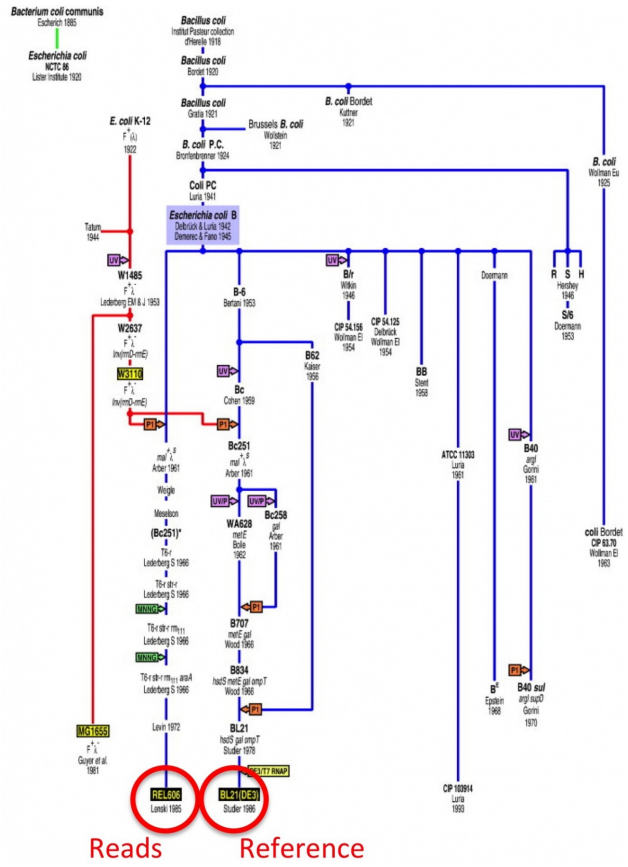
The lecture will start at 9:15, the first tutorial (*Running command-line BLAST*) will start at 10:30, and the second tutorial will start at 1:30.

Variant calling

The goal of this tutorial is to show you the basics of variant calling using Samtools.

We’re going to be looking at variation in laboratory grown strains of Escherichia coli. We have reads from B strain REL606 and we’ll be mapping it to a reference genome from BL21(DE3). This is a different lab strain, and there’s an interesting paper where they trace the origin and transfer of all the different E. coli strains between scientisits through the decades.

Citation: Tracing Ancestors and Relatives of Escherichia coli B, and the Derivation of B Strains REL606 and BL21(DE3) Journal of Molecular Biology, Volume 394, Issue 4, 11 December 2009, Pages 634–643



Booting an Amazon AMI

Start up an Amazon computer (large or xlarge) with an storage of 100Gb.

Install software

Log into your instance. Install ruby and git, then install linuxbrew.

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install ruby git
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/master/
↪install)"
export PATH="/home/ubuntu/.linuxbrew/bin:$PATH"
export MANPATH="/home/ubuntu/.linuxbrew/share/man:$MANPATH"
export INFOPATH="/home/ubuntu/.linuxbrew/share/info:$INFOPATH"
brew tap homebrew/science
```

Now we can install anything available from linuxbrew science

```
brew install samtools
brew install zlib
brew install bcftools
brew install bwa
```

See what is installed:

```
brew list
```

Download data

Links to learn more about the data:

- Reference Genome
- Reads

Download the reference genome and the resequencing reads

```
curl "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=NC_
↪012971&rettype=fasta&retmode=text" > Ecoli_BL21.fasta
curl -O ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR098/SRR098038/SRR098038.fastq.gz
```

Note, this last URL is the "Fastq files (FTP)" link from the EBI page. Its compressed, lets decompress

```
gunzip SRR098038.fastq.gz
```

Just in case EBI is down , you can also get reads this way

```
curl -O ftp://ftp.ddbj.nig.ac.jp/ddbj_database/dra/fastq/SRA026/SRA026813/SRX040675/
↪SRR098038.fastq.bz2
```

Rename the reference

The reference is named something really long and complicated. Check it out

```
head Ecoli_BL21.fasta
```

Lets shorten that for fewer headaches. Use nano to make the header look like this:

```
>NC_012971.2
```

Read mapping

Create the BWA index

```
bwa index Ecoli_BL21.fasta
```

Now, do the mapping of the raw reads to the reference genome

```
bwa aln Ecoli_BL21.fasta SRR098038.fastq > SRR098038.sai
```

Make a SAM file (this would be done with 'sampe' if these were paired-end reads)

```
bwa samse Ecoli_BL21.fasta SRR098038.sai SRR098038.fastq > SRR098038.sam
```

A sam file contains all of the information about where each read hits on the reference. Links for more info:

- SAM the file format

- Samtools the software

Next, index the reference genome with samtools

```
samtools faidx Ecoli_BL21.fasta
```

Convert the SAM into a BAM file

```
samtools view -bS SRR098038.sam > SRR098038.bam
```

Sort the BAM file

```
samtools sort SRR098038.bam > SRR098038.sorted.bam
```

And index the sorted BAM file

```
samtools index SRR098038.sorted.bam
```

Visualizing alignments

At this point you can visualize with samtools tview. Other visualization software: * Tablet. * IGV

'samtools tview' is a text interface that you use from the command line; run it like so

```
samtools tview SRR098038.sorted.bam Ecoli_BL21.fasta
```

The '.'s are places where the reads align perfectly in the forward direction, and the ','s are places where the reads align perfectly in the reverse direction. Mismatches are indicated as A, T, C, G, etc.

You can scroll around using left and right arrows; to go to a specific coordinate, use 'g' and then type in the contig name and the position. For example, type 'g' and then 'NC_012971.2:553093<ENTER>' to go to position 553093 in the BAM file. (This name is taken from the fasta reference file, you could change to something more reasonable).

Use 'q' to quit.

Statistics of alignments

This command

```
samtools view -c -f 4 SRR098038.bam
```

will count how many reads DID NOT align to the reference (214518).

This command

```
samtools view -c -F 4 SRR098038.bam
```

will count how many reads DID align to the reference (6832113).

And this command

```
wc -l SRR098038.fastq
```

will tell you how many lines there are in the FASTQ file (28186524). Reminder: there are four lines for each sequence.

There is another package, Picard Tools, that can give you more in depth information. Lets install with linuxbrew

```
brew install picard-tools
```

And use the particular tool CollectAlignmentSummaryMetrics

```
picard CollectAlignmentSummaryMetrics R=Ecoli_BL21.fasta I=SRR098038.sorted.bam␣
↪O=statistics.txt
```

More picard tools stuff here

You can see the output with cat

```
cat statistics.txt
```

The definitions of all the columns in this file.

Calling SNPs

You can use samtools and bcftools to call SNPs. They have great documentation of a standard workflow for calling SNPs, you should read more about it. We're going to do a simplified and updated version here.

Start with mpileup and pipe the results to bcftools

```
samtools mpileup -uf Ecoli_BL21.fasta SRR098038.sorted.bam | bcftools call -vmO v -o␣
↪SRR098038.vcf --ploidy 1 --threads 2
```

You can check out the VCF file by using 'tail' to look at the bottom

```
tail SRR098038.vcf
```

Each variant call line consists of the chromosome name (for E. coli REL606, there's only one chromosome); the position within the reference; an ID (here always '.'); the reference call; the variant call; and a bunch of additional information about the variant.

The information at the end can be very useful but difficult to interpret. One way to quickly look up the label shorthand is to grep

```
 grep '<ID=VDB' SRR098038.vcf
    grep '<ID=AC' SRR098038.vcf



samtools tview SRR098038.sorted.bam Ecoli_BL21.fasta
```

You can use 'samtools tview' again and then type (for example) 'g' 'rel606:4616538' to go visit one of the positions. The format for the address to go to with 'g' is 'chr:position'.

```
NC_012971.2:4558366
```

You can read more about the VCF file format here.

Using IGV for Visualization

Installing IGV requires registration and some patience. IGV Link.

To open your alignments, you'll need three files on your local computer: `Ecoli_BL21.fasta`, `SRR098038.sorted.bam`, and `SRR098038.sorted.bam.bai`. You can copy them over using scp (secure copy), for example. You will do this from a terminal on your computer that is NOT connected to amazon.

```
scp -i ~/Downloads/???.pem ubuntu@???:/home/ubuntu/Ecoli_BL21.fasta ~/Downloads
scp -i ~/Downloads/???.pem ubuntu@???:/home/ubuntu/SRR098038.sorted.bam ~/Downloads
scp -i ~/Downloads/???.pem ubuntu@???:/home/ubuntu/SRR098038.sorted.bam.bai ~/
↪Downloads
```

To add the gene annotation, get this file as well

```
curl ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/Bacteria/Escherichia_coli_
↪BL21_DE3__uid161947/NC_012971.gff
```

Student Exercise

You are eager to use some E. coli reads from a collaborator, which you can download here

```
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR201/004/SRR2014554/SRR2014554_1.fastq.gz
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR201/004/SRR2014554/SRR2014554_2.fastq.gz
```

You need to quality trim them, map them to the E. coli reference, and call SNPs. How far can you get?

Interval Analysis and Visualization

The results generate below are based on a question posed by a participant in the course. She wanted to know how well contigs of an unfinished genomic build of and ecoli strain match the common (K-12 strain MG1655) genomic build.

Download the results from:

http://apollo.huck.psu.edu/data/ms115.zip

How did we get the results in the file above? A short description follows:

Data collection

The partial genomic build is located at:

http://www.ncbi.nlm.nih.gov/genome/167?genome_assembly_id=161608

From this we downloaded the summary file `code/ADTL01.txt` that happens to be a tab delimited file that lists accession numbers. We then wrote a very simple program `code/getdata.py` to parse the list of accessions and download the data like so

```
# requires BioPython
from Bio import Entrez
Entrez.email = "A.N.Other@example.com"
stream = file("ADTL01.txt")
stream.next()

for line in stream:
    elems = line.strip().split()
    val = elems[1]
    handle = Entrez.efetch(db="nucleotide", id=val, rettype="fasta", retmode="text")
    fp = file("data/%s.fa" % val, 'wt')
```

```
    fp.write(handle.read())
    fp.close()
```

Finally we merged all data with:

```
cat *.fa > MS115.fa
```

Then we went hunting for the EColi genome, we found it here:

http://www.genome.wisc.edu/sequencing/updating.htm

Turns out that this site only distributes a GBK (Genbank file). We now need to extract the information from the GBK file to FASTA (genome) and GFF (genomic feature) file. For this we need to install the ReadSeq program:

http://iubio.bio.indiana.edu/soft/molbio/readseq/java/

Once we have this we typed:

```
# GBK to GFF format
java -jar readseq.jar -inform=2 -f 24 U00096.gbk

# GBK to FASTA
java -jar readseq.jar -inform=2 -f 24 U00096.gbk
```

This will create the files U00096.gbk.fasta and U00096.gbk.gff

Now lets map the ms115.fa contigs to the U00096.fa reference:

```
bwa index U00096.fa
bwa mem U00096.fa ms115.fa | samtools view -bS - | samtools sort - U00096
```

will produce the U00096.bam file. We have converted the U00096.bam to BED format via the:

```
bedtools bamtobed -i  U00096.bam  > U00096.bed
```

Visualizing the data

Download and run IGV

http://www.broadinstitute.org/igv/

Create a custom genome via *Genomes -> Create .genome* options

We will now visualize the BAM, GFF and BED files and discuss the various aspects of it.

Running bedtools

Install bedtools:

```
sudo apt-get bedtools
```

This works best if you store your files in Dropbox, that way you can edit the file on your computer then load them up on your IGV instance.

Understanding the SAM format

Log into your instance, create a new directory, navigate to that directory:

```
cd /mnt
    mkdir sam
    cd sam

    # Get the makefile.
    wget https://raw.githubusercontent.com/ngs-docs/angus/2014/files/Makefile-
↪samtools -O Makefile
```

A series of exercises will show what the SAM format is and how it changes when the query sequence is altered and how that reflects in the output.

Also, for the speed of result generation here is a one liner to generate a bamfile:

```
# One line bamfile generation.
bwa mem index/sc.fa query.fa | samtools view -bS - | samtools sort - results
```

This will produce the `results.bam` output.

Control Flow and loops in R

Control Flow

The standard if else

```
p.test <- function(p) {
    if (p <= 0.05)
        print("yeah!!!!") else if (p >= 0.9)
        print("high!!!!") else print("somewhere in the middle")
}
```

Now pick a number and put it in `p.test`

```
p.test(0.5)
```

```
## [1] "somewhere in the middle"
```

ifelse()

A better and vectorized way of doing this is `ifelse(test, yes, no)` function. ifelse() is far more useful as it is vectorized.

```
p.test.2 <- function(p) {
    ifelse(p <= 0.05, print("yippee"), print("bummer, man"))
}
```

Test this with the following sequence. See what happens if you use `if` vs. `ifelse()`.

```
x <- runif(10, 0, 1)
x
```

```
##  [1] 0.27332 0.14155 0.89000 0.07041 0.79419 0.25013 0.02324 0.86766
##  [9] 0.41114 0.56165
```

Now try it with `p.test()` (uses `if`).

```
p.test(x)
```

```
## Warning: the condition has length > 1 and only the first element will be used
## Warning: the condition has length > 1 and only the first element will be used
```

```
## [1] "somewhere in the middle"
```

Now try it with `p.test.2()`

```
p.test.2(x)
```

```
## [1] "yippee"
## [1] "bummer, man"
```

```
##  [1] "bummer, man" "bummer, man" "bummer, man" "bummer, man" "bummer, man"
##  [6] "bummer, man" "yippee"      "bummer, man" "bummer, man" "bummer, man"
```

Other vectorized ways of control flow.

There are many times that you may think you need to use an if with (iterating with a for loop... see below), or ifelse, but there may be far better ways.

For instance, say you are doing some simulations for a power analysis, and you want to know how often your simulation gives you a p-value less than 0.05.

```
p.1000 <- runif(n = 1000, min = 0, max = 1)
```

The line above generates 1000 random values between 0-1, which we will pretend are our p-values for differential expression from our simulation.

You may try and count how often it less than 0.05

```
p.ifelse <- ifelse(p.1000 < 0.05, 1, 0)  # If it is less than 0.05, then you get a 1,␣
↪otherwise 0.
```

Our approximate false positives. Should be close to 0.05

```
sum(p.ifelse)/length(p.1000)
```

```
## [1] 0.059
```

However the best and fastest way to accomplish this is to use the index, by setting up the Boolean (TRUE/FALSE) in the index of the vector.

```
length(p.1000[p.1000 < 0.05])/length(p.1000)
```

```
## [1] 0.059
```

Same number, faster and simpler computation.

Simple loops

while() function..

I tend to avoid these, so you will not see them much here

```
i <- 1
while (i <= 10) {
    print(i)
    i <- i + 0.5
}
```

```
## [1] 1
## [1] 1.5
## [1] 2
## [1] 2.5
## [1] 3
## [1] 3.5
## [1] 4
## [1] 4.5
## [1] 5
## [1] 5.5
## [1] 6
## [1] 6.5
## [1] 7
## [1] 7.5
## [1] 8
## [1] 8.5
## [1] 9
## [1] 9.5
## [1] 10
```

for loop

If I run a loop I most often use `for(){}` automatically iterates across a list (in this case the sequence from 1:10).

```
for (i in 1:10) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

If you do not want to use integers, how might you do it using the for()?

```
for (i in seq(from = 1, to = 5, by = 0.5)) {
    print(i)
}
```

```
## [1] 1
## [1] 1.5
## [1] 2
## [1] 2.5
## [1] 3
## [1] 3.5
## [1] 4
## [1] 4.5
## [1] 5
```

Using strings is a bit more involved in R, compared to other languages. For instance the following does not do what you want:

```
.. code:: r
```

for (letter in "word") { print(letter)

}

```
## [1] "word"
```

(try letters for a hoot.)

Instead in R, we have to split the word "word" into single characters using strsplit(), i.e:

```
.. code:: r
```

strsplit("word", split = "")

```
## [[1]]
## [1] "w" "o" "r" "d"
```

So for the for loop we would do the following:

```
for (letter in strsplit("word", split = "")) {
    print(letter)
}
```

```
## [1] "w" "o" "r" "d"
```

More avoiding loops

Many would generate random numbers like so.

```
for (i in 1:100) {
    print(rnorm(n = 1, mean = 0, sd = 1))
}
```

```
## [1] -0.1837
## [1] -0.9313
## [1] 1.648
## [1] -0.6964
```

```
## [1] 0.2112
## [1] 0.3441
## [1] 1.036
## [1] 0.7439
## [1] 0.5859
## [1] -0.6087
## [1] -0.4014
## [1] 1.44
## [1] -0.3906
## [1] -1.861
## [1] -0.739
## [1] -1.204
## [1] 0.07794
## [1] -1.65
## [1] 1.261
## [1] 0.6753
## [1] 0.6736
## [1] 0.3238
## [1] -1.316
## [1] 0.2965
## [1] 1.499
## [1] 0.4326
## [1] 0.4488
## [1] 0.8873
## [1] -1.304
## [1] -0.347
## [1] 0.3491
## [1] 0.24
## [1] 0.1425
## [1] -0.2785
## [1] -0.5072
## [1] -1.775
## [1] -0.04051
## [1] 0.9452
## [1] 0.3322
## [1] -0.01994
## [1] -0.2308
## [1] -0.4053
## [1] -0.5685
## [1] -1.631
## [1] -0.1484
## [1] 0.434
## [1] 1.653
## [1] 1.57
## [1] 0.1308
## [1] -1.059
## [1] -0.7157
## [1] -0.8316
## [1] 0.06561
## [1] 0.8243
## [1] 0.1841
## [1] 1.048
## [1] 0.1612
## [1] -0.9553
## [1] -0.7569
## [1] -0.288
## [1] -1.837
## [1] 0.7301
```

```
## [1] -2.103
## [1] -1.869
## [1] -1.298
## [1] -1.077
## [1] -0.2139
## [1] -0.9419
## [1] 0.4694
## [1] -1.344
## [1] -0.08514
## [1] -2.055
## [1] -0.803
## [1] -0.7281
## [1] 1.778
## [1] -1.116
## [1] 1.33
## [1] 0.1535
## [1] -2.897
## [1] 0.7305
## [1] 1.228
## [1] 1.697
## [1] -0.8183
## [1] -1.013
## [1] -0.634
## [1] -0.942
## [1] -0.3395
## [1] 0.1396
## [1] 1.022
## [1] 0.9868
## [1] -0.7778
## [1] 1.075
## [1] -0.1029
## [1] 0.2644
## [1] 0.01165
## [1] 0.8025
## [1] -1.24
## [1] -0.8865
## [1] 0.981
## [1] 0.5333
```

We are cycling through and generating one random number at each iteration. Look at the indices, and you can see we keep generating vectors of length 1.

better/cleaner/faster to generate them all at one time

```
rnorm(n = 100, mean = 0, sd = 1)
```

```
##   [1] -0.08683 -1.55262 -1.16909  0.30451 -1.14555  0.76682  0.12643
##   [8] -0.61174 -0.29103 -0.10707 -0.03397 -0.05926  0.27294  1.32693
##  [15] -0.53284  1.83234  0.43959 -0.88991  0.25383  0.96709 -0.23210
##  [22] -1.00190 -1.32289  1.80030  1.15272 -1.82907  0.75989  1.35966
##  [29]  0.53943  0.01429 -0.58707 -0.11886 -0.70367 -2.38988  0.08033
##  [36] -0.22795 -0.62166 -0.19832 -1.95990 -0.85127  0.94236  0.37771
##  [43]  0.32617 -0.08393 -0.54506 -2.58781 -0.58433  0.20985 -0.41613
##  [50]  0.60527  0.51713  1.57950 -0.61079 -0.28564 -0.16444  0.55007
##  [57]  0.57258  0.58513 -0.86728 -0.81185 -0.29333 -1.23935  0.46169
##  [64] -1.53586 -0.32583  0.17629 -0.85579  1.04989  1.22120  1.53359
##  [71] -2.37276  1.44393  1.47506  0.40110 -0.10157  0.35485 -0.72068
##  [78] -1.27910  0.63152 -0.65216  1.60160  0.27109  0.50904 -1.00531
```

```
##  [85]  0.76743 -0.78954 -0.01159  1.06944  1.15661 -0.91031  1.54919
##  [92] -0.84334  2.19994  0.26716  0.02081  0.53577  0.07840 -0.79387
##  [99] -1.18941  1.24745
```

The not advisable approach

First we initialize a vector to store all of the numbers. Why do we initialize this vector first?

```
n <- 1e+05
x <- rep(NA, n)
```

The step above creates a vector of n NA's. They will be replaced sequentially with the random numbers as we generate them (using a function like the above one).

```
head(x)
```

```
## [1] NA NA NA NA NA NA
```

Now we run the for loop.

```
for (i in 1:n) {
    x[i] <- rnorm(n = 1, mean = 0, sd = 1)
}
```

for each `i` in the index, one number is generated, and placed in x

```
head(x)
```

```
## [1]  0.2848 -0.5432  1.1391 -1.0901  0.8515  0.5490
```

However this is computationally inefficient in R. Which has vectorized operations.

```
system.time(

for (i in 1:n){
    x[i] <- rnorm(n=1, mean=0, sd=1)})
```

```
##    user  system elapsed
##   0.562   0.023   0.584
```

We can also use the replicate function to do the same thing. Easier syntax to write.

```
system.time(z <- replicate(n, rnorm(n = 1, mean = 0, sd = 1)))
```

```
##    user  system elapsed
##   0.561   0.035   0.841
```

This is ~20% faster.

However, since R is vectorized, both of the will be far slower than:

```
system.time(y <- rnorm(n, 0, 1))
```

```
##    user  system elapsed
##   0.010   0.000   0.011
```

About 65 times faster than the for loop

The general rule in R is that loops are slower than the apply family of functions (for small to medium data sets, not true for very large data) which are slower than vectorized computations.

Variant calling and exploration of polymorphisms

Now that we have some experience in R, we will check out a vcf file with polymorphisms from

Getting the data and installing extra packages

Installing a bunch of stuff:

get bwa

```
cd /root
wget -O bwa-0.7.10.tar.bz2 http://sourceforge.net/projects/bio-bwa/files/bwa-0.7.10.
↪tar.bz2/download
```

untar and compile (via make) bwa

```
tar xvfj bwa-0.7.10.tar.bz2
cd bwa-0.7.10
make

cp bwa /usr/local/bin
```

install some tools

```
apt-get update
apt-get -y install samtools screen git curl gcc make g++ python-dev unzip \
      default-jre pkg-config libncurses5-dev r-base-core \
      r-cran-gplots python-matplotlib sysstat libcurl4-openssl-dev libxml2-dev

git clone https://github.com/schimar/ngs2014_popGen.git

cd ngs2014_popGen/var_call2/
```

Let's do another round of variant calling

index the reference genome

```
bwa index ref_genome.fna
```

map our reads to the indexed reference genome

```
bwa aln ref_genome.fna read_file.fq > mapped_reads.sai
```

Create the SAM file

```
bwa samse ref_genome.fna mapped_reads.sai read_file.fq > mapped_reads.sam
```

Index the reference genome

```
samtools faidx ref_genome.fna
```

Convert from SAM to BAM

```
samtools view -b -S -o mapped_reads.bam mapped_reads.sam
```

Sort the BAM

```
samtools sort mapped_reads.bam mapped_reads.sorted
```

And index again, but now the sorted BAM file

```
samtools index mapped_reads.sorted.bam
```

Visualize the alignment

```
samtools tview mapped_reads.sorted.bam ref_genome.fna
```

Variant exploration with Bioconductor

Now simply type R in the shell and:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("VariantAnnotation")
biocLite("SNPlocs.Hsapiens.dbSNP.20101109")
biocLite("BSgenome.Hsapiens.UCSC.hg19_1.3.1000")
```

Quality control

Now we load the VariantAnnotation package as well as the data. The objective of this exercise is to compare the quality of called SNPs that are located in dbSNP, versus those that are novel.

```
library(VariantAnnotation)
fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
```

Locate the sample data in the file system. Explore the metadata (information about the content of the file) using scanVcfHeader . Discover the 'info' fields VT (variant type), and RSQ (genotype imputation quality).

```
(hdr <- scanVcfHeader(fl))
info(hdr)[c("VT", "RSQ"),]
```

Input the data and peak at their locations:

```
(vcf <- readVcf(fl, "hg19"))

head(rowData(vcf), 3)
```

SNPs were called with MaCH/thunder (part of GotCloud) , for more info, see :doc: http://genome.sph.umich.edu/wiki/Thunder and http://genome.sph.umich.edu/wiki/MaCH_FAQ. Notice that the seqnames (chromosome levels) are set to '22', we want to rename those

```
rowData(vcf) <- renameSeqlevels(rowData(vcf), c("22"="ch22"))
```

We now load the SNP database and discover whether our SNPs are in dbSNP

```
library(SNPlocs.Hsapiens.dbSNP.20101109)

destination <- tempfile()
pre <- FilterRules(list(isLowCoverageExomeSnp = function(x) {
grepl("LOWCOV,EXOME", x, fixed=TRUE)
}))
filt <- FilterRules(list(isSNP = function(x) info(x)$VT == "SNP"))
snpFilt <- filterVcf(fl, "hg19", destination, prefilters=pre, filters= filt)
vcf_filt <- readVcf(snpFilt, "hg19")

rowData(vcf)
rowData(vcf_filt)
```

If we compare vcf and vcf_filt, we see that of the 10376 SNPs in our initial vcf file, 794 are in the database.

```
inDbSNP <- rownames(vcf) %in% rownames(vcf_filt)
table(inDbSNP)
metrics <- data.frame(inDbSNP = inDbSNP, RSQ = info(vcf)$RSQ)
```

Let's finally visualize it:

```
library(ggplot2)
ggplot(metrics, aes(RSQ, fill=inDbSNP)) +
geom_density(alpha=0.5) +
scale_x_continuous(name="MaCH / Thunder Imputation Quality") +
scale_y_continuous(name="Density") +
theme(legend.position="top")
```

(This won't work in R on EC2, simply because we can't run X11 through an ssh connection)

A complete de novo assembly and annotation protocol for mRNASeq

The goal of this tutorial is to run you through (part of) a real mRNAseq analysis protocol, using a small data set that will complete quickly.

Prepare for this tutorial by working through *Start up an EC2 instance*, but follow the instructions to start up amazon/starting-up-a-custom-ami instead; use AMI ami-7607d01e.

Switching to root

Start by making sure you're the superuser, root:

```
sudo bash
```

Updating the software on the machine

Copy and paste the following two commands

```
apt-get update
apt-get -y install screen git curl gcc make g++ python-dev unzip \
        default-jre pkg-config libncurses5-dev r-base-core \
        r-cran-gplots python-matplotlib sysstat samtools python-pip
```

If you started up a custom operating system, then this should finish quickly; if instead you started up Ubuntu 14.04 blank, then this will take a minute or two.

Downloading the sample data

The mRNAseq protocol works with the data set that you put in '/data'. Here, we will download a small data set (a subset of the data from this paper, data from embryonic Nematostella>'__), and put it in /data

```
mkdir /mnt/data
ln -fs /mnt/data /data
cd /data
curl -O http://athyra.idyll.org/~t/mrnaseq-subset.tar
tar xvf mrnaseq-subset.tar
```

Check it out:

```
ls
```

You'll see a bunch of different files – these are the kinds of files you'll get from your sequencing facility.

Starting on the protocols

We're going to work with a special version of the protocols today, one that we adapted specifically for this course.

In general, you should use the latest version, which will be at https://khmer-protocols.readthedocs.org/.

For today, we'll be using http://khmer-protocols.readthedocs.org/en/ngs2014/ instead.

Work through the following:

1. Quality trimming
2. Applying digital normalization
3. Running the actual assembly
4. BLASTing your assembly

Actually using the BLAST Web server

To connect to your BLAST Web server, you need to enable inbound traffic on your computer. Briefly:

- go to your instance and look at what security group you're using

(should be 'launch-wizard-' something). On the left panel, under Network and Security, go into Security Groups. Select your security group, and select Inbound, and Edit. Click "Add rule", and change "Custom TCP rule" to "http". Then click "save". Done!

You can try pasting this into your BLAST server:

```
MDRSVNVIQCAAAPTRIQCEEINAKLMLGVGVFGLCMNIVLAVIMSFGAAHPHSHGMLSSVEFDHDVDYH
SRDNHHGHSHLHHEHQHRDGCSHSHGNGGADMQRLECASPESEMMEEVVVETTSSNAESICSHERGSQSM
NLRAAVLHVFGDCLQSLGVVLAACVIWAGNNSSVGVPSSAHSYYNLADPLLSVLFGVITVYTTLNLFKEV
IVILLEQVPPAVEYTVARDALLSVEKVQAVDDLHIWAVGPGFSVLSAHLCTNGCATTSEANAVVEDAECR
CRQLGIVHTTIQLKHAADVRNTGA
```

Assembly with SOAPdenovo-Trans

Startup AMI

```
sudo apt-get -y install screen git curl gcc make g++ python-dev unzip \
    default-jre pkg-config libncurses5-dev r-base-core \
    r-cran-gplots python-matplotlib sysstat samtools python-biopython
```

Installation

```
wget http://downloads.sourceforge.net/project/soapdenovotrans/SOAPdenovo-Trans/bin/v1.
↪03/SOAPdenovo-Trans-bin-v1.03.tar.gz

tar -zxf SOAPdenovo-Trans-bin-v1.03.tar.gz
```

make SOAP folder and get into it

```
mkdir SOAP
cd SOAP
```

Make config file

```
nano config.txt

#maximal read length
max_rd_len=100
[LIB]
#maximal read length in this lib
rd_len_cutof=45
#average insert size
avg_ins=200
#if sequence needs to be reversed
reverse_seq=0
#in which part(s) the reads are used
asm_flags=3
#minimum aligned length to contigs for a reliable read location
map_len=32
#fastq file for read 1
q1=/path/**LIBNAMEA**/fastq_read_1.fq
#fastq file for read 2 always follows fastq file for read 1
q2=/path/**LIBNAMEA**/fastq_read_2.fq
```

Assembly optimization

```
mkdir SOAP
nano config.txt
for k in 31 41 51 61 71 91;
    do SOAPdenovo-Trans-127mer all -L 300 -p 4 -K $k -s config.txt -o assembly$k; done
```

Pick the best assembly

```
Transrate -> http://hibberdlab.com/transrate/
```

Mapping and Counting

Install and run cd-hit est

```
wget https://cdhit.googlecode.com/files/cd-hit-v4.6.1-2012-08-27.tgz
tar -zxf cd-hit-v4.6.1-2012-08-27.tgz
cd cd-hit-v4.6.1-2012-08-27
make
PATH=$PATH:/home/ubuntu/cd-hit-v4.6.1-2012-08-27
cd-hit-est -i Trinity_all_X.fasta -o trin.fasta
```

bwa -> BWA is used to map quality trimmed reads (not normalized) to the reference transcriptome that we just generated using cd-hit-est. There are several different read mappers (bwa, bowtie, bowtie2, Mosaik, etc) all of which do the same thing (map reads), though the underlying statistical underpinnings, and thus the quality of the mapping may vary.

```
cd $HOME

# Pull the most recent version from the Github repository
git clone https://github.com/lh3/bwa.git
cd bwa
#make BWA
make
#Put the BWA executable in the $PATH
PATH=$PATH:/home/ubuntu/bwa
```

eXpress -> http://bio.math.berkeley.edu/eXpress/. eXpress is the software that takes a SAM/BAM as input and produces a table of counts (as well as TPM, FPKM, and lots of other numbers). Read the website and manuscript http://dx.doi.org/10.1038/nmeth.2251

```
cd $HOME

wget http://bio.math.berkeley.edu/eXpress/downloads/express-1.5.1/express-1.5.1-linux_
↪x86_64.tgz
tar -xzf express-1.5.1-linux_x86_64.tgz
cd express-1.5.1-linux_x86_64
PATH=$PATH:/home/ubuntu/express-1.5.1-linux_x86_64
```

Do Mapping and generate count data. We will map reads with BWA, and generate count data with eXpress.

```
cd $HOME
mkdir soap_index && cd soap_index

# make an index using your reference transcriptome.
bwa index -p all /path/to/your/cd-hit-est.fastA


#map the reads to your reference transcriptome using BWA
#This produces a SAM file

bwa mem -t 4 all \
```

```
/mnt/ebs/trimmed_x/ORE_sdE3_rep1_1_pe \
/mnt/ebs/trimmed_x/ORE_sdE3_rep1_2_pe  > ORE_sdE3_rep1.sam

#Count reads mapping to your reference.

express -p4 /path/to/your/cd-hit-est.fastA ORE_sdE3_rep1.sam
```

Analyzing RNA-seq counts with DESeq

In this tutorial, we will continue analyzing the Drosophila RNA-seq data from earlier to look for differentially expressed genes.

Instead of running these analyses on an Amazon EC2 instance, we'll run this locally on our own computers. Before you begin, you will need to download all of the count files we generated using HTSeq. (You can use scp at the command line, or WinSCP to download them.) Place them all in a single folder on your computer. Then, start R.

First, we need to make sure that R can find the files when we try to load them. Just like Unix, R has a current working directory. You can set the working directory using the setwd() function:

```
setwd("...")
```

Be sure to replace the "..." with the path to the folder where you have placed your files.

Right now, we have our data spread out across multiple files, but we need to combine them all into a single dataset for DESeq to be able to use them. There are some convenience functions (see below) to perform this. But for many programs you need to write your own, so we will go through getting it all in writing the function ourselves.

> Let's start by creating a vector, to store the names of the samples and then write a function that will read in a single dataset given a sample name:

```
samples <- c("ORE_wt_rep1","ORE_wt_rep2","ORE_sdE3_rep1","ORE_sdE3_rep2",
↪"SAM_wt_rep1","SAM_wt_rep2","SAM_sdE3_rep1","SAM_sdE3_rep2","HYB_wt_rep1",
↪"HYB_wt_rep2","HYB_sdE3_rep1","HYB_sdE3_rep2")

#A function to read one of the count files produced by HTSeq
read.sample <- function(sample.name) {
        file.name <- paste(sample.name, "_htseq_counts.txt", sep="")
        result <- read.delim(file.name, col.names=c("gene", "count"), sep="\t
↪", colClasses=c("character", "numeric"))
}
```

Now let's try it out:

```
#Read the first sample
sample.1 <- read.sample(samples[1])
```

Let's double check that we've loaded the first sample properly by looking at the first few rows and seeing how many rows there are:

```
head(sample.1)
nrow(sample.1)
```

Do you get the output you expected? Remember you can always check by using `wc -l yourfile` at the shell

Now let's read the second sample, and double check that it has the same number of rows as the first sample. Before we merge the two datasets by :code:‘cbind()‘ing the count columns together, we should make sure that the same gene is represented in each row in the two datasets:

```
#Read the second sample
sample.2 <- read.sample(samples[2])

#Let's make sure the first and second samples have the same number of rows and the␣
↪same genes in each row
nrow(sample.1) == nrow(sample.2)
all(sample.1$gene == sample.2$gene)
```

Looks good. Now let's do all the merging using a simple for loop:

```
#Now let's combine them all into one dataset
all.data <- sample.1
all.data <- cbind(sample.1, sample.2$count)
for (c in 3:length(samples)) {
        temp.data <- read.sample(samples[c])
        all.data <- cbind(all.data, temp.data$count)
}

#We now have a data frame with all the data in it:
head(all.data)
```

You'll notice that the column names are not very informative. We can replace the column names manually to something more useful like this:

```
colnames(all.data)[2:ncol(all.data)] <- samples

#Now look:
head(all.data)

#Let's look at the bottom of the data table
tail(all.data)
```

When you look at the bottom of the dataset, you'll notice that there are some rows we don't want to include in our analysis. We can remove them easily by taking a subset of the data that includes everything except the last 5 rows:

```
all.data <- all.data[1:(nrow(all.data)-5),]

tail(all.data)
```

Now we're ready to start working with DESeq. If you don't already have it installed on your computer, you will want to install it like this:

```
source("http://bioconductor.org/biocLite.R")
biocLite("DESeq")
```

Once it's installed, we need to load the library like this:

```
library("DESeq")
```

Before we're quite ready to work with the data in DESeq, we need to re-format it a little bit more. DESeq wants every column in the data frame to be counts, but we have a gene name column, so we need to remove it. We can still keep the gene names, though, as the row names (just like each column has a name in a data frame in R, each row also has a name).

```
#Remove the first column
raw.deseq.data <- all.data[,2:ncol(all.data)]
#Set row names to the gene names
```

```
rownames(raw.deseq.data) <- all.data$gene

head(raw.deseq.data)
```

Now we have our data, but we need to tell DESeq what our experimental design was. In other words, say we want to look for genes that are differentially expressed between the two genetic backgrounds, or between the mutant and wild-type genotypes–DESeq needs to know which samples belong to each treatment group. We do this by creating a second data table that has all the sample information. We could do this by creating another data table in a text editor (or if you absolutely must, something like Excel, but be careful because sometimes R has trouble reading files generated by Excel, even if you've saved them as "flat" tab-delimited or comma-delimited files).

But instead, we'll generate this sample information table manually in R since it's not very complicated in this case:

```
#Create metadata
wing.design <- data.frame(
        row.names=samples,
        background=c(rep("ORE", 4), rep("SAM", 4), rep("HYB", 4)),
        genotype=rep( c("wt", "wt", "sdE3", "sdE3"), 3 ),
        libType=rep("paired-end", 12)
)
#Double check it...
wing.design
```

By default R picks the "reference" level for each treatment alphanumerically. So in this case the reference level for background would be "HYB" and for genotype it would be "sdE3". However it will be a bit easier for us to intepret the data if we used the wild type (genotype="wt") as a reference. Also for the background, using one of the pure strains, and not their F1 hybrid may help. We accomplish this as follows:

```
wing.design$genotype <- relevel(wing.design$genotype, ref="wt")
wing.design$background <- relevel(wing.design$background, ref="ORE")
```

Now we can create a DESeq data object from our raw count table and our experimental design table:

```
deseq.data <- newCountDataSet(raw.deseq.data, wing.design)
```

Note that DESeq also has a function newCountDataSetFromHTSeqCount that can automatically handle merging all the raw data files together (from HTSeq), but it's useful to be able to do this manually because not every statistical package you work with will have this functionality.

Now we have to do the "normalization" step and estimate the dispersion for each gene:

```
deseq.data <- estimateSizeFactors(deseq.data)
deseq.data <- estimateDispersions(deseq.data)
```

We have used the "vanilla settings" for both, but it highly recommedable to look at the defaults for the size factors and dispersions. As we discussed, how the gene-wise estimates for dispersions are estimated can have pretty substantial effects.

Let's make sure the dispersion estimates look reasonable:

```
plotDispEsts(deseq.data)
```

In this case it looks ok, although the number of points on the graph is relatively modest compared to most RNA-seq studies, since we have intentionally included only genes on the X chromosome. Note that if the dispersion estimates don't look good you may need to tweak the parameters for the estimateDispersions() function (e.g., maybe try using fitType="local").

Now let's fit some models. These steps are a little bit slower, though not terribly so:

```
fit.full <- fitNbinomGLMs(deseq.data, count ~ background + genotype +␣
↪background:genotype)
fit.nointeraction <- fitNbinomGLMs(deseq.data, count ~ background + genotype)
fit.background <- fitNbinomGLMs(deseq.data, count ~ background)
fit.genotype <- fitNbinomGLMs(deseq.data, count ~ genotype)
fit.null <- fitNbinomGLMs(deseq.data, count ~ 1)
```

Now that we've fit a bunch of models, we can do pairwise comparisons between them to see which one best explains the data [for each gene]. For example, we can ask, for which genes does an interaction term between wt/mutant genotype and genetic background help explain variation in expression?:

```
#Generate raw p-values for the first comparison: full model vs. reduced model without␣
↪an interaction term; significant p-values will tell you that the full/more complex␣
↪model does a "significantly" better job at explaining variation in gene expression␣
↪than the reduced/less complex model (in this case, the one without the interaction␣
↪term)
pvals.interaction <- nbinomGLMTest(fit.full, fit.nointeraction)
#Generate p-values adjusted for multiple comparisons using the Benjamini-Hochberg␣
↪approach
padj.interaction <- p.adjust(pvals.interaction, method="BH")
#Look at the genes that have a significant adjusted p-value
fit.full[(padj.interaction <= 0.05) & !is.na(padj.interaction),]
```

Note that the estimates are already log2 transformed counts, and that by default DESeq (and the glm() family of functions in R) use a "treatment contrast by default. So the genotype column represents a fold change relative to the intercept (in this case for the "ORE" & "wt" treatment combinations). If you had an intercept estimate of 5.2 for a given gene you could just use:

```
2^5.2
```

Which would estimate the number of counts for ORE wild type flies.

We could do a more extreme comparison between the full model and the null model:: pvals.fullnull <- nbinomGLMTest(fit.full, fit.null) padj.fullnull <- p.adjust(pvals.fullnull, method="BH") fit.full[(padj.fullnull <= 0.05) & !is.na(padj.fullnull),] #And so on...

It is also very useful to do some plotting both for QC and for examining the totatily of your data. There are many different graphical approaches to examining your data, which we do not have time to get into here. For now we will just do an MA-plot and a volcano plot as a quick starting point. We will use these to compare the mutant (sdE3) from the wild type (i.e. the mutational treatment):

```
pvals.mutant <- nbinomGLMTest(fit.genotype, fit.null)
padj.interaction <- p.adjust(pvals.interaction, method="BH")
fit.full[(padj.interaction <= 0.05) & !is.na(padj.interaction),]
```

For the volcano plot we have fold change on the X-axis and -log10 of the p-value on the y-axis. First we create a dataframe containing the two relevant columns (the fold change from the treatment contrast, and the unadjusted p values):

```
volcano_mutant <- cbind(pvals.mutant, fit.genotype[,2])
```

Now we can plot this (I am going to produce the same plot but zoom in the second one):

```
plot(x=volcano_mutant[,2], y=-log10(volcano_mutant[,1]), xlab = "FOLD CHANGE", ylab="-
↪log10(p)")
```

You probably notice the weird points that are extreme for fold change, but with low p-values. These are exactly the reason you always need to check your data graphically. What might be going on with these points? We could subset

the data based on fold changes to pull out those genes. However, R has a reasonably useful function that might help for these cases `identify()`. Once it is called scroll your mouse over the plot and click to highlight points. The numbers that appear are the index. Once you have finished press escape:

```
identify(x=volcano_mutant[,2], y=-log10(volcano_mutant[,1]))
```

Remember to press `esc` (or right click)!!! I have highlighted one point and it returned a value of 3713. So I will go back to the original count data and take a look at that row:

```
raw.deseq.data[3713,]
```

Aha! This is a gene with very few counts (mostly zeroes), and we forgot to exclude such genes! In general if the mean number of counts (for a given gene) are below some threshold (say 5 or 10) you should probably exclude that gene since you have sampled so poorly from it that it would not be meaningful.

For now though, we will just zoom in to take a look:

```
plot(x=volcano_mutant[,2], y=-log10(volcano_mutant[,1]), xlab = "FOLD CHANGE", ylab="-
↪log10(p)", xlim=c(-5,5))
abline
```

We can also fit an MA-plot, comparing the mean expression level of a gene with its fold change. As I am lazy, I did not actually compute the mean itself, but used the "intercept" (which represents the mean for the ORE wild type flies). However, this is likely sufficient for this plot:

```
plot(x=fit.genotype[,1], y=fit.genotype[,2], xlim=c(0,20), ylim=c(-5,5),
xlab= " ~ mean log2(counts)", ylab=" fold change", cex.lab=3)
```

We can also do some exploratory plotting to see if there's anything weird going on in our data. For example, we want to make sure that, if we cluster our samples based on overall gene expression patterns, we see clusters based on biological attributes rather than, say, the lane they were sequenced in or the day that the library preps were performed (the latter would indicate that there might be some unaccounted variable in our experimental design that is influencing gene expression):

```
#First, get dispersion estimates "blindly", i.e., without taking into account the␣
↪sample treatments
cdsFullBlind = estimateDispersions( deseq.data, method = "blind" )
vsdFull = varianceStabilizingTransformation( cdsFullBlind )

library("RColorBrewer")
library("gplots")
# Note: if you get an error message when you try to run the previous two lines,
# you may need to install the libraries, like this:
install.packages("RColorBrewer")
install.packages("gplots")
# After the libraries installed, don't forget to load them by running the library()␣
↪calls again
```

Now let's make some heat maps:

```
select = order(rowMeans(counts(deseq.data)), decreasing=TRUE)[1:30]
hmcol = colorRampPalette(brewer.pal(9, "GnBu"))(100)

# Heatmap of count table -- transformed counts
heatmap.2(exprs(vsdFull)[select,], col = hmcol, trace="none", margin=c(10, 6))

# Heatmap of count table -- untransformed counts; you can see this looks pretty␣
↪different
```

```
# from the first heat map
heatmap.2(counts(deseq.data)[select,], col = hmcol, trace="none", margin=c(10,6))

# Heatmap of sample-to-sample distances
dists = dist( t( exprs(vsdFull) ) )
mat = as.matrix( dists )
rownames(mat) = colnames(mat) = with(pData(cdsFullBlind), paste(background, genotype,␣
↪sep=" : "))
heatmap.2(mat, trace="none", col = rev(hmcol), margin=c(13, 13))
```

We can also do a principal components analysis (PCA):

```
print(plotPCA(vsdFull, intgroup=c("background", "genotype")))
```

In this case, we see not only that the samples cluster by genotype and genetic background but also that PC1 represents genetic background, and PC2 seems to represent wt vs. mutant genotype.

RNA-seq: mapping to a reference genome with tophat and counting with HT-seq

In this tutorial, we'll use some sample data from a project we did on flies (Drosophila melanogaster) to illustrate how you can use RNA-seq data to look for differentially expressed genes. We'll try a few different approaches to see whether different tools give similar results. Here's some brief background on the project: we're trying to understand how different wild-type genetic backgrounds can influence the phenotypic effects of mutations, using the developing fly wing as our model system. We have several mutations that disrupt wing development, and we've backcrossed them into the genetic backgrounds of two different wild-type fly strains (SAM and ORE). If you would like to see more about this project (although the data we are using is not yet published) see this link. http://www.genetics.org/cgi/content/long/196/4/1321

For this tutorial, we've taken a subset of the data–we'll look for expression differences in developing wing tissues between wild type and scalloped mutant (sd[E3]) flies in each of the two genetic backgrounds, and in flies with a "hybrid" genetic background (i.e., crosses between SAM/ORE flies, again both with and without the mutation). To make things run a little bit faster, we've included only sequence reads that map to X-linked genes (so consider some of the potential biases for mapping and generating the transcriptome for other tutorials).

First, launch an EC2 instance and log in. Start up an Amazon computer (m1.large or m1.xlarge) using AMI ami-7607d01e (see *Start up an EC2 instance* and amazon/starting-up-a-custom-ami).Go back to the Amazon Console. Now select "snapshots" from the left had column. Changed "Owned by me" drop down to "All Snapshots". Search for "snap-028418ad" - (This is a snapshot with our test RNASeq Drosophila data from Chris) The description should be "Drosophila RNA-seq data". Under "Actions" select "Create Volume", then ok.

> Next, create an EBS volume from our snapshot (snap-642349cb). Make sure to create your EC2 instance and your EBS volume in the same availability zone! The snapshot has the raw reads, as well as pre-computed results files so we don't need to wait for every step to finish running before we proceed.

ow on the left select "Volumes". You should see an "in-use" volume - this is for your running instance, as well as an "available" volume - this is the one you just created from the snapshot and should have the snap-028418ad label. Select the available volume and from the drop down select "Attach Volume". The white box pop up will appear - select in the empty instance box, your running instance should appear as an option. Select it. For the device, enter /dev/sdf. Now attach.

Log in with Windows or from Mac OS X.

Become root

```
sudo bash
```

Attach the EBS volume to your instance and mount it in /mnt/ebs/. If you don't know how to do this, ask for a demonstration.

Mount the data volume. (This is for the data that we added as a snapshot)

```
cd /root
mkdir /mnt/ebs
mount /dev/xvdf /mnt/ebs
```

First, we'll need to install a bunch of software. Some of these tools can be installed using apt-get. Note that apt-get does not necessarily always install the most up-to-date versions of this software! You should always double check versions when you do this. For instance, when I was writing this tutorial, apt-get gave me a warning that cufflinks might be out of date, so we're going to install by downloading it directly from the authors.

```
#Install samtools, bowtie, tophat
apt-get -y install samtools
apt-get -y install bowtie
apt-get -y install tophat
apt-get -y install python-pip
pip install pysam

#Create a working directory to hold software that we're going to install other tools
cd /mnt/ebs
mkdir tools
cd tools

#Download and install HTSeq
curl -O https://pypi.python.org/packages/source/H/HTSeq/HTSeq-0.6.1.tar.gz
tar -xzvf HTSeq-0.6.1.tar.gz
cd HTSeq-0.6.1/
python setup.py build
python setup.py install

#Download and install cufflinks
cd ..
curl -O http://cufflinks.cbcb.umd.edu/downloads/cufflinks-2.2.1.Linux_x86_64.tar.gz
tar -xzvf cufflinks-2.2.1.Linux_x86_64.tar.gz
cd cufflinks-2.2.1.Linux_x86_64/
find . -type f -executable -exec cp {} /usr/local/bin \;
cd ..
```

Next, we need to get our reference genome. This is another area where you want to be careful and pay attention to version numbers–the public data from genome projects are often updated, and gene ID's, coordinates, etc., can sometimes change. At the very least, you need to pay attention to exactly which version you're working with so you can be consistent throughout all your analyses.

In this case, we'll download the reference Drosophila genome and annotation file (which has the ID's and coordinates of known transcripts, etc.) from ensembl. We'll put it in its own directory so we keep our files organized:

```
cd /mnt/ebs
mkdir references
cd references
curl -O ftp://ftp.ensembl.org/pub/release-75/fasta/drosophila_melanogaster/dna/
↪Drosophila_melanogaster.BDGP5.75.dna.toplevel.fa.gz
gunzip Drosophila_melanogaster.BDGP5.75.dna.toplevel.fa.gz
```

```
curl -O ftp://ftp.ensembl.org/pub/release-75/gtf/drosophila_melanogaster/Drosophila_
↪melanogaster.BDGP5.75.gtf.gz
gunzip Drosophila_melanogaster.BDGP5.75.gtf.gz
```

We also need to prepare the genomes for use with our software tools by indexing them. This is simple to do but takes a little time for large genomes. You can run the following code, but do not have to, since we've included pre-computed indexes on the snapshot:

```
bowtie-build Drosophila_melanogaster.BDGP5.75.dna.toplevel.fa Drosophila_melanogaster.
↪BDGP5.75.dna.toplevel
samtools faidx Drosophila_melanogaster.BDGP5.75.dna.toplevel.fa
```

Now we're ready for the first step: mapping our RNA-seq reads to the genome. We will use tophat+bowtie1, which together are a splicing-aware read aligner. The raw sequencing reads are in /mnt/ebs/drosophila_reads/. Feel free to take a look at how we've named and organized the files:

```
cd /mnt/ebs/drosophila_reads/
ls -lh
```

Don't forget that with your reads, you'll want to take care of the usual QC steps before you actually begin your mapping. The drosophila_reads directory contains raw reads; the trimmed_x directory contains reads that have already been cleaned using Trimmomatic. We'll use these for the remainder of the tutorial, but you may want to try running it with the raw reads for comparison.

Since we have a lot of files to map, it would take a long time to re-write the mapping commands for each one. And with so many parameters, we might make a mistake or typo. It's usually safer to use a simple shell script with shell variables to be sure that we do the exact same thing to each file. Using well-named shell variables also makes our code a little bit more readable:

```
#Create an array to hold the names of all our samples
#Later, we can then cycle through each sample using a simple foor loop
samples[1]=ORE_wt_rep1
samples[2]=ORE_wt_rep2
samples[3]=ORE_sdE3_rep1
samples[4]=ORE_sdE3_rep2
samples[5]=SAM_wt_rep1
samples[6]=SAM_wt_rep2
samples[7]=SAM_sdE3_rep1
samples[8]=SAM_sdE3_rep2
samples[9]=HYB_wt_rep1
samples[10]=HYB_wt_rep2
samples[11]=HYB_sdE3_rep1
samples[12]=HYB_sdE3_rep2

#Create shell variables to store the location of our reference genome and annotation␣
↪file
#Note that we are leaving off the .fa from the reference genome file name, because␣
↪some of the later commands will require just the base of the file name
reference=/mnt/ebs/references/Drosophila_melanogaster.BDGP5.75.dna.toplevel
annotation=/mnt/ebs/references/Drosophila_melanogaster.BDGP5.75.gtf

#Make sure w are in the right directory
#Let's store all of our mapping results in /mnt/ebs/rnaseq_mapping/ to make sure we␣
↪stay organized
cd /mnt/ebs
mkdir rnaseq_mapping
cd rnaseq_mapping
```

```
#Now we can actually do the mapping
for i in 1 2 3 4 5 6 7 8 9 10 11 12
do
    sample=${samples[${i}]}
    #Map the reads
    tophat -p 4 -G ${annotation} -o ${sample} ${reference} /mnt/ebs/trimmed_x/$
↪{sample}_1_pe /mnt/ebs/trimmed_x/${sample}_2_pe
    #Count the number of reads mapping to each feature using HTSeq
    htseq-count --format=bam --stranded=no --order=pos ${sample}/accepted_hits.bam $
↪{annotation} > ${sample}_htseq_counts.txt
done
```

We now have count files for each sample. Take a look at one of the count files using less. You'll notice there are a lot of zeros, but that's partially because we've already filtered the dataset for you to include only reads that map to the X chromosome.

```
less HYB_sdE3_rep1_htseq_counts.txt
```

You can also visualize these read mapping using tview (*Variant calling*):

```
samtools index HYB_sdE3_rep1/accepted_hits.bam
samtools tview HYB_sdE3_rep1/accepted_hits.bam ${reference}.fa
```

Now we'll need to import them into R to use additional analysis packages to look for differentially expressed genes–in this case, DESeq. At this point I usually download these data files and run the analyses locally. I would suggest copying the files using scp or through a synchronized Dropbox folder. Once you've got them downloaded, we're now ready to start crunching some numbers.

RNA-seq: mapping to a reference genome with BWA and counting with HTSeq

The goal of this tutorial is to show you one of the ways to map RNASeq reads to a transcriptome and to produce a file with counts of mapped reads for each gene. This is an alternative approach to mapping to the reference genome, and by using the same dataset as the previous lesson (see drosophila_rnaseq1, we can see the differences between the two approaches.

We will again be using BWA for the mapping (previously used in the variant calling example) and HTSeq for the counting.

Booting an Amazon AMI

Start up an Amazon computer (m1.large or m1.xlarge) using AMI ami-7607d01e (see *Start up an EC2 instance* and amazon/starting-up-a-custom-ami.

Go back to the Amazon Console.

- Select "snapshots" from the left side column.
- Changed "Owned by me" drop down at the top to "All Snapshots"
- Search for "snap-028418ad" - (This is a snapshot with our test RNASeq Drosophila data from Chris) The description should be "Drosophila RNA-seq data".

- Under “Actions” select “Create Volume”, then ok.

Make sure to create your EC2 instance and your EBS volume in the same availability zone, for this course we are using N. Virginia.

- Select “Volumes” from the left side column
- You should see an “in-use” volume - this is for your running instance. You will also see an “available” volume - this is the one you just created from the snapshot from Chris and should have the snap-028418ad label. Select the available volume
- From the drop down select “Attach Volume”.
- A white box pop up will appear - click in the empty instance box, your running instance should appear as an option. Select it.
- For the device, enter /dev/sdf.
- Attach.

Log in with Windows or from Mac OS X.

Updating the operating system

Become root:

```
sudo bash
```

Copy and paste the following two commands to update the computer with all the bundled software you’ll need.

```
apt-get update
apt-get -y install screen git curl gcc make g++ python-dev unzip \
        pkg-config libncurses5-dev r-base-core \
        r-cran-gplots python-matplotlib sysstat git python-pip
pip install pysam
```

Mount the data volume. (This is the volume we created earlier from Chris’s snapshot - this is where our data will be found):

```
cd /root
mkdir /mnt/ebs
mount /dev/xvdf /mnt/ebs
```

Install software

First, we need to install the BWA aligner

```
cd /root
wget -O bwa-0.7.10.tar.bz2 http://sourceforge.net/projects/bio-bwa/files/bwa-0.7.10.
↪tar.bz2/download
tar xvfj bwa-0.7.10.tar.bz2
cd bwa-0.7.10
make
cp bwa /usr/local/bin
```

We also need a new version of samtools:

```
cd /root
curl -O -L http://sourceforge.net/projects/samtools/files/samtools/0.1.19/samtools-0.
↪1.19.tar.bz2
tar xvfj samtools-0.1.19.tar.bz2
cd samtools-0.1.19
make
cp samtools /usr/local/bin
cp bcftools/bcftools /usr/local/bin
cd misc/
cp *.pl maq2sam-long maq2sam-short md5fa md5sum-lite wgsim /usr/local/bin/
```

Create a working directory to hold some more software that we're going to install

```
cd /mnt/ebs
mkdir tools
cd tools
```

Download and install HTSeq

```
curl -O https://pypi.python.org/packages/source/H/HTSeq/HTSeq-0.6.1.tar.gz
tar -xzvf HTSeq-0.6.1.tar.gz
cd HTSeq-0.6.1/
python setup.py build
python setup.py install
chmod u+x ./scripts/htseq-count
```

We are also going to get a project, chado-test, from Scott Cain's git hub account that will allow us to use a convenient file format conversion script.

```
cd /mnt/ebs/tools
git clone https://github.com/scottcain/chado_test.git
```

For that we will need bioperl installed

```
cpan
```

Answer yes until you get a prompt that looks like

```
cpan[1]>
```

And type

```
install Bio::Perl
```

When it asks "Do you want to run tests that require connection to servers across the internet", answer no. The final line when finished should be:

```
./Build install  -- OK
```

Now exit the CPAN shell

```
exit
```

Preparing the reference

Next, we are going to work with our reference transcriptome. Drosophila has a reference genome, but for this adventure, we are going to pretend that it doesn't. Instead we are going to use the Trinity assembly as our reference - Chris has provided this file, named Trinity_all_X.fasta. Notice the fasta format; each line beginning with a > is a new sequence, followed by another line (or multiple lines) containing the sequence itself. If we want to count how many transcripts are in the file, we can just count the number of lines that begin with >

```
cd /mnt/ebs/trinity_output
grep '>' Trinity_all_X.fasta | wc -l
```

You should see 8260. Now lets use bwa to index the file, this enables the file to be used a reference for mapping:

```
bwa index Trinity_all_X.fasta
```

To generate count files, we will use HTSeq. But HTSeq is expecting a genome annotation file, which we don't have (since we're using the transcriptome). So we have to do some data massaging. We will will create an annotation file that says that the entire length of each "scaffold" is in fact a coding region.

```
cd /mnt/ebs/rnaseq_mapping2
/mnt/ebs/tools/chado_test/chado/bin/gmod_fasta2gff3.pl \
--fasta_dir /mnt/ebs/trinity_output/Trinity_all_X.fasta \
--gfffilename Trinity_all_X.gff3 \
--type CDS \
--nosequence
```

Now you should have a file named Trinity_all_X.gff3 in your current directory.

Mapping

Lets check out the reads to be mapped

```
cd /mnt/ebs/drosophila_reads
ls -lh
```

Don't forget that with your reads, you'll want to take care of the usual QC steps before you actually begin your mapping. The drosophila_reads directory contains raw reads; the trimmed_x directory contains reads that have already been cleaned using Trimmomatic. We'll use these for the remainder of the tutorial, but you may want to try running it with the raw reads for comparison.

We've got 12 sets of data, each with two files (R1 and R2). Let's run bwa on the first pair to map our paired-end sequence reads to the transcriptome. To make our code a little more readable and flexible, we'll use shell variables in place of the actual file names. In this case, let's first specify what the values of those variables should be:

```
reference=/mnt/ebs/trinity_output/Trinity_all_X.fasta
sample=HYB_sdE3_rep1
```

Now we can use these variable names in our mapping commands. The advantage here is that we can just change the variables later on if we want to apply the same pipeline to a new set of samples:

```
cd /mnt/ebs
mkdir rnaseq_mapping2
cd rnaseq_mapping2
bwa mem ${reference} /mnt/ebs/trimmed_x/${sample}_1_pe /mnt/ebs/trimmed_x/${sample}_2_
↪pe > ${sample}.sam
```

The output is a file named HYB_sdE3_rep1_2.sam in the current working directory. This file contains all of the information about where each read hits on the reference. Next, we want to use SAMTools to convert it to a BAM, and then sort and index it:

```
samtools view -Sb ${sample}.sam > ${sample}.unsorted.bam
samtools sort ${sample}.unsorted.bam ${sample}
samtools index ${sample}.bam
```

Now we can generate a counts file with the HTSeq-count script:

```
htseq-count --format=bam --stranded=no --type=CDS --order=pos --idattr=Name ${sample}.
↪bam Trinity_all_X.gff3 > ${sample}_htseq_counts.txt
```

Optional - Script these steps

Since we have a lot of files to map, it would take a long time to re-write the mapping commands for each one. And with so many parameters, we might make a mistake or typo. It's usually safer to use a simple shell script with shell variables to be sure that we do the exact same thing to each file. Using well-named shell variables also makes our code a little bit more readable. Open a file named map_and_count.sh and paste in the following code:

```
#Create an array to hold the names of all our samples
#Later, we can then cycle through each sample using a simple foor loop
samples[1]=ORE_wt_rep1
samples[2]=ORE_wt_rep2
samples[3]=ORE_sdE3_rep1
samples[4]=ORE_sdE3_rep2
samples[5]=SAM_wt_rep1
samples[6]=SAM_wt_rep2
samples[7]=SAM_sdE3_rep1
samples[8]=SAM_sdE3_rep2
samples[9]=HYB_wt_rep1
samples[10]=HYB_wt_rep2
samples[11]=HYB_sdE3_rep1
samples[12]=HYB_sdE3_rep2

#Create a shell variable to store the location of our reference genome
reference=/mnt/ebs/trinity_output/Trinity_all_X.fasta

#Make sure we are in the right directory
#Let's store all of our mapping results in /mnt/ebs/rnaseq_mapping2/
↪ to make sure we stay organized
#If this directory already exists, thats ok, but files might get overwritten
cd /mnt/ebs
mkdir rnaseq_mapping2
cd rnaseq_mapping2

#Now we can actually do the mapping and counting
for i in 1 2 3 4 5 6 7 8 9 10 11 12
do
    sample=${samples[${i}]}
    #Map the reads
    bwa mem ${reference} /mnt/ebs/trimmed_x/${sample}_1_pe /mnt/ebs/trimmed_x/$
↪{sample}_2_pe  > ${sample}.sam
    samtools view -Sb ${sample}.sam > ${sample}.unsorted.bam
    samtools sort ${sample}.unsorted.bam ${sample}
    samtools index ${sample}.bam
```

```
    htseq-count --format=bam --stranded=no --type=CDS --order=pos --idattr=Name $
↪{sample}.bam Trinity_all_X.gff3 > ${sample}_htseq_counts.txt
done
```

To run this script, change the permissions and run:

```
chmod u+x ./map_and_count.sh
./map_and_count.sh
```

We now have count files for each sample. Take a look at one of the count files using less. You'll notice there are a lot of zeros, but that's partially because we've already filtered the dataset for you to include only reads that map to the X chromosome.

You can also visualize these read mapping using tview *Variant calling*.

Genome comparison and phylogeny

This tutorial will introduce genome comparison techniques and some simple methods to compute phylogeny. It will introduce the following pieces of software:

Mauve, for genome alignment PhyloSift & FastTree, for phylogeny

We'll analyze some *E. coli* genome assemblies that were precomputed with techniques previously introduced in the course.

Interactive visual genome comparison with Mauve

Download a copy of the Mauve GUI installer for your platform: Mac OS X Windows Linux

Download the following GenBank format genomes from NCBI: E. coli O157:H7 EDL933 E. coli CFT073

An assembly of the Ribiero et al 2012 *E. coli* MiSeq data (accession SRR519926) made by the A5-miseq pipeline (paper here):

E. coli A5-miseq assembly

An assembly of the Chitsaz et al 2011 data made by the Velvet diginorm workflow:

E. coli Velvet DN assembly

Running a genome alignment

Launch the Mauve software, either from the start menu on Windows, or by double-clicking the Mauve app on Mac. On newer Mac OS it may be necessary to disable the nanny security features that prohibit opening downloaded software. If you see error messages suggesting the disk image is damaged and can not be opened then you need to disable the security check.

Once Mauve is open, go to the File menu, select "Align with progressiveMauve..." Drag & drop the NCBI genomes in first. Then drag & drop in the draft *E. coli* assemblies

The assembly will take 20+ minutes to compute. Proceed to the next steps while this process runs.

Booting an Amazon AMI

Start up an Amazon computer (m1.large or m1.xlarge) using an ubuntu 64-bit linux instance (see *Start up an EC2 instance* and amazon/starting-up-a-custom-ami).

Log in with Windows or from Mac OS X.

Ensure you have dropbox installed on your virtual machine and mounted at /mnt/Dropbox

Logging in & updating the operating system

Copy and paste the following three commands

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java6-installer
sudo apt-get remove openjdk-7-jre openjdk-7-jre-headless
```

to update the computer with all the bundled software you'll need. Mauve, when run in command-line mode, depends on the Oracle Java 6 runtime which the above commands install. The above commands will also remove the openjdk if it's present on the system. If you would prefer to keep this for some reason then look at the `update-alternatives` system.

Packages to install

Install the latest Mauve snapshot to your home directory:

```
cd
curl -O http://gel.ahabs.wisc.edu/mauve/snapshots/2012/2012-06-07/linux-x64/mauve_
↪linux_snapshot_2012-06-07.tar.gz
tar xzf mauve_linux_snapshot_2012-06-07.tar.gz
export PATH=$PATH:$HOME/mauve_snapshot_2012-06-07/linux-x64/
```

as well as the PhyloSift software, for marker-based analysis of genome & metagenome phylogeny:

```
cd
curl -O http://edhar.genomecenter.ucdavis.edu/~koadman/phylosift/releases/phylosift_
↪v1.0.1.tar.bz2
tar xjf phylosift_v1.0.1.tar.bz2
```

Getting the E. coli genome data

Now, let's create a working directory:

```
cd
mkdir draft_genome
cd draft_genome
```

Download some genome assemblies. The first one is an assembly constructed with A5-miseq. The second is an assembly constructed with Velvet.

```
curl -O http://edhar.genomecenter.ucdavis.edu/~koadman/ngs2014/ecoli_a5.final.
↪scaffolds.fasta
curl -O http://edhar.genomecenter.ucdavis.edu/~koadman/ngs2014/ecoli_dn_velvet25.fa
```

What is the nearest reference genome?

In this step, we will use the PhyloSift software to determine where our newly assembled genome falls in the tree of life, and by extension what closely related genomes might be available for comparative genomics. In principle this would be a first step after sequencing a novel organism

```
~/phylosift_v1.0.1/bin/phylosift all --debug ecoli_dn_velvet25.fa
```

When that finishes (probably after a very long time), copy the results to dropbox for local viewing:

```
cp -r PS_temp/ecoli_dn_velvet25.fa /mnt/Dropbox
```

once on your local computer, open the .html file in the ecoli_dn_velvet25.fa directory in your web browser. It should be possible to launch the concat marker viewer in java from the link in the bottom left of the page.

Ordering the assembly contigs against a nearby reference

In the previous step we discovered that our genome was similar to E. coli. We will now use an existing finished-quality E. coli genome as a reference for ordering and orienting the contigs in the assembly. This is useful because the genome assembly process usually creates a large number of contigs or scaffolds rather than complete reconstructions of the chromosome(s) and these sequences appear in an arbitrary order. By ordering against a reference we can generate a candidate ordering which could be used for later manual closure efforts (e.g. via PCR) or other analyses. Let's use the *E. coli* K12 genome from NCBI as a reference:

```
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K_12_substr__DH10B_
↪uid58979/NC_010473.fna
```

And now run the Mauve Contig Mover to order the contigs:

```
java -Xmx500m -Djava.awt.headless=true -cp ~/mauve_linux_snapshot_2012-06-07/Mauve.
↪jar  org.gel.mauve.contigs.ContigOrderer -output reorder_a5 -ref NC_010473.fna -
↪draft ecoli_a5.final.scaffolds.fasta
java -Xmx500m -Djava.awt.headless=true -cp ~/mauve_linux_snapshot_2012-06-07/Mauve.
↪jar  org.gel.mauve.contigs.ContigOrderer -output reorder_a5 -ref NC_010473.fna -
↪draft ecoli_dn_velvet25.fa
```

Let's copy the newly ordered genome to dropbox:

```
cp reorder_a5/alignment3/ecoli_a5.final.scaffolds.fasta /mnt/Dropbox/
```

This could now be aligned with Mauve as demonstrated above to observe the improvement in contig order.

Making a phylogeny of many E. coli assemblies

For this component we will use a collection of related *E. coli* and *Shigella* genomes already on NCBI. In practice, you might use your own collection of assemblies of these genomes. Let's start out in a new directory and download these:

```
mkdir ~/phylogeny ; cd ~/phylogeny

# download a bunch of genomes from NCBI. Alternatively you can use the approach that
# Adina introduced in the previous lesson to programmatically download many files
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_O157_H7_EDL933_
↪uid57831/NC_002655.fna
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_CFT073_uid57915/NC_
↪004431.fna
```

```
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K_12_substr__DH10B_
↪uid58979/NC_010473.fna
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_O104_H4_2011C_3493_
↪uid176127/NC_018658.fna
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Shigella_flexneri_2a_2457T_uid57991/
↪NC_004741.fna
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Shigella_boydii_Sb227_uid58215/NC_
↪007613.fna
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Shigella_dysenteriae_Sd197_uid58213/
↪NC_007606.fna
curl -O http://edhar.genomecenter.ucdavis.edu/~koadman/ngs2014/ecoli_a5.final.
↪scaffolds.fasta
curl -O http://edhar.genomecenter.ucdavis.edu/~koadman/ngs2014/ecoli_dn_velvet25.fa

# find homologs of the elite marker genes
find . -maxdepth 1 -name "*.fna" -exec ~/phylosift_v1.0.1/bin/phylosift search --
↪isolate --besthit {} \;
~/phylosift_v1.0.1/bin/phylosift search --isolate --besthit ecoli_a5.final.scaffolds.
↪fasta

# align to the marker gene profile HMMs
find . -maxdepth 1 -name "*.fna" -exec ~/phylosift_v1.0.1/bin/phylosift align --
↪isolate --besthit {} \;
~/phylosift_v1.0.1/bin/phylosift align --isolate --besthit ecoli_a5.final.scaffolds.
↪fasta

# combine the aligned genes into a single file
find . -type f -regex '.*alignDir/concat.codon.updated.1.fasta' -exec cat {} \; | sed␣
↪-r 's/\.1\..*//'  > codon_alignment.fa

# infer a phylogeny with FastTree
~/phylosift_v1.0.1/bin/FastTree -nt -gtr < codon_alignment.fa > codon_tree.tre

# now copy the tree over to dropbox
cp codon_tree.tre /mnt/Dropbox/
```

From tree file to figures

At last we have a phylogeny! The last steps are to view it, interpret it, and publish it. There are many phylogeny viewer softwares, here we will use FigTree. You will need to download and install FigTree to your computer. Once installed, either launch by double-click (Mac) or via the start menu (Windows). Now we can open the tree file codon_tree.tre from dropbox.

Once open, enable the node labels which show bootstrap confidence. Optionally midpoint root the tree, adjust the line width, and export a PDF.

This document (c) copyleft 2014 Aaron Darling.

Automation, scripts, git, and GitHub

Start up an Ubuntu 14.04 instance and run

```
sudo bash
```

```
apt-get update
apt-get -y install screen git curl gcc make g++ python-dev unzip \
        default-jre pkg-config libncurses5-dev r-base-core \
        r-cran-gplots python-matplotlib sysstat
```

Automation and scripts

Suppose you want to run a few different commands in succession – this is called "automating tasks". See http://xkcd.com/1205/

Key caveat: none of the commands require any user input ('y', 'n', etc)

1. Figure out what commands you want to run.
2. Put them in a file using a text editor, like nano or pico, or TextEdit, or TextWrangler, or vi, or emacs.
3. Run the file.

For example, try putting the following shell commands in a script called 'test.sh':

```
cd /root
curl -O http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.
↪32.zip
unzip Trimmomatic-0.32.zip
cp Trimmomatic-0.32/trimmomatic-0.32.jar /usr/local/bin
cd /mnt
curl -O http://athyra.idyll.org/~t/subreads-A-R1.fastq.gz
curl -O http://athyra.idyll.org/~t/subreads-A-R2.fastq.gz

java -jar /usr/local/bin/trimmomatic-0.32.jar PE subreads-A-R1.fastq.gz subreads-A-R2.
↪fastq.gz s1_pe s1_se s2_pe s2_se LEADING:2 TRAILING:2 SLIDINGWINDOW:4:2 MINLEN:25
```

If you want to do this ghetto style, type:

```
cd /root
cat > test.sh
```

then paste in the above, put a newline at the end, and then type CTRL-D.

Now, run:

```
bash test.sh
```

...and this will run all of the stuff that you put in test.sh.

This is called a shell script (because it is a *script* for running things, written in *shell language*). Adina has already told you about Python scripts a bit.

You can do this kind of "scripting" with any set of commands. Just keep track of exactly what you're teaching at the command line by pasting it into a document somewhere, then save the document in text format, and that's your script!

Note: what happens when you run 'bash test.sh' again?

Another note: notice the explicit 'cd' steps... why?

But now! You're stuck keeping track of all of these files.

See: http://www.phdcomics.com/comics/archive.php?comicid=1531

This is what version control is for.

Here are some things to try.

Some git koans

Forking a repository on github

1. **In a browser**, log into github.com.
2. Go to https://github.com/ngs-docs/ngs-scripts and click "fork" (upper right). This will make a copy of that git repository under *your* account. It should leave you at http://github.com/<YOUR ACCOUNT>/ngs-scripts.
3. Select the URL about midway down the page ('https://...') and copy it to your clipboard. Hint: There's a handy little button on the right to do this.
4. Go to your EC2 command line, and type:

   ```
   git clone https://github.com/<YOUR ACCOUNT>/ngs-scripts.git
   ```

   where the last bit is pasted from what you copied in step 3.
5. Change into the ngs-scripts directory:

   ```
   cd ngs-scripts/
   ```

   and poke around.
6. Marvel. Note that what is in your directory is the same as what you can see via the github interface.
7. **In a browser**, go back to your copy of ngs-scripts. Select 'README.md' in the top-level directory.
8. Select 'Edit'.
9. Change something in the text box (e.g. add "Kilroy was here.")
10. Click "Commit changes".
11. Note that in the browser, README.md has been updated.
12. **In the command line**, note that README.md hasn't changed. The repositories are distinct and separate.
13. Type:

    ```
    git pull https://github.com/<YOUR ACCOUNT>/ngs-scripts.git master
    ```

    to *pull* the changes from github into your local copy.
14. Now README.md is the same in both places!!

What you have done here is *cloned* your repository, then *edited* your file in the original repository, and then *pulled* the changes from the original repository into your new repository.

Create a new file on github and edit it, then pull

Note the '+' after the directory name (next to 'branch: ') just above the list of files.

This gives you the opportunity to create and edit a *new* file.

Do so, 'commit new file', and then do step #13 above.

Now you've created a new file on github!

Edit local file and push to github

At the command line,

1. Edit the README file (either with a local editor like 'pico', or with Dropbox, or something; e.g. do:

```
cp README.md ~/Dropbox
(edit it)
cp ~/Dropbox/README.md .
```

   to update it remotely and copy it back over). Use 'more' to make sure your local copy is different.

2. Type:

```
git diff
```

   to see your changes. The lines with '+' at the beginning are your new changes, the lines with '-' at the beginning are what they replaced.

2. Type:

```
git commit -am "made some changes"
```

   to commit the changes as things you want to do.

   (At this point, you could also type 'git checkout README.md' to replace the changed file with the original.)

3. Type:

```
git push https://github.com/<YOUR ACCOUNT>/ngs-scripts.git master
```

4. Marvel that the local changes are now viewable on github.com directly!

What you have done here is to edit files in one repository, and then *pushed* the changes to another (remote) repository.

Create a new repository; add some files to it.

Let's create a new repository, just for you.

In a Web browser,

1. Go to http://github.com/ and click on "New repository."
2. Make up a repository name (it will suggest one; ignore it.)
3. Select the "initialize this repo with a README."
4. Select 'Create repository."
5. Now, clone it to your EC2 machine:

```
git clone https://github.com/<YOUR ACCOUNT>/<YOUR REPO NAME>.git
```

6. Change into the new repo directory:

```
cd <YOUR REPO NAME>
```

7. Create a new file:

```
echo hello world > greetings.txt
```

8. Add it to your repository:

```
git add greetings.txt
```

9. Commit it:

```
git commit -am "added greetigs"
```

10. Push it to your github repository:

```
git push https://github.com/<YOUR ACCOUNT>/<YOUR REPO NAME>.git master
```

11. Go check it out on the Web – do you see greetings.txt?

MG-RAST and its API

Just like the NCBI databases, there are many ways you can interact with MG-RAST, and the web interface is possibly the *worst* way.

Another way you could work with MG-RAST is to download the entire database and then write parsers to get what you want out of it. I've also found this incredibly painful but if you want to do that, you can find its database here.

The best way to access MG-RAST data in my experience is to learn to use their API. MG-RAST has done a decent job publishing API documentation – it just takes a bit of practice to understand its structure.

Example Usage

You read a paper, and the authors reference MG-RAST metagenomes. You want to download these so you can reproduce some of the analysis and ask some of your own questions.

Table S2. Diversity metrics, MG-RAST IDs, and percent of metagenomic reads annotated

Biome type	Sample ID	MG-RAST ID	% of quality reads annotated	Metagenomic richness (S)	Metagenomic diversity (H′)	Bacterial 16S richness (S)	Bacterial 16S diversity (H′)	Bacterial 16S phylogenetic diversity (PD)
Polar desert	EB017	4477900.3	14.5	1,535	6.39	4,527	5.31	300.0
Polar desert	EB019	4477901.3	23.6	1,663	6.52	2,796	3.60	261.1
Polar desert	EB020	4477902.3	17.3	1,376	6.33	4,936	5.79	305.6
Polar desert	EB021	4477903.3	15.9	1,228	6.17	2,845	4.57	195.3
Polar desert	EB024	4477904.3	17.2	1,386	6.34	4,124	5.56	270.0
Polar desert	EB026	4477803.3	20.5	2,231	6.78	2,935	4.92	232.9
Hot desert	MD3	4477805.3	16.4	1,948	6.60	8,895	6.72	485.8
Hot desert	SF2	4477872.3	14.4	1,850	6.56	10,078	6.93	554.4
Hot desert	SV1	4477873.3	17.3	1,981	6.68	9,929	7.14	527.4
Tropical forest	AR3	4477875.3	13.3	1,814	6.51	9,264	5.72	537.1
Boreal forest	BZ1	4477876.3	17.5	2,270	6.79	9,002	6.54	512.9
Temperate deciduous forest	CL1	4477877.3	18.2	2,393	6.81	12,352	7.06	675.0
Temperate coniferous forest	DF1	4477899.3	18.3	2,414	6.81	12,150	6.68	664.6
Temperate grassland	KP1	4477804.3	17.2	2,193	6.72	10,253	6.60	557.4
Tropical forest	PE6	4477807.3	15.6	2,317	6.70	8,772	6.66	476.8
Arctic tundra	TL1	4477874.3	18.8	2,375	6.84	6,965	6.27	437.6

Percentages of quality-filtered shotgun metagenomic reads that could be annotated to functional gene categories and diversity indices calculated from both the shotgun metagenomic data and the 16S rRNA gene amplicon data.

For example, here is some data from a recent PNAS paper, "Cross-biome metagenomic analyses of soil microbial communities and their functional attributes"

If we wanted to download this data with the API, I'd look at the documentation *download*, here. You'll see a couple examples that lists how you would download different stages:

```
http://api.metagenomics.anl.gov/1/download/mgm4447943.3?file=350.1
```

Or...:

```
http://api.metagenomics.anl.gov/1/download/mgm4447943.3?stage=650
```

These two commands above download a specific file or show files from a specific stage for the MG-RAST metagenome ID 4447943.3. You'll notice how they look similar to the NCBI API calls, with a specific structure. You're also requesting specific data with the query terms given after the ID with this *&* structure. Try putting these *urls* into your web browser and you can see the results.

Remember that you can also access the same commands on the shell with the *curl* command, but you need to know what kind of output you expect.

This command outputs a file so you need to save the file to an output:

> curl "http://api.metagenomics.anl.gov/1/download/mgm4447943.3?file=350.1" > 350.1.fastq.gz

This command returns text (in JSON structure):

> curl "http://api.metagenomics.anl.gov/1/download/mgm4447943.3?stage=650"

As a beginner, I often didn't know what to expect and would just try things out – which I recommend as a good way to learn.

Even more useful, I think is the following command:

> http://api.metagenomics.anl.gov/1/download/mgm4447943.3

I like to put this in a web browser because it pretty prints the JSON text output. This command above gives all the data that can be obtained from the *download* call for this metagenome.

A challenge for MG-RAST is that the types of files and the stages aren't that well-documented. You can get a good guess of what the files and their content from the download page on the web interface, e.g., here. I can tell you from experience that the most important files for me are as follows:

1. File 050.2 - This is the unfiltered metagenome that was originally uploaded to MG-RAST
2. File 350.2 & 350.3 - These are the protein coding genes (amino acids and nucleotides)
3. File 440.1 - These are predicted rRNA sequences (I do not recommend using MG-RAST for sensitive rRNA annotation. It does not use the internal structure of the gene, which other programs appropriately use for classification)
4. File 550.1 - This file shows clustered sequences which are 90% identical, to reduce the number of sequences that need to be annotated. Many folks don't even know that this happens within MG-RAST.
5. File 650.1 & 650.2 - These files are essentially the blat tabular output from comparing your sequence to the database.

A few words on the MG-RAST database. This often confuses people about MG-RAST. The central part of the MG-RAST database is a set of known protein sequences. These known sequences are identified by a unique ID (a mix of numbers and letters). Each known sequence is then related to a known annotation in several databases (e.g., RefSeq, KEGG, SEED, etc.). In other words, the search of your sequences to the database involves a sequence comparison to a sequences in the M5nr sequence database and these sequences are then linked to "a hub" of annotations in several databases. If MG-RAST wants to add another database to its capabilities, it would identify the IDs of sequences related to the sequences in the database. If it existed, the new database annotation would be added to the hub. Otherwise, a new ID would be created and also a new annotation hub. As a consequence of all this, the main thing I work with in MG-RAST is these unique IDs.

Exercise - Download

Try downloading a few metagenomes from the PNAS paper and associated files. Can you think of how to automate doing this?

MG-RAST annotates sequences and can estimate the abundance of taxonomy and function. Using structured databases like SEED, you can thus find broad functional summaries, e.g., the amount of carbon metabolism in various metagenomes.

In general, I'm paranoid and like to do any sort of abundance counting on my own. Let me give you an example, if one of my sequences hits two sequences in the MG-RAST database with identical scores, what should one do in the abundance accounting?

Working with Annotations

Honestly, I'm never sure what MG-RAST is doing, so I like to be in charge of those decisions. Most typically, I am working with 3 types of datasets in any sort of experimental analysis:

1. an annotation file linking my sequence to a database (hopefully one with some structure like SEED),
2. an abundance file (giving estimates of each of my sequences in my database), and
3. some sort of metadata describing my experiment and samples.

MG-RAST can provide you with all three of these, but I typically use it only for #1 (and thus this tutorial also focuses on this). This does require a good deal of know-how in scripting land.

To download these annotation files for specific databases (rather than the unique MG-RAST ID), I use the API annotation command. Using the API, I'll select the database I'd like to use and the type of data within that database I would like returned (e.g., function, taxonomy, or unique ID – aka md5sum).

There are a couple examples on the documentation that are worth trying:

```
http://api.metagenomics.anl.gov/1/annotation/sequence/mgm4447943.3?evalue=10&
↪type=organism&source=SwissProt
```

The above returns a sequence FASTA file with the annotation included in the header of each sequence.:

```
http://api.metagenomics.anl.gov/1/annotation/similarity/mgm4447943.3?identity=80&
↪type=function&source=KO
```

I use this more often. The above returns the BLAT results in a tabular format, including the annotations in the last column. Note that with the *curl* command I can save this to a file and then parse it on my own.

Some comments on the parameters within *type* within these API calls:

1. Organism and function are self-explanatory.
2. Ontology is the "structure" of the database, e.g., Subsystems groups SEED sequences into broader functional groups which have their own unique IDS like SS0001.
3. Feature - This is the most basic ID within the database of choice, e.g., in RefSeq, this would be its accession ID.
4. MD5 - this is the unique ID within MG-RAST.

Note: The other good parameter to be aware of is *version*. This is important to keep all your analysis consistent. And also guarantees that you are working with the most recent database. Also, when you have to go back and repeat the analysis, you'll know what version you used. The problem is that MG-RAST has almost *no* documentation on versions right now. You should write them and complain.

If you do want to download aspects of the database for your analysis, you'll want to explore the documentation for m5nr API calls. With these calls, you can download the various databases you interact with and more importantly, the *ontology* structure of databases.

For example, you can see the information for any md5 ID in RefSeq:

```
http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?
↪source=RefSeq&version=10
```

Or in all MG-RAST databases:

```
http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10
```

If you want to download taxonomy information:

```
http://api.metagenomics.anl.gov/1/m5nr/taxonomy?version=1
```

Or functional information in the SEED:

```
http://api.metagenomics.anl.gov/1/m5nr/ontology?source=Subsystems&min_level=function
```

Note: One of the things you'll notice when you run these commands in the command line with *curl* is that the output is pretty ugly. You'll want to parse these outputs in a programming language you know and look for a JSON parser. I'm most familiar with Python's library json, which can import JSON text into Python libraries easily.

I generally use these downloads to link to my annotations. For example, I'd get the SSID that a sequence might be associated with in a BLAT table download and then link it to the database ontology with a m5nr download call.

A note on JSON

You might be wondering how to work with these JSON outputs in your own scripting. For example, for this call:

```
curl http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?
↪version=10
```

The output of the raw JSON looks like this:

```
{
   next: "http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10&offset=10",
   prev: null,
   version: "10",
   url: "http://api.metagenomics.anl.gov//m5nr/md5/000821a2e2f63df1a3873e4b280002a8?version=10&offset=0",
 - data: [
    - {
         source: "InterPro",
         function: "Sulfatase",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR000917"
      },
    - {
         source: "InterPro",
         function: "Domain of unknown function DUF1705",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR012549"
      },
    - {
         source: "InterPro",
         function: "Alkaline phosphatase-like, alpha/beta/alpha",
         type: "protein",
         ncbi_tax_id: 0,
         md5: "000821a2e2f63df1a3873e4b280002a8",
         organism: "Serratia proteamaculans (strain 568)",
         accession: "IPR017849"
      },
```


If you look closely, it looks a lot like a Python *dictionary* structure and that's how most folks interact with it. Since I program mainly in Python, I use its JSON libraries to work with these outputs in my scripting. I installed the library ijson. In your home directory on your instance, install the library:

```
wget https://pypi.python.org/packages/source/i/ijson/ijson-1.1.tar.gz
tar -zxvf ijson-1.1.tar.gz
cd ijson-1.1
python setup.py install
```

You can test that it was installed:

```
python
>>import ijson
>>
```

No error message means you're good to go.

To work with this data structure, I'd look at it first in your pretty JSON-printed webbrowser.

You'll notice that the data is broken down into a set of nested objects. In this example, the first level contains objects like the version, url, and data. If you go into the data object, you'll see nested data about source, function, type, ncbi_tax_id, etc.

I access the specific object "data" in Python with the following code:

```
import urllib
import ijson

url_string = "http://api.metagenomics.anl.gov//m5nr/md5/
↪000821a2e2f63df1a3873e4b280002a8?version=10"

f = urllib.urlopen(url_string)
```

```
objects = ijson.items(f, '')
for item in objects:
    for x in item["data"]:
        print x["function"], x["ncbi_tax_id"], x["organism"], x["source"], x["type"],␣
↪x["md5"]
```

Now, if I had a much larger object, say the one below, I'd save it to a file first:

```
curl http://api.metagenomics.anl.gov/1/m5nr/taxonomy?version=1 > taxonomy_download.
↪json
```

Then, I would parse through the file:

```
import urllib
import ijson
import sys

f = open(sys.argv[1])
objects = ijson.items(f, '')

for item in objects:
    for x in item["data"]:
        if x.has_key("domain"):
            print x["domain"], x["ncbi_tax_id"]
            #note that not all tax_id's have an associated domain
```

Exercise - linking MG-RAST to taxonomy

One of the most aggravating searches in MG-RAST is linking a md5sum to its taxonomy. But...once you do it, you can give yourself a huge pat on the back for understanding how to interact with this API.

Can you figure out how to do it? For a given md5sum, identify its taxonomic lineage. What if you had to automate this for several md5sums?

1. Download the BLAT tabular output for mgm4447943.3 (Hint: the file type is 650.2)
2. Identify the best hits for the first 50 reads. (Hint: remember your BLAST tutorial?)
3. Find the taxonomy id associated with the first 50 reads using the API call. (Hint: you're going to want to write your own script for interacting with the following string "http://api.metagenomics.anl.gov//m5nr/md5/" + m5nr + "?source=GenBank")
4. Find the taxonomy lineage associated with that taxon ID (Hint: See this script.

You can also get taxonomy from NCBI returned in XML format:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=taxonomy&id=376637
```

Another tool I've used is Biopython, which has parsers for XML and Genbank files. Its something I think is worth knowing exists and occasionally I use it, especially for its parsers. Here's a script that I use it for to get taxonomy for a NCBI Accession Number, here and its also in the repo I've been working with during the workshop.

So you want to get some sequencing data out of NCBI?

Requirements: You'll need to start an Ubuntu EC2 instance and have root access. The first part of this tutorial will be on your local computer and then we'll move onto the EC2 instance. Also, this tutorial assumes that someone has

talked to you about paths and you know how to change directories and execute a program on a file. If you get an error that a program or file does not exist, make sure you are in the right path.

First, let's think about how these databases are structured. I am going to create a database for folks to deposit whole genome sequences. What kind of information am I going to store in this? Many of you may be familiar with such a database, hosted by the NCBI. The scripts that complement this tutorial can be downloaded with the following:

```
git clone https://github.com/adina/scripts-for-ngs.git
```

Let's come up with a list of things we'd like stored in this database and discuss some of the challenges involved in database creation, management, and access.

The challenge

So, you've been given a list of genomes and been asked to create a phylogenetic tree of these genomes. How big would this list be before you thought about hiring an undergraduate to download sequences?

Say the list is only 3 genomes:

```
CP000962
CP000967
CP000975
```

The following are some ways with which I've used to grab genome sequences:

1. Use the web portal and look up each FASTA
2. Use the FTP site, find each genome, and download manually
3. Use the NCBI Web Services API to download the data

Among these, I'm going to assume many of you are familiar with the first two. This tutorial then is going to go through an example of the first approach and then focus on using APIs.

What is an API and how does it relate to NCBI?

Here's some answers, among which my favorite is "an interface through which you access someone else's code or through which someone else's code accesses yours – in effect the public methods and properties."

The NCBI has a whole toolkit which they call *Entrez Programming Utilities* or *eutils* for short. You can read all about it in the documentation. There are a lot of things you can do to interface with all things NCBI, including publications, etc., but I am going to focus today on downloading sequencing data.

To do this, you're going to be using one tool in *eutils*, called *efetch*. There is a whole chapter devoted to efetch – when I first started doing this kind of work, this documentation always broke my heart. Its easier for me to just show you how to use it.

You can use the NCBI efetch utility on your web browser (as is true of many APIs). Open a web browser, and try the following URL to download the nucleotide genome for CB00962:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&
↪rettype=fasta&retmode=text
```

You'll note that a file downloaded on your computer. Take a look at it. You'll note that it looks a lot like what you would see if you had searched for CP00962 on the NCBI search page. Check it out here.

If I want to access other kinds of data associated with this genome. For example, if I want the Genbank file as an output rather than a FASTA file, I would try the following command:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&
↪rettype=gb&retmode=text
```

Do you notice the difference in these two commands? Let's breakdown the command here:

1. <http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?> This is command telling your computer program (or your browser) to talk to the NCBI API tool efetch.
2. <db=nuccore> This command tells the NCBI API that you'd like it to look in this particular database for some data. Other databases that the NCBI has available can be found here.
3. <id=CP000962> This command tells the NCBI API efetch the ID of the genome you want to find.
4. <rettype=gb&retmode=text> These two commands tells the NCBI how the data is returned. You'll note that in the two examples above this command varied slightly. In the first, we asked for only the FASTA sequence, while in the second, we asked for the Genbank file. Here's some elusive documentation on where to find these "return" objects.

Also, a useful command is also <version=1>. There are different versions of sequences and some times that is useful. For reproducibility, I try to specify versions in my queries, see these comments.

Note: Notice the "&" that comes between each of these little commands, it is necessary and important.

Automating with an API

Ok, let's think of automating this sort of query.

In the shell, you could run the same commands above with the addition of *curl* (a program to get information from remote sources) on your EC2 instance:

```
curl "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&
↪rettype=fasta&retmode=text"
```

You'll see it fly on to your screen. Don't panic - you can save it to a file with the redirect command ">" and make it more useful.:

```
curl "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=CP000962&
↪rettype=fasta&retmode=text" > CP000962.fa
```

You've now saved a query you've sent to the NCBI API to a file. Now, you could now imagine writing a program where you made a list of IDs you want to download and put it in a for loop, *curling* each genome and saving it to a file. The following is a script. Thanks to Jordan Fish who gave me the original version of this script before I even knew how and made it easy to use.

This script has some nifty documentation that you can see by trying to execute the script. To see the documentation for this script:

```
python fetch-genomes.py
```

You'll see that you need to provide a list of IDs and a directory where you want to save the downloaded files. What do you need to provide to this script? The first thing is a file that contains a list of IDs (note that this is a required format, each ID on a new line) to fetch the data from NCBI. Second, you need to name a directory where you want the program to put the files you fetch from the NCBI API. Note that the lazy programmer who wrote this script requires you to identify a directory that does not currently exist.

So to use this script, two things are needed. What are they?

To help out, I have provided a list of 50 IDs in a file called "interesting-genomes.txt." How can you tell there are 50 genomes in this file? But I don't want to just go wild and download all 50 at once.

Note: You may want to run this on just a few of these IDs to begin with. You can create a smaller list using the *head* command with the -n parameter in the shell. For example, head -n 3 interesting-genomes.txt > 3genomes.txt.

To run the script:: python fetch-genomes.py 3genomes.txt 3genomes

Take a look at what happened, and when you're ready to try it for more files you could try the following. Note that the directory
python fetch-genomes.py interesting-genomes.txt genbank-files

Let's take a look inside this script. The meat of this script uses the following code:

```
url_template = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?
↪db=nucleotide&id=%s&rettype=gb&retmode=text"
```

You'll see that the *id* here is a string character which is obtained from list of IDs contained in a separate file. The rest of the script manages where the files are being placed and what they are named. It also prints some output to the screen so you know its running.

Exercise - Downloading data

Try modifying the fetch_genomes.py script to download just the FASTA sequences of the genes.

Running this script should allow you to download genomes to your heart's content. But how do you grab specific genes from this data then? Specifically, the challenge was to make a phylogenetic tree of sequences, so let's target the conserved bacterial gene, *16S ribosomal RNA gene*.

Comment on Genbank files

Genbank files have a special structure to them. You can look at it and figure it out for the most part, or read about it in detail here. To find out if your downloaded Genbank files contain 16S rRNA genes, I like to run the following command:

```
grep 16S *gbk
```

This should look somewhat familiar from your shell lesson, but basically we're looking for anylines that contain the character "16S" in any Genbank file we've downloaded. Note that you'll have to run this in the directory where you downloaded these files.

The structure of the Genbank file allows you to identify 16S genes. For example,

```
rRNA            9258..10759
                /gene="rrs"
                /locus_tag="CLK_3816"
                /product="16S ribosomal RNA"
                /db_xref="Pathema:CLK_3816"
```

You could write code to find text like 'rRNA' and '/product="16S ribosomal RNA"', grab the location of the gene, and then go to the FASTA file and grab these sequences. To make things easy, there are existing packages to parse Genbank files. I have the most experience with BioPython. To begin with, let's just use BioPython to help us with our program.

First, we'll have to install BioPython on your instance and they've made that pretty easy:

```
sudo apt-get update
sudo apt-get install python-biopython
```

Fan Yang (Iowa State University) and I wrote a script to extract 16S rRNA sequences from Genbank files, here. It basically searches for text strings in the Genbank structure that is appropriate for these particular genes. You can read more about BioPython here and its Genbank parser here. In this script, we are looking for an "rRNA" feature and looking for specific text in its "/product" line. If this is true, we go through the genome sequence and extract the coordinates of these genes, providing the specific gene sequence.

To run this script on the Genbank file for CP000962. Note make sure you are in the right directory for both the program and the files:

```
python parse-genbank.py genbank-files/CP000962.gbk > genbank-files/CP000962.gbk.16S.fa
```

The resulting output file contains all 16S rRNA genes from the given Genbank file.

To run this for multiple files, I use a shell for loop:

```
for x in genbank-files/*; do python parse-genbank.py $x > $x.16S.fa; done
```

There are multiple ways to get this done – but this is how I like to do it. Now, you can figure out how you like to do it.

And there you have it, you can now pretty much automatically grab 16S rRNA genes from any number of genomes in NCBI databases.

Challenge:

Find your favorite gene, download a database of it from NCBI, and find matching sequences from a sequencing dataset.

Looking at k-mer abundance distributions

Start up an Ubuntu 14.04 instance and run

```
sudo bash

apt-get update
apt-get -y install screen git curl gcc make g++ python-dev unzip \
        default-jre pkg-config libncurses5-dev r-base-core \
        r-cran-gplots python-matplotlib sysstat ipython-notebook
```

Install khmer:

```
cd /usr/local/share
git clone https://github.com/ged-lab/khmer.git
cd khmer
git checkout v1.1
make install
```

Download the notebook for today's session:

```
cd /mnt
curl -O https://raw.githubusercontent.com/ngs-docs/angus/2014/files/kmer-abundance-
↪distributions.ipynb
```

Run IPython Notebook:

```
ipython notebook --no-stdout --no-browser --ip='*' --port=80 --notebook-dir=/mnt
```

(Note: you can use:

–directory=/root/Dropbox

instead if you want to store notebooks in your Dropbox.)

To connect to your notebook server, you need to enable inbound traffic on HTTP to your computer. Briefly, go to your instance and look at what security group you're using (should be 'launch-wizard-' something). On the left panel, under Network and Security, go into Security Groups. Select your security group, and select Inbound, and Edit. Click "Add rule", and change "Custom TCP rule" to "http". Then click "save". Done!

Now, open your EC2 machine's address in your Web browser.

—

PacBio Tutorial

Launch a generic AMI (m3.2xlarge), update and install basic software. You can use the generic ami-864d84ee or any other Ubuntu machine.

```
#update stuff
sudo apt-get update

#install basic software
sudo apt-get -y install screen git curl gcc make g++ python-dev unzip \
default-jre pkg-config

#install Perl modules required by PBcR, paste in to terminal one at a time..
#Will be a couple of prompts (answer YES to both)
sudo cpan App::cpanminus
sudo cpanm Statistics::Descriptive

#Install wgs-assembler
wget http://sourceforge.net/projects/wgs-assembler/files/wgs-assembler/wgs-8.2beta/
↪wgs-8.2beta-Linux_amd64.tar.bz2
tar -jxf wgs-8.2beta-Linux_amd64.tar.bz2

#add wgs to $PATH
PATH=$PATH:$HOME/wgs-8.2beta/Linux-amd64/bin/
```

Download sample Lambda phage dataset. We are using this only because it is very small and can be assembled quickly and with limited hardware requirements. For a more challenging test (read: expert with a big computer) try one of publicly available PacBio datasets here: https://github.com/PacificBiosciences/DevNet/wiki/Datasets

```
#make sure you have the appropriate permissions to read and write.
sudo chown -R ubuntu:ubuntu /mnt
mkdir /mnt/data
cd /mnt/data

#Download the sample data
wget http://www.cbcb.umd.edu/software/PBcR/data/sampleData.tar.gz
tar -zxf sampleData.tar.gz
cd sampleData/
```

Convert fastA to faux-fastQ

```
#This is really old PacBio data, provided in fastA format. Look at the reads - note␣
↪that they are not actually as long as I just told you they should be. The PacBio␣
↪tech has improved massively over the past few years.

java -jar convertFastaAndQualToFastq.jar \
pacbio.filtered_subreads.fasta > pacbio.filtered_subreads.fastq
```

Run the assembly, using wgs, after error-correcting the reads. You could do the error correction separately, but no need to, here, for our purposes.

```
PBcR -length 500 -partitions 200 -l lambda -s pacbio.spec \
-fastq pacbio.filtered_subreads.fastq genomeSize=50000
```

Look at the output. The phage genome has been assembled into 2 contigs (meh). Try a larger dataset for a more difficult (and rewarding challenge)

RNASeq Transcript Mapping and Counting (BWA and HtSeq Flavor)

The goal of this tutorial is to show you one of the ways to map RNASeq reads to a transcriptome and to produce a file with counts of mapped reads for each gene.

We will be using BWA for the mapping (previously used in the variant calling example) and HtSeq for the counting.

Booting an Amazon AMI

Start up an Amazon computer (m1.large or m1.xlarge) using AMI ami-7607d01e (see *Start up an EC2 instance* and amazon/starting-up-a-custom-ami).

Go back to the Amazon Console. Now select "snapshots" from the left had column. Changed "Owned by me" drop down to "All Snapshots". Search for "snap-028418ad" - (This is a snapshot with our test RNASeq Drosophila data from Chris) The description should be "Drosophila RNA-seq data". Under "Actions" select "Create Volume", then ok.

Now on the left select "Volumes". You should see an "in-use" volume - this is for your running instance, as well as an "available" volume - this is the one you just created from the snapshot from Chris and should have the snap-028418ad label. Select the available volume and from the drop down select "Attach Volume". The white box pop up will appear - select in the empty instance box, your running instance should appear as an option. Select it. For the device, enter /dev/sdf. Now attach.

Log in with Windows or from Mac OS X.

Updating the operating system

Become root

```
sudo bash
```

Copy and paste the following two commands

```
apt-get update
apt-get -y install screen git curl gcc make g++ python-dev unzip \
        default-jre pkg-config libncurses5-dev r-base-core \
        r-cran-gplots python-matplotlib sysstat
```

to update the computer with all the bundled software you'll need.

Mount the data volume. (This is for Chris's data that we added as a snapshot).

```
cd /root
mkdir /mnt/ebs
mount /dev/xvdf /mnt/ebs
```

Install software

First, we need to install the BWA aligner:

```
cd /root
wget -O bwa-0.7.10.tar.bz2 http://sourceforge.net/projects/bio-bwa/files/bwa-0.7.10.
↪tar.bz2/download

tar xvfj bwa-0.7.10.tar.bz2
cd bwa-0.7.10
make

cp bwa /usr/local/bin
```

We also need a new version of samtools:

```
cd /root
curl -O -L http://sourceforge.net/projects/samtools/files/samtools/0.1.19/samtools-0.
↪1.19.tar.bz2
tar xvfj samtools-0.1.19.tar.bz2
cd samtools-0.1.19
make
cp samtools /usr/local/bin
cp bcftools/bcftools /usr/local/bin
cd misc/
cp *.pl maq2sam-long maq2sam-short md5fa md5sum-lite wgsim /usr/local/bin/
```

Evaluating the quality of your short reads, and trimming them

As useful as BLAST is, we really want to get into sequencing data, right? One of the first steps you must do with your data is evaluate its quality and throw away bad sequences.

Before you can do that, though, you need to install a bunch o' software.

Logging in

Log in and type:

```
sudo bash
```

to change into superuser mode.

Packages to install

Install khmer:

```
cd /usr/local/share
git clone https://github.com/ged-lab/khmer.git
cd khmer
git checkout v1.1
make install
```

Install Trimmomatic:

```
cd /root
curl -O http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.
↪32.zip
unzip Trimmomatic-0.32.zip
cp Trimmomatic-0.32/trimmomatic-0.32.jar /usr/local/bin
```

Install FastQC:

```
cd /usr/local/share
curl -O http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.2.zip
unzip fastqc_v0.11.2.zip
chmod +x FastQC/fastqc
```

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.

Getting some data

Start at your EC2 prompt, then type

```
cd /mnt
```

Now, grab the 5m E. coli reads from our data storage (originally from Chitsaz et al.):

```
curl -O https://s3.amazonaws.com/public.ged.msu.edu/ecoli_ref-5m.fastq.gz
```

You can take a look at the file contents by doing:

```
gunzip -c ecoli_ref-5m.fastq.gz | less
```

(use 'q' to quit the viewer). This is what raw FASTQ looks like!

Note that in this case we've given you the data *interleaved*, which means that paired ends appear next to each other in the file. Most of the time sequencing facilities will give you data that is split out into s1 and s2 files. We'll need to split it out into these files for some of the trimming steps, so let's do that –

```
split-paired-reads.py ecoli_ref-5m.fastq.gz
mv ecoli_ref-5m.fastq.gz.1 ecoli_ref-5m_s1.fq
mv ecoli_ref-5m.fastq.gz.2 ecoli_ref-5m_s2.fq
```

This uses the khmer script 'split-paired-reads' (see documentation) to break the reads into left (/1) and right (/2). (This takes a long time! 5m reads is a lot of data...)

We'll also need to get some Illumina adapter information – here:

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

These sequences are (or were) "trade secrets" so it's hard to find 'em. Don't ask me how I got 'em.

Trimming and quality evaluation of your sequences

Start at the EC2 login prompt. Then,

```
cd /mnt
```

Make a directory to store all your trimmed data in, and go there:

```
mkdir trim
cd trim
```

Now, run Trimmomatic to eliminate Illumina adapters from your sequences –

```
java -jar /usr/local/bin/trimmomatic-0.32.jar PE ../ecoli_ref-5m_s1.fq ../ecoli_ref-
↪5m_s2.fq s1_pe s1_se s2_pe s2_se ILLUMINACLIP:../illuminaClipping.fa:2:30:10
```

Next, let's take a look at data quality using FastQC

```
mkdir /root/Dropbox/fastqc
/usr/local/share/FastQC/fastqc s1_* s2_* --outdir=/root/Dropbox/fastqc
```

This will dump the FastQC output into your Dropbox folder, under the folder 'fastqc'. Go check it out on your local computer in Dropbox – you're looking for folders named <filename>_fastqc, for example 's1_pe_fastqc'; then double click on 'fastqc_report.html'. (You can't look at these on the dropbox.com Web site – it won't interpret the HTML for you.)

It looks like a lot of bad data is present after base 70, so let's just trim all the sequences that way. Before we do that, we want to interleave the reads again (don't ask) –

```
interleave-reads.py s1_pe s2_pe > combined.fq
```

(interleave-reads is another khmer scripts)

Now, let's use the FASTX toolkit to trim off bases over 70, and eliminate low-quality sequences. We need to do this both for our combined/paired files and our remaining single-ended files:

```
fastx_trimmer -Q33 -l 70 -i combined.fq | fastq_quality_filter -Q33 -q 30 -p 50 >␣
↪combined-trim.fq

fastx_trimmer -Q33 -l 70 -i s1_se | fastq_quality_filter -Q33 -q 30 -p 50 > s1_se.filt
```

Let's take a look at what we have –

```
ls -la
```

You should see:

```
drwxr-xr-x 2 root root       4096 2013-04-08 03:33 .
drwxr-xr-x 4 root root       4096 2013-04-08 03:21 ..
-rw-r--r-- 1 root root  802243778 2013-04-08 03:33 combined-trim.fq
-rw-r--r-- 1 root root 1140219324 2013-04-08 03:26 combined.fq
-rw-r--r-- 1 root root  570109662 2013-04-08 03:23 s1_pe
-rw-r--r-- 1 root root     407275 2013-04-08 03:23 s1_se
-rw-r--r-- 1 root root     319878 2013-04-08 03:33 s1_se.filt
-rw-r--r-- 1 root root  570109662 2013-04-08 03:23 s2_pe
-rw-r--r-- 1 root root          0 2013-04-08 03:22 s2_se
```

Let's run FastQC on things again, too:

```
mkdir /root/Dropbox/fastqc.filt
/usr/local/share/FastQC/fastqc combined-trim.fq s1_se.filt --outdir=/root/Dropbox/
↪fastqc.filt
```

Now go look in your Dropbox folder under 'fastqc.filt', folder 'combined-trim.fq_fastqc' – looks a lot better, eh?

Instructor's Guide to ANGUS Materials

The main repository is here: https://github.com/ngs-docs/angus. Please try to keep everything in there as much as possible.

For 2014, contribute to the branch '2014'.

We use Sphinx to build the site from multiple files, and each file is written in reStructuredText.

Merges to the '2014' branch are automatically built and posted by readthedocs.org at http://angus.readthedocs.org/en/2014/

You can use pull requests OR you can just send Titus your github ID and he will give you merge privileges. For the first few modifications we would still suggest using pull requests just so you can get the hang of reST.

Put static files that you do not want interpreted by Sphinx (e.g. presentation PDFs) in the `files/` directory.

Licensing

Everything you do must be releasable under CC0 except for your presentation slides, which must be accessible and posted somewhere reasonably permanent (in our repo, on slideshare, etc) but can be under whatever license you choose.

Workshop Code of Conduct

All attendees, speakers, sponsors and volunteers at our workshop are required to agree with the following code of conduct. Organisers will enforce this code throughout the event. We are expecting cooperation from all participants to help ensuring a safe environment for everybody.

tl; dr: Don't be a jerk.

Need Help?

You can reach the course director, Matt or Meg, at macmanes@gmail.com / mestato@gmail.com or via the cell phone number on the course info page. You can also talk to any of the instructors or TAs if you need immediate help.

Judi Brown Clarke, jbc@msu.edu, is the person to contact if Titus is not available or there are larger problems; she is available via phone at 517.353.5985.

The Quick Version

Our workshop is dedicated to providing a harassment-free workshop experience for everyone, regardless of gender, age, sexual orientation, disability, physical appearance, body size, race, or religion (or lack thereof). We do not tolerate harassment of workshop participants in any form. Sexual language and imagery is not appropriate for any workshop venue, including talks, workshops, parties, Twitter and other online media. Workshop participants violating these rules may be sanctioned or expelled from the workshop *without a refund* at the discretion of the workshop organisers.

The Less Quick Version

Harassment includes offensive verbal comments related to gender, age, sexual orientation, disability, physical appearance, body size, race, religion, sexual images in public spaces, deliberate intimidation, stalking, following, harassing photography or recording, sustained disruption of talks or other events, inappropriate physical contact, and unwelcome sexual attention.

Participants asked to stop any harassing behavior are expected to comply immediately.

If a participant engages in harassing behavior, the workshop organisers may take any action they deem appropriate, including warning the offender or expulsion from the workshop with no refund.

If you are being harassed, notice that someone else is being harassed, or have any other concerns, please contact a member of workshop staff immediately.

Workshop instructors and TAs will be happy to help participants contact KBS security or local law enforcement, provide escorts, or otherwise assist those experiencing harassment to feel safe for the duration of the workshop. We value your attendance.

We expect participants to follow these rules at workshop and workshop venues and workshop-related social events.

This work is licensed under a Creative Commons Attribution 3.0 Unported License.

This Code of Conduct taken from http://confcodeofconduct.com/. See

http://www.ashedryden.com/blog/codes-of-conduct-101-faq

for more information on codes of conduct.