labibi Documentation

Release 5.0

C. Titus Brown

August 18, 2015

Contents

This is the schedule for the 2014 MSU NGS course.

This workshop has a Workshop Code of Conduct.

Download all of these materials or visit the GitHub repository.

Day	Schedule
Monday 8/4	• 1:30pm lecture: Welcome! (Titus) • Tutorial: Day 1 - Getting started with Amazon • 7pm: research presentations
Tuesday 8/5	• Day 2 – Running BLAST and other things at the command line • 9:15am lecture: Sequencing considerations (Titus) • 10:30am: tutorial, Running command-line BLAST (Titus) • Afternoon: assessment • 1:15pm: tutorial, Short Read Quality Control (Elijah and Istvan) • (CTB alternate version Evaluating the quality of your short reads, and trimming them) • Evening: *firepit social*
Wed 8/6	• 9:15am lecture: Mapping and Assembly (Titus) • 10:30am: tutorial, Variant calling (Titus) • 1:15pm: Understanding the SAM format (Istvan) • 7:15pm: tutorial, UNIX command line (Elijah)
Thursday 8/7	• 9:15am lecture: Genomic Intervals (Istvan) • 10:30am mini-diversion: The Bioinformatics Skill System (Istvan) • 10:45am: tutorial, Interval Analysis and Visualization (Istvan) • 1:15pm: tutorial, Assembling E. coli sequences with Velvet (Titus) • 5:30pm: leave for Kalamazoo
Friday 8/8	• 9:15am-noon lecture/tutorial, R Tutorial for NGS2014 R etc. (Ian Dworkin and Martin Schilling) • 1:15pm: tutorial, Variant calling and exploration of polymorphisms • 1:15pm: lecture, more variant calling (Martin Schilling) • 7pm: lecture, Gene and genome annotation: PowerPoint | PDF (Daniel Standage)
Saturday 8/9	• 9:15am-noon: lecture/tutorial, A complete de novo assembly and annotation protocol for mRNASeq (Titus) • 1:15pm: lecture/discussion, mRNAseq assembly with Trinity (Meg Staton)
Monday 8/11	• 9:15am lecture, mRNAseq and counting PDF (Ian Dworkin) • 10:30am tutorial, RNA-seq: mapping to a reference genome with tophat and counting with HTseq (Chris Chandler) • 10:45am tutorial, RNASeq Transcript Mapping and Counting (BWA and HtSeq Flavor) (Meg) • 2:15pm tutorial, Assembly with SOAPdenovo-

CHAPTER 1

Dramatis personae

Instructors:

- Istvan Albert
- C Titus Brown
- Ian Dworkin

TAs:

- Amanda Charbonneau
- Elijah Lowe
- Will Pitchers
- Aswathy Sebastian
- Qingpeng Zhang

Lecturers:

- Chris Chandler
- Adina Chuang Howe
- Matt MacManes
- Martin Schilling
- Daniel Standage
- Meg Staton

He Who Drives Many Places:

- Cody Nicks

CHAPTER 2

Papers and References

2.1 Books

- Practical Computing for Biologists

  This is a highly recommended book for people looking for a systematic presentation on shell scripting, programming, UNIX, etc.

2.2 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.

2.3 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 3

Links

3.1 Humor

- Data Sharing and Management Snafu in 3 Short Acts

3.2 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.

3.3 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!

Complete table of contents

4.1 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.

4.1.1 Start up an EC2 instance

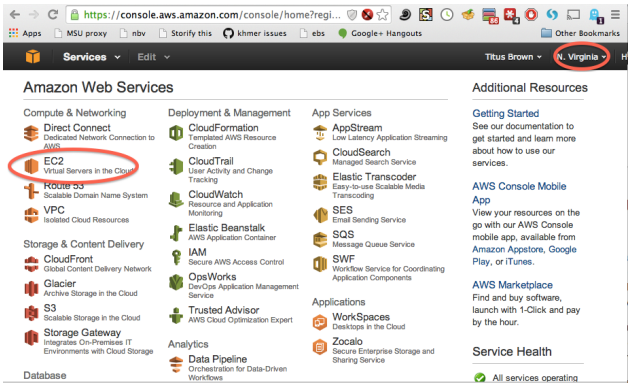
Here, we're going to startup an Amazon Web Services (AWS) Elastic Cloud Computing (EC2) "instance", or computer.

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

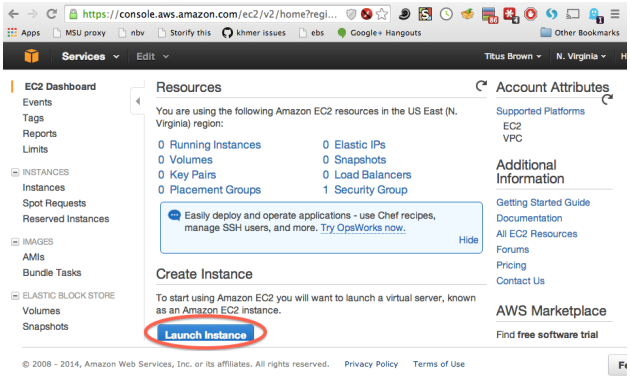
Select 'My Account/Console' menu option 'AWS Management Console."

Log in with your username & password.

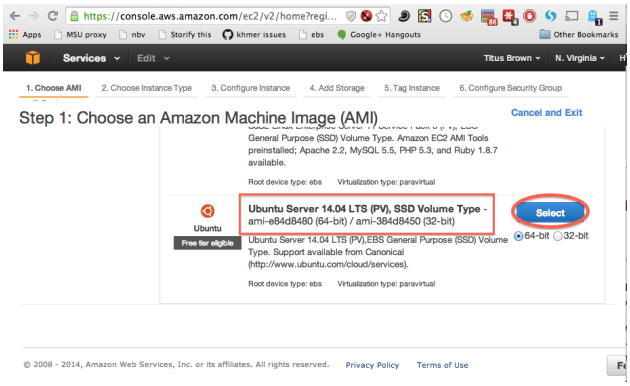
Make sure it says North Virginia in the upper right, then select EC2 (upper left).



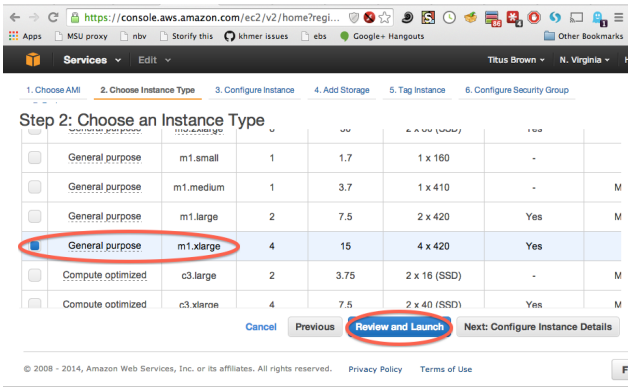
Select "Launch Instance" (midway down the page).



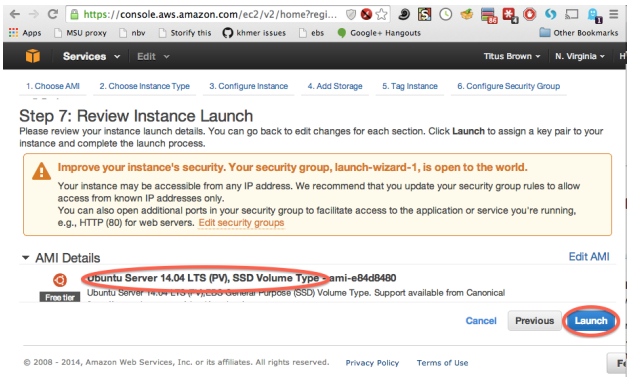
Next, scroll down the list of operating system types until you find Ubuntu 14.04 LTS (PV) – it should be at the very bottom. Click 'select'. (See Starting up a custom operating system if you want to start up a custom operating system instead of Ubuntu 14.04.)



Scroll down the list of instance types until you find "m1.xlarge". Select the box to the left, and then click "Review and Launch."

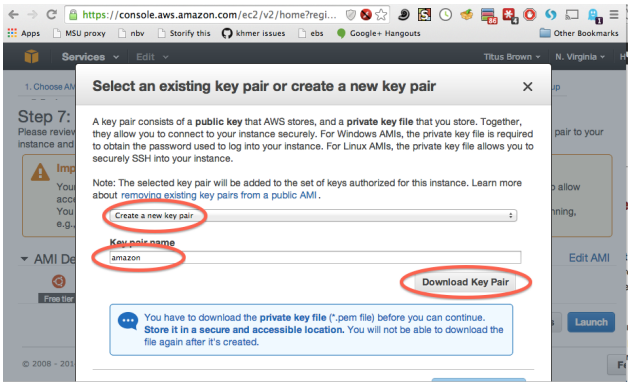


Ignore the warning, check that it says "Ubuntu 14.04 LTS (PV)", and cick "Launch".

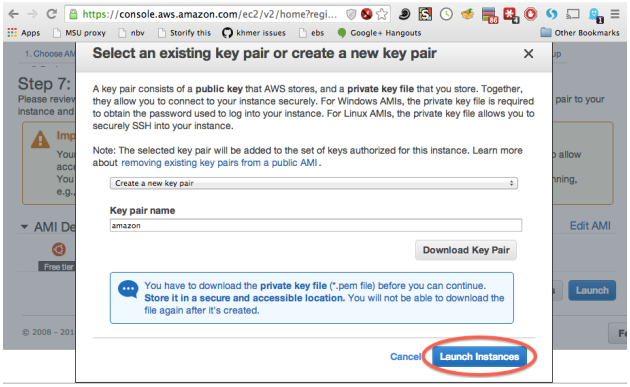


The *first* time through, you will have to "create a new key pair", which you must then name (something like 'amazon') and download.

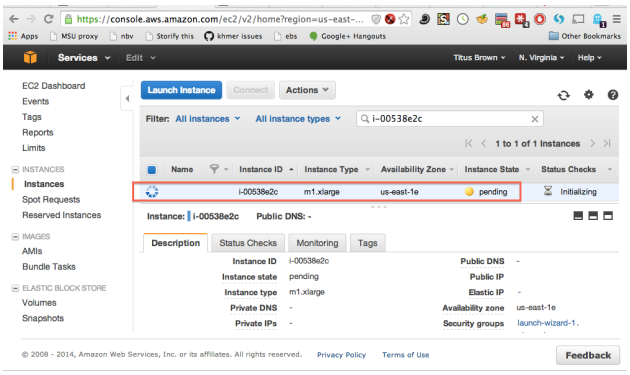
After this first time, you will be able to select an existing key pair.



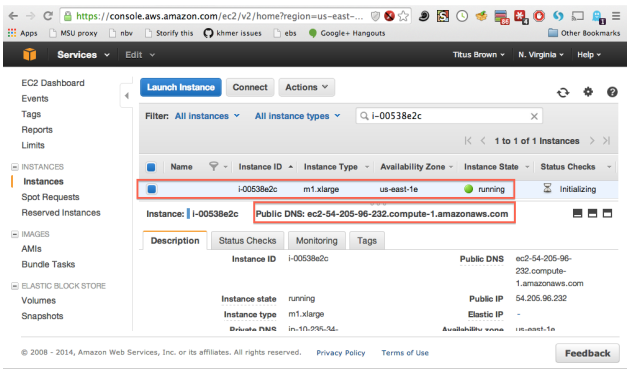
Select "Launch Instance."



Select "view instance" and you should see a "pending" line in the menu.



Wait until it turns green, then make a note of the "Public DNS" (we suggest copying and pasting it into a text notepad somewhere). This is your machine name, which you will need for logging in.



Then, go to Logging into your new instance "in the cloud" (Windows version) or Logging into your new instance "in the cloud" (Mac version)

You might also want to read about Terminating (shutting down) your EC2 instance.

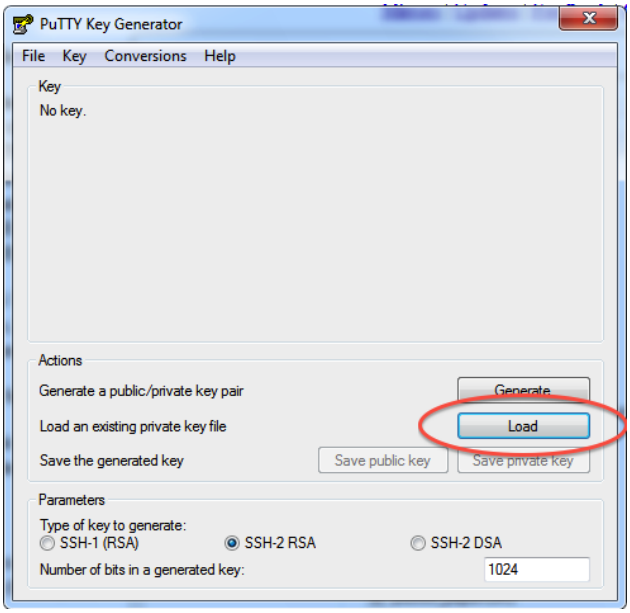
4.1.2 Logging into your new instance "in the cloud" (Windows version)

Download Putty and Puttygen from here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

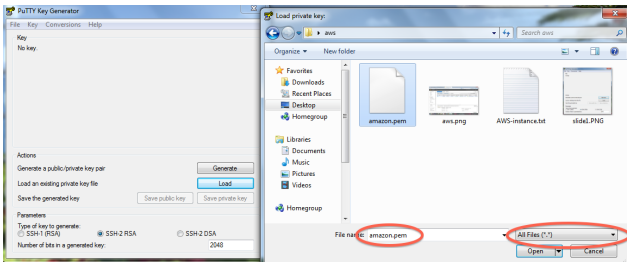
Generate a ppk file from your pem file

(You only need to do this once!)

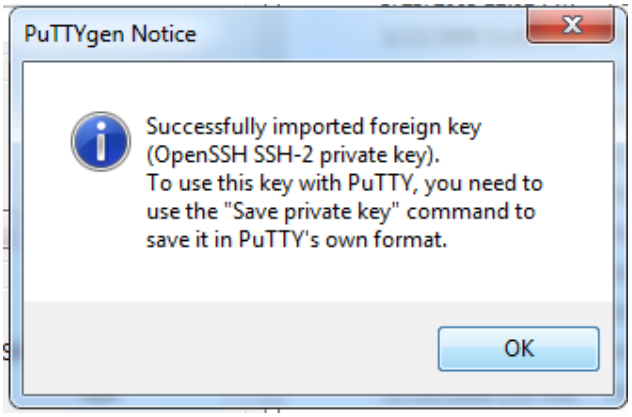
Open puttygen; select "Load".



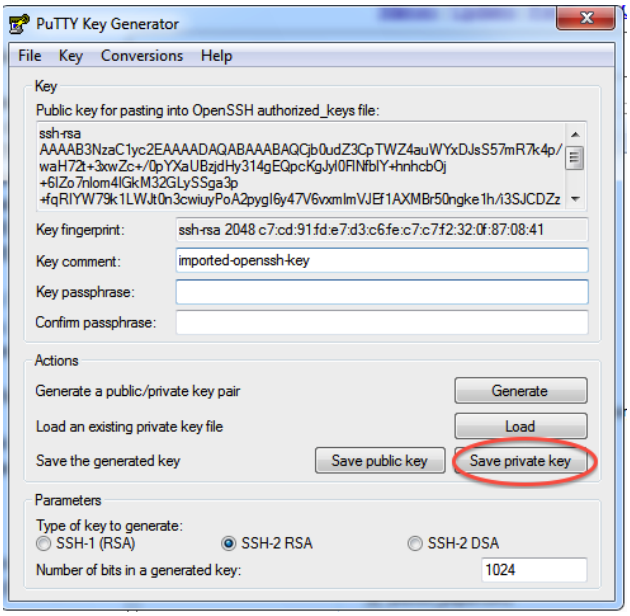
Find and load your '.pem' file; it's probably in your Downloads folder. Note, you have to select 'All files' on the bottom.



Load it.

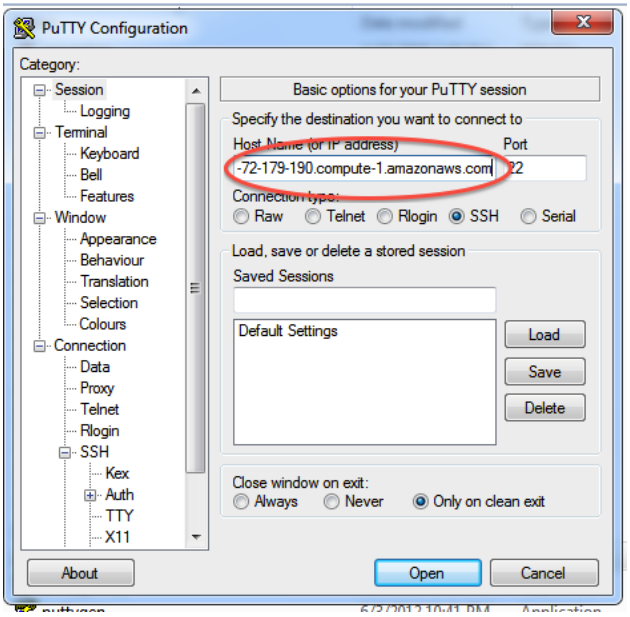


Now, "save private key". Put it somewhere easy to find.

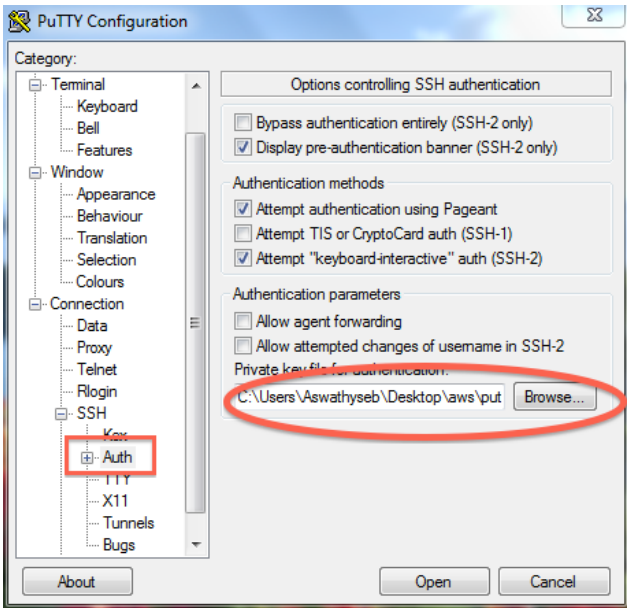


Logging into your EC2 instance with Putty

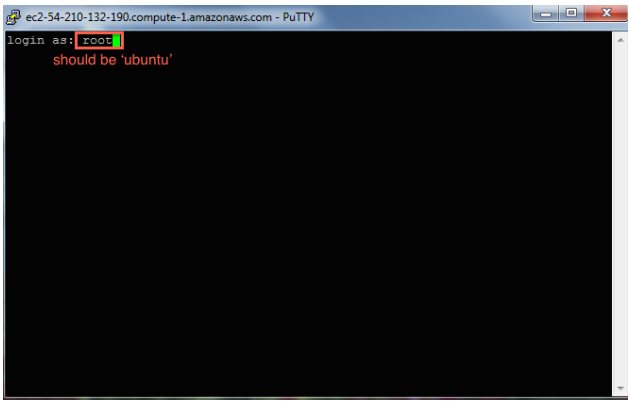
Open up putty, and enter your hostname into the Host Name box.



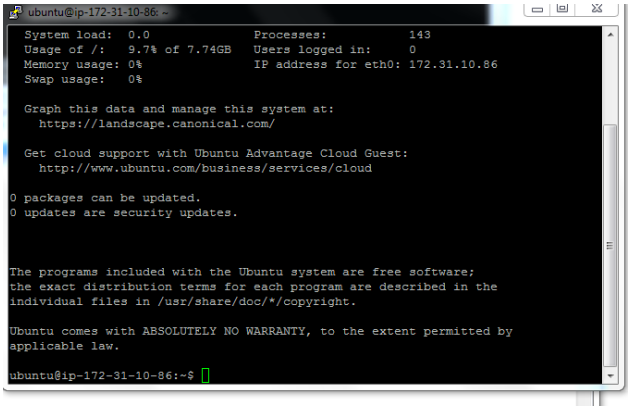
Now, go find the 'SSH' section and enter your ppk file (generated above by puttygen). Then select 'Open'.



Log in as “ubuntu”.



Declare victory!



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.

You should now see a text line that starts with something like `ubuntu@ip-10-235-34-223:~$`. You’re in! Now type:

```
sudo bash
cd /root
```

to switch into superuser mode (see: http://xkcd.com/149/) and go to your home directory.

This is where the rest of the tutorials will start!

If you have Dropbox, you should now visit Installing Dropbox on your EC2 machine.

You might also want to read about Terminating (shutting down) your EC2 instance.

To log out, type:

```
exit
logout
```

or just close the window.

4.1.3 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 'root', 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
```

(but you have to replace the stuff after the '@' sign with the name of the host).

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.

You should now see a text line that starts with something like `ubuntu@ip-10-235-34-223:~$`. You're in! Now type:

```
sudo bash
cd /root
```

to switch into superuser mode (see: http://xkcd.com/149/) and go to your home directory.

This is where the rest of the tutorials will start!

If you have Dropbox, you should now visit Installing Dropbox on your EC2 machine.

You might also want to read about Terminating (shutting down) your EC2 instance.

To log out, type:

```
exit
logout
```

or just close the window.

4.1.4 Installing Dropbox on your EC2 machine

IMPORTANT: Dropbox will sync everything you have to your EC2 machine, so if you are already using Dropbox for a lot of stuff, you might want to create a separate Dropbox account just for the course.

Start at the login prompt on your EC2 machine:

```
sudo bash
cd /root
```

Then, grab the latest dropbox installation package for Linux:

```
wget -O dropbox.tar.gz "http://www.dropbox.com/download/?plat=lnx.x86_64"
```

Unpack it:

```
tar -xvzf dropbox.tar.gz
```

Make the Dropbox directory on /mnt and link it in:

```
mkdir /mnt/Dropbox
ln -fs /mnt/Dropbox /root
```

and then run it, configuring it to put stuff in /mnt:

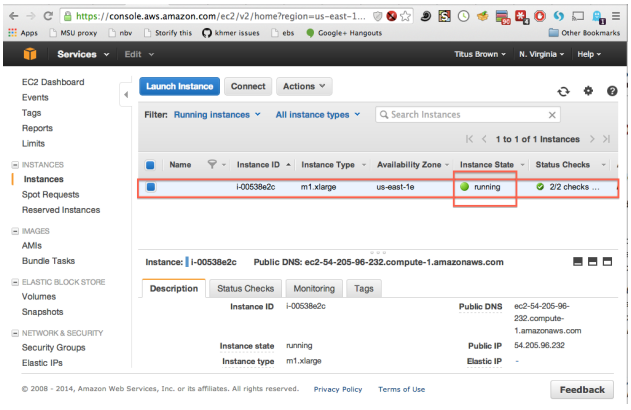
```
HOME=/mnt /root/.dropbox-dist/dropboxd &
```

When you get a message saying "this client is not linked to any account", copy/paste the URL into browser and go log in. Voila!

Your directory '/root/Dropbox', or, equivalently, '/mnt/Dropbox', is now linked to your Dropbox account!

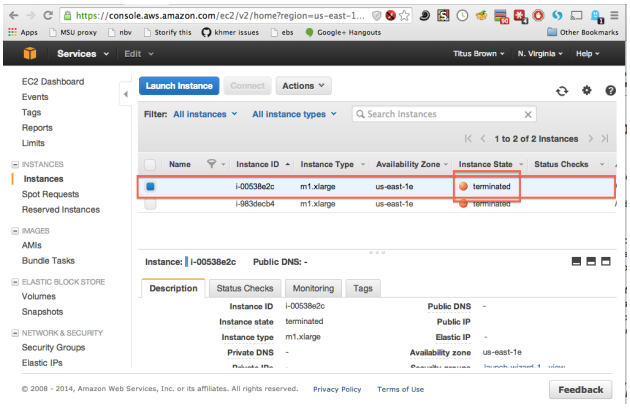
4.1.5 Terminating (shutting down) your EC2 instance

While your instance is running, Amazon will happily charge you on a per-hour basis – check out the pricing for more information. In general, you will want to shut down your instance when you're done with it; to do that, go to your EC2 console and find your running instances (in green).



Then, select one or all of them, and go to the 'Actions...' menu, and then select 'Terminate'. Agree.

After a minute or two, the console should show the instance as "terminated".



4.2 Day 2 – Running BLAST and other things at the command line

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. Make sure you follow the Dropbox instructions, too!

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.

4.2.1 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.

Prepare for this tutorial by working through Start up an EC2 instance, but follow the instructions to start up Starting up a custom operating system instead; use AMI ami-7606d01e.

All of the commands below can and should be copy/pasted rather than re-typed.

Note: on Windows using TeraTerm, you can select the commands in the Web browser, then go to TeraTerm and click your right mouse button to paste. On Mac OS X using Terminal, you can select the commands in the Web browser, use Command-C to copy, and then go the terminal and use Command-V to paste.

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
```

(make sure to hit enter after the paste – sometimes the last line doesn't paste completely.)

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.

Install BLAST

Here, we're using curl to download the BLAST distribution from NCBI; then we're using 'tar' to unpack it into the current directory; and then we're copying the program files into the directory /usr/local/bin, where we can run them from anywhere.

```
cd /root

curl -O ftp://ftp.ncbi.nih.gov/blast/executables/release/2.2.26/blast-2.2.26-x64-linux.tar.gz
tar xzf blast-2.2.26-x64-linux.tar.gz
cp blast-2.2.26/bin/* /usr/local/bin
cp -r blast-2.2.26/data /usr/local/blast-data
```

OK – now you can run BLAST from anywhere!

Again, this is basically what "installing software" means – it just means copying around files so that they can be run, and (in some cases) setting up resources so that the software knows where specific data files are.

Running BLAST

Try typing:

```
blastall
```

You'll get a long laundry list of output, with all sorts of options and arguments. Let's play with some of them.

First! We need some data. Let's grab the mouse and zebrafish RefSeq protein data sets from NCBI, and put them in /mnt, which is the scratch disk space for Amazon machines

```
cd /mnt

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

If you look at the files in the current directory, you should see both files, along with a directory called lost+found which is for system information:

```
ls -l
```

should show you:

```
drwx------ 2 root root   16384 2013-01-08 00:14 lost+found
-rw-r--r-- 1 root root 9454271 2013-06-11 02:29 mouse.protein.faa.gz
-rw-r--r-- 1 root root 8958096 2013-06-11 02:29 zebrafish.protein.faa.gz
```

Both of these files are FASTA protein files (that's what the .faa suggests) that are compressed by gzip (that's what the .gz suggests).

Uncompress them

```
gunzip *.faa.gz
```

and let's look at the first few sequences:

```
head -11 mouse.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.

Let's take those first two sequences and save them to a file. We'll do this using output redirection with '>', which says "take all the output and put it into this file here."

```
head -11 mouse.protein.faa > mm-first.fa
```

So now, for example, you can do 'cat mm-first.fa' to see the contents of that file (or 'less mm-first.fa').

Now let's BLAST these two sequences against the entire zebrafish protein data set. First, we need to tell BLAST that the zebrafish sequences are (a) a database, and (b) a protein database. That's done by calling 'formatdb'

```
formatdb -i zebrafish.protein.faa -o T -p T
```

Next, we call BLAST to do the search

```
blastall -i mm-first.fa -d zebrafish.protein.faa -p blastp
```

This should run pretty quickly, but you're going to get a LOT of output!! What's going on? A few things –

- if you BLAST a sequence against a large database, odds are it will turn up a lot of spurious matches. By default, blastall uses an e-value cutoff of 10, which is very relaxed.
- blastall also reports the first 100 matches, which is usually more than you want.
- a lot of proteins also have trace similarity to other proteins!

For all of these reasons, generally you only want the first few BLAST matches, and/or the ones with a "good" e-value. We do that by adding '-b 2 -v 2' (which says, report only two matches and alignments); and by adding '-e 1e-6', which says, report only matches with an e-value of 1e-6 or better

```
blastall -i mm-first.fa -d zebrafish.protein.faa -p blastp -b 2 -v 2 -e 1e-6
```

Now you should get a lot less text! (And indeed you do...) Let's put it an output file, 'out.txt'

```
blastall -i mm-first.fa -d zebrafish.protein.faa -p blastp -b 2 -v 2 -o out.txt
```

The contents of the output file should look exactly like the output before you saved it into the file – check it out:

```
cat out.txt
```

Converting BLAST output into CSV

Suppose we wanted to do something with all this BLAST output. Generally, that's the case - you want to retrieve all matches, or do a reciprocal BLAST, or something.

As with most programs that run on UNIX, the text output is in some specific format. If the program is popular enough, there will be one or more parsers written for that format – these are just utilities written to help you retrieve whatever information you are interested in from the output.

Let's conclude this tutorial by converting the BLAST output in out.txt into a spreadsheet format, using a Python script. (We're not doing this just to confuse you; this is really how we do things around here.)

First, we need to get the script. We'll do that using the 'git' program

```
git clone https://github.com/ngs-docs/ngs-scripts.git /root/ngs-scripts
```

We'll discuss 'git' more later; for now, just think of it as a way to get ahold of a particular set of files. In this case, we've placed the files in /root/ngs-scripts/, and you're looking to run the script blast/blast-to-csv.py using Python

```
python /root/ngs-scripts/blast/blast-to-csv.py out.txt
```

This outputs a spread-sheet like list of names and e-values. To save this to a file, do:

```
python /root/ngs-scripts/blast/blast-to-csv.py out.txt > /root/Dropbox/out.csv
```

The end file, 'out.csv', should soon be in your Dropbox on your local computer. If you have Excel installed, try double clicking on it.

And that's the kind of basic workflow we'll be teaching you:

1. Download program
2. Download data
3. Run program on data
4. Look at results

...but in many cases more complicated :).

Note that there's no limit on the number of sequences you BLAST, etc. It's just sheer compute speed and disk space that you need to worry about, and if you look at the files, it turns out they're not that big – so it's mostly your time and energy.

This will also maybe help you understand why UNIX programs are so powerful – each program comes with several, or several dozen, little command line "flags" (parameters), that help control how it does its work; then the output is fed into another such program, etc. The possibilities are literally combinatorial.

We're running a Python program 'blast-to-csv.py' above – if you're interested in what the Python program does, take a look at the source code:

https://github.com/ngs-docs/ngs-scripts/blob/master/blast/blast-to-csv.py

Summing up

Command-line BLAST lets you do BLAST searches of any sequences you have, quickly and easily. It's probably the single most useful skill a biologist can learn if they're doing anything genomics-y ;).

Its main computational drawback is that it's not fast enough to deal with some of the truly massive databases we now have, but that's generally not a problem for individual users. That's because they just run it and "walk away" until it's done!

The main practical issues you will confront in making use of BLAST:

- getting your sequence(s) into the right place.
- formatting the database.
- configuring the BLAST parameters properly.
- doing what you want after BLAST!

Other questions to ponder:

- if we're using a pre-configured operating system, why did we have to install BLAST?

4.2.2 Short Read Quality Control

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 try to improve it.

> **Summary**: a sequencing run may contain data of low reliability. It may also contain various contaminants and artificial sequence fragments. Some (but not all) of these problem can be corrected.
>
> **Caution**: Don't apply any type of correction without evaluating the results it produces. In general it is best to be conservative with QC. We are altering the data based on our expectations of what it should be like! The process may also introduce its own biases into the dataset.

Biostar QoD (questions of the day)

QC is one of the most *mis-under-estimated* tasks of NGS data analysis. People assume there is very little to it once they know how to run the tool. The reality is a more complicated than that.

QC also happens to be a pet peeve of mine (Istvan) as demonstrated below in the following Biostar threads (and others):

1. FastQC quality control shootout
2. So What Does The Sequence Duplication Rate Really Mean In A Fastqc Report
3. Revisiting the FastQC read duplication report

Quick Start

The first part of this tutorial will run on your own computer. It assumes that you have Java installed. Download both FastQC and two smaller datasets onto your system

1. http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
2. http://apollo.huck.psu.edu/data/SRR519926_1.fastq.zip
3. http://apollo.huck.psu.edu/data/SRR447649_1.fastq.zip

Run FastQC on each the files from the graphical interface. Let's discuss the output in more detail.

FastQC at the command line

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

We will use a so called `Makefile`, a simple text file that can contains a series of commands that you could otherwise type in yourself. There will be more information on shell programming and automation later. For now think of a `Makefile` as a simple way to pick which commands you can execute yourself. Let's get started. Install *make*:

```
sudo apt-get install make -y
```

You can also investigate the *Makefile* yourself: https://github.com/ngs-docs/angus/blob/2014/files/Makefile-short-read-quality

This tutorial will download datasets. You may want to create a directory (folder) that stores this data:

```
mkdir qc
cd qc
```

We assume that you are running the subsequent scripts from this folder.

Important: Obtain the *Makefile* and save it onto your cloud system:

```
# This is where you get the Makefile
wget https://raw.githubusercontent.com/ngs-docs/angus/2014/files/Makefile-short-read-quality -O Makef
```

You can investigate the file:

```
# Look at the Makefile
# more Makefile or pico Makefile
```

So we now have a *Makefile* and our system can execute this *Makefile* via the *make* command.:

```
make
```

Setup

In our case you have to always specify which section of the *Makefile* do you wish to execute. For example you can type:

```
make setup
```

This will execute the parts of the Makefile that is listed below:

```
        #
        # Run initial setup tasks
        #

        # This directory will contain the executables
        mkdir -p ~/bin

        # Add the ~/bin to the user path
        echo 'export PATH=$PATH:~/bin' >> ~/.bashrc

        # Install the unzip library
        sudo apt-get install -y unzip

        # Change the mount point so it is user writeable
        sudo chmod 777 /mnt

        # Update the installation sources
        sudo apt-get update

        # Install java
        sudo apt-get install -y default-jdk
```

Note that you could also just type in these commands yourself for the same results. The Makefile just automates this.

Software Install

The next step is installing FastQC and Trimmomatic on your instance:

```
make install
```

command will execute the following lines.

```
        #
        # The src folder will contain the downloaded software
        #
        mkdir -p ~/src

        #
        # The bin folder will contain the binaries that you have downloaded
        #
        mkdir -p ~/bin

        #
        # Install Trimmomatic
        #
        curl http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/Trimmomatic-0.32.zip -o ~
        unzip -o ~/src/Trimmomatic-0.32.zip -d ~/src
        ln -fs ~/src/Trimmomatic-0.32/trimmomatic-0.32.jar ~/bin

        #
        # Install FastQC
        #
        curl http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.2.zip -o ~/src/fas
        unzip -o ~/src/fastqc_v0.11.2.zip -d ~/src
        chmod +x ~/src/FastQC/fastqc
        ln -fs ~/src/FastQC/fastqc ~/bin/fastqc
```

Where did stuff go? The downloaded code went into *~/src* the binaries are linked into *~/bin* To test that everything works well type:

```
fastqc -v
```

This will print the version number of *fastqc*

Download Data

Start at your EC2 prompt, then type

```
make download
```

This will execute the following lines. Remember that you could also type these in yourself.

```
        #
        # Data download
        #
        curl apollo.huck.psu.edu/data/SRR.tar.gz -o SRR.tar.gz

        # Unpack the gzipped data
        tar xzvf SRR.tar.gz
```

The FASTQ Format

In class explanation of the format. See a good description at http://en.wikipedia.org/wiki/FASTQ_format

If you don't understand the format, you don't understand the basic premise of the data generation!

Run a FastQC analysis on each dataset:

```
make fastqc
```

would run the commands:

```
        #
        # Run FastQC on every file that we have downloaded.
        #
        fastqc *.fastq
```

This command will generate an HTML file for each file. Copy these files to your dropbox and look at them (a short walkthrough on what each plot means).

Alternatively you can generate the fastqc output directly to your Dropbox like so:

```
fastqc *.fastq -o /mnt/Dropbox
```

Pattern Matching

We can also investigate what the files contain by matching:

```
# Find start codons
grep ATG SRR519926_1.fastq  --color=always | head

# Find a subsequence
grep AGATCGGAAG SRR519926_1.fastq  --color=always | head
```

Pattern matching via expressions is an extremely powerful concept. We'll revisit them later.

Trimming

Note that there are vary large number of tools that perform quality/adapter trimming.

Now, run Trimmomatic to eliminate Illumina adapters from your sequences. First we need to find the adapter sequences:

```
ln -s ~/src/Trimmomatic-0.32/adapters/TruSeq3-SE.fa
```

Tip: You may also want to shorten the command line like so:

```
alias trim='java -jar ~/src/Trimmomatic-0.32/trimmomatic-0.32.jar'
```

You can now invoke the tool just by typing:

```
trim
```

Among the (many) agonizing decisions that you will have to make is what parameters to pick: how big should be my window be, how long should the reads be, what should be the average quality be? What kinds of contaminants do I have. Run, rerun and evaluate. Err on the side of caution.

Trim by quality alone:

```
        #
        # Run the quality trimming.
        #
        java -jar ~/src/Trimmomatic-0.32/trimmomatic-0.32.jar SE SRR447649_1.fastq good.fq SLIDINGWIN
        fastqc good.fq -o /mnt/Dropbox
```

Quality and clipping:

```
        #
        # Run quality trimming and clipping at the same time.
        #
        java -jar ~/src/Trimmomatic-0.32/trimmomatic-0.32.jar PE SRR519926_1.fastq good1.fq bad1.fq S
        fastqc good1.fq -o /mnt/Dropbox
```

Now a homework:

Note: Read the manual for Trimmomatic. Trim the reads in parallel for both readfiles in a sample.

Note: BTW: cool kids have pretty prompts, but you too can be cool, all you need to do is:

```
echo "export PS1='\[\e]0;\w\a\]\n\[\e[32m\]\u@\h \[\e[33m\]\w\[\e[0m\]\n\$ '" >> ~/.bashrc
```

Then relog. Don't ask why this works, it is one of those things that is best left undisturbed.

4.3 Variant calling

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

We'll be using data from one of Rich Lenski's LTEE papers, the one on the evolution of citrate consumption in LTEE.

4.3.1 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 Starting up a custom operating system).

Log in with Windows or from Mac OS X.

4.3.2 Updating the operating system

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.

4.3.3 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
```

Also install samtools:

```
apt-get -y install samtools
```

4.3.4 Download data

Download the reference genome and the resequencing reads:

```
cd /mnt

curl -O http://athyra.idyll.org/~t/REL606.fa.gz
gunzip REL606.fa.gz

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 European Nucleotide Archive (ENA) for this sample: http://www.ebi.ac.uk/ena/data/view/SRR098042.

4.3.5 Do the mapping

Now let's map all of the reads to the reference. Start by indexing the reference genome:

```
cd /mnt

bwa index REL606.fa
```

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

```
bwa aln REL606.fa SRR098038.fastq.gz  > SRR098038.sai
```

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

```
bwa samse REL606.fa SRR098038.sai SRR098038.fastq.gz > SRR098038.sam
```

This file contains all of the information about where each read hits on the reference.

Next, index the reference genome with samtools:

```
samtools faidx REL606.fa
```

Convert the SAM into a BAM file:

```
samtools import REL606.fa.fai SRR098038.sam SRR098038.bam
```

Sort the BAM file:

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

And index the sorted BAM file:

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

4.3.6 Visualizing alignments

At this point you can visualize with samtools tview or Tablet.

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

```
samtools tview SRR098038.sorted.bam REL606.fa
```

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 'rel606:553093<ENTER>' to go to position 553093 in the BAM file.

Use 'q' to quit.

For the Tablet viewer, click on the link and get it installed on your local computer. Then, start it up as an application. To open your alignments in Tablet, you'll need three files on your local computer: `REL606.fa`, `SRR098042.sorted.bam`, and `SRR098042.sorted.bam.bai`. You can copy them over using Dropbox, for example.

4.3.7 Counting 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:

```
gunzip -c SRR098038.fastq.gz | wc
```

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

4.3.8 Calling SNPs

You can use samtools to call SNPs like so:

```
samtools mpileup -uD -f REL606.fa SRR098038.sorted.bam | bcftools view -bvcg - > SRR098038.raw.bcf
```

(See the 'mpileup' docs here.)

Now convert the BCF into VCF:

```
bcftools view SRR098038.raw.bcf > SRR098038.vcf
```

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

```
tail *.vcf
```

Each variant call line consists of the chromosome name (for E. coli REL606, there's only one chromosome - rel606); the position within the reference; an ID (here always '.'); the reference call; the variant call; and a bunch of additional information about

Again, you can use 'samtools tview' 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'.

You can read more about the VCF file format here.

4.3.9 Questions/discussion items

Why so many steps?

4.4 Assembling E. coli sequences with Velvet

The goal of this tutorial is to show you the basics of assembly using the Velvet assembler.

We'll be using data from Efficient de novo assembly of single-cell bacterial genomes from short-read data sets, Chitsaz et al., 2011.

4.4.1 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 Starting up a custom operating system).

Log in with Windows or from Mac OS X.

Logging in

Log in and type:

```
sudo bash
```

to change into superuser mode.

4.4.2 Updating the operating system

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.

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
```

and install the Velvet assembler:

```
cd /root
curl -O http://www.ebi.ac.uk/~zerbino/velvet/velvet_1.2.10.tgz
tar xzf velvet_1.2.10.tgz
cd velvet_1.2.10
make MAXKMERLENGTH=51
cp velvet? /usr/local/bin
```

as well as Quast, software for evaluating the assembly against the known reference:

```
cd /root
curl -O -L https://downloads.sourceforge.net/project/quast/quast-2.3.tar.gz
tar xzf quast-2.3.tar.gz
```

Getting the data

Now, let's create a working directory:

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

Download some E. coli data. The first data set (ecoli_ref-5m-trim.fastq.gz) is the trimmed PE data sets from the other day (see Short Read Quality Control), and the second data set is a specially processed data set using digital normalization that will assemble quickly.

```
curl -O https://s3.amazonaws.com/public.ged.msu.edu/ecoli_ref-5m-trim.fastq.gz
curl -O https://s3.amazonaws.com/public.ged.msu.edu/ecoli-reads-5m-dn-paired.fa.gz
```

Running an assembly

Now... assemble the small, fast data sets, using the Velvet assembler. Here we will set the required parameter k=21:

```
velveth ecoli.21 21 -shortPaired -fasta.gz ecoli-reads-5m-dn-paired.fa.gz
velvetg ecoli.21 -exp_cov auto
```

Check out the stats for the assembled contigs for a cutoff of 1000:

```
python /usr/local/share/khmer/sandbox/assemstats3.py 1000 ecoli.*/contigs.fa
```

Also try assembling with k=23 and k=25:

```
velveth ecoli.23 23 -shortPaired -fasta.gz ecoli-reads-5m-dn-paired.fa.gz
velvetg ecoli.23 -exp_cov auto

velveth ecoli.25 25 -shortPaired -fasta.gz ecoli-reads-5m-dn-paired.fa.gz
velvetg ecoli.25 -exp_cov auto
```

Now check out the stats for the assembled contigs for a cutoff of 1000:

```
python /usr/local/share/khmer/sandbox/assemstats3.py 1000 ecoli.*/contigs.fa
```

(Also read: What does k control in de Bruijn graph assemblers?.)

Comparing and evaluating assemblies - QUAST

Download the true reference genome:

```
cd /mnt/assembly
curl -O https://s3.amazonaws.com/public.ged.msu.edu/ecoliMG1655.fa.gz
gunzip ecoliMG1655.fa.gz
```

and run QUAST:

```
/root/quast-2.3/quast.py -R ecoliMG1655.fa ecoli.*/contigs.fa
```

Note that here we're looking at *all* the assemblies we've generated.

Now look at the results:

```
more quast_results/latest/report.txt
```

The first bits to look at are Genome fraction (%) and # misassembled contigs, I think.

Searching assemblies – BLAST

Install BLAST:

```
cd /root

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

Build BLAST databases for the assemblies you've done:

```
cd /mnt/assembly

for i in 21 23 25
do
   extract-long-sequences.py -o ecoli-$i.fa -l 500 ecoli.$i/contigs.fa
   formatdb -i ecoli-$i.fa -o T -p F
done
```

and then let's search for a specific gene – first, download a file containing your protein sequence of interest:

```
curl -O http://athyra.idyll.org/~t/crp.fa
```

and now search:

```
blastall -i crp.fa -d ecoli-21.fa -p tblastn -b 1 -v 1
```

Questions and Discussion Points

Why do we use a lower cutoff of 1kb for the assemstats3 links, above? Why not 0?

Followup work

Try running an assembly of the larger read data set:

```
velveth ecoli-full.31 31 -short -fastq.gz ecoli_ref-5m-trim.fastq.gz
velvetg ecoli-full.31 -exp_cov auto
```

4.5 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:

4.5.1 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.

4.6 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.

4.7 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.

This tutorial may be clearer to view it in markdown https://github.com/ngs-docs/angus/blob/2014/R_Introductory_tutorial_2014.md>

4.8 R Tutorial for NGS2014

In this tutorial I will introduce `R`, a programming language that is currently the *lingua franca* for data analysis (although `python` has many powerful data analysis tools through `numpy`, `scipy` and other libraries.

4.8.1 What is R?

R is a bit of funny programming language, with a funny history. While many researchers with a statistical bent really appreciate aspects of the syntax, and how it works, there are a great number of idiosyncracies, and "features" of the language that drive many programmers crazy. It is also known to be a bit of a memory hog, so big data sets **can** be an issue to work with if you are not aware of the appropriate approaches to deal with. Despite this, there are some features of `R` that make it great for doing data analyses of all kinds, and there are a huge number of libraries of tools covering all aspects of statistics, genomics, bioinformatics, etc... In many cases new methods are associated with at least a basic implementation in `R` if not a full blown library.

4.8.2 Installing R

Many of you may already have R installed on your local machine. If you would like to install R on your computer just go to http://www.r-project.org/ click download and follow for your OS. For linux machines, it is best to instead use your package manager (`yum` or `apt-get` for instance).

If you would like to run this on an Amazon EC2 instance, set up and log into your instance as you did in the earlier tutorial.

R generally does not come pre-installed on most Linux distributions including Ubuntu (Debian) Linux, which we are using on our Amazon EC2 instance but it is very easy to install:

```
apt-get install r-base
```

You may wish to link and mount your instance to dropbox as we did in the earlier tutorials.

It is also important to point out that `R` has been packaged in a number of different easy to use ways. For instance, many scientists really like R-studio which is also a free easy to use IDE that makes it easier for you to implement *version control* and do reproducible research. I personally do not use `R` via R-studio, but feel free to try it out on your local machine if you want to.

For the remainder of the tutorial I am assuming that we will all be running `R` on your Amazon EC2 instance though.

R

4.8.3 What is R, really....

While many of you are used to seeing `R` through some GUI (whether on windows, OSX,etc).. It is fundamentally just a command line program like what you have been using at the command line or when you call `python`, etc.. The GUI versions just add some additional functionality.

At your shell prompt just type:

```
R
```

You now have a new command prompt (likely >) which means you are in R. Indeed you probably see something that looks like:

```
##  R version 3.0.1 (2013-05-16) -- "Good Sport"
##  Copyright (C) 2013 The R Foundation for Statistical Computing
## Platform: x86_64-apple-darwin10.8.0 (64-bit)

##  R is free software and comes with ABSOLUTELY NO WARRANTY.
##  You are welcome to redistribute it under certain conditions.
##  Type 'license()' or 'licence()' for distribution details.
##
##   Natural language support but running in an English locale
##
##  R is a collaborative project with many contributors.
##  Type 'contributors()' for more information and
##  'citation()' on how to cite R or R packages in publications.

##  Type 'demo()' for some demos, 'help()' for on-line help, or
## 'help.start()' for an HTML browser interface to help.
##  Type 'q()' to quit R.
##
##  >
```

I will quickly (on the Mac OSX version) show you some of navigating the GUI.

4.8.4 How to close R

To quit R:

```
q()
```

NOTE to MAC (OSX) users: This may no longer work (R V2.11.+) from the Mac R GUI... See below in OSX specific notes..

NOTE: For the moment when it asks you to save workspace image, say no.

4.8.5 R Basics

Ok, return to R (type R at unix command prompt) so we can begin the tutorial.

I stole this table from *Josh Herr's ''R'* tutorial <https://github.com/jrherr/quick_basic_R_tutorial/blob/master/R_tutorial.md>`__:

Table 1 - Important R Symbols

| Symbol | Meaning / Use | |:————-:|:————-:| | > | Prompt for a new command line, you do NOT type this, it appears in the console | | + | Continuation of a previously started command, you do NOT type this | | # | Comment in code; R does not "see" this text – important to explain your computational train of thought | | <- or = (= not prefered) | set a value to a name |

Note: Comments in R are performed by using #. Anything followed by the number sign is ignored by R.

For the purposes of this tutorial we will be typing most things at the prompt. However this is annoying and difficult to save. So it is generally best practice to write things in a script editor. The R GUIs have some (OSX version with syntax highlighting as does gedit). So does R Studio. If we have time I will show you how to set up syntax highlighting on nano as well.

4.8.6 R as a calculator

Let's start by adding `2+2`.

```
2 + 2
```

This will produce the output:

```
## [1] 4
```

Since `R` is a programming language we can easily generate variables to store all sorts of things.

```
y = 2
```

When you create a variable like this, it does not provide any immediate output.

However when you type `y` and press return:

```
y
```

You will see the output:

```
## [1] 2
```

Note that the `[1]` is just an index for keeping track where the answer was put. It actually means that it is the first element in a vector (more on this in a few minutes).

R is case SENSITIVE. That is `y` & `Y` are not the same variable.

```
y
```

```
## [1] 2
```

```
Y
```

```
## Error: object 'Y' not found
```

We can generate variables and do computations with them.

```
x = 3
x + y
```

```
## [1] 5
```

```
z <- x + y
```

You will notice that sometimes I am using '=' and sometimes '<-'. These are called *assignment operators*. In most instances they are equivalent. The '<-' is preferred in R, and can be used anywhere. You can look at the help file (more on this in a second) to try to parse the difference

```
`?`("=")
```

We may want to ask whether a variable that we have computed equals something in particular for this we need to use '==' not '=' (one equals is an assignment, two means 'equal to')

```
x == 3
```

```
## [1] TRUE
```

```
x == 4
```

```
## [1] FALSE
```

```
x == y
```

```
## [1] FALSE
```

What happens if we write?

```
x = y
```

We have now assigned the current value of `y` (2) to `x`. This also shows you that you can overwrite a variable assignment. This is powerful, but also means you need to be very careful that you are actually using the value you think you are.

Operator `*` for multiplication.

```
2 * 3
```

```
## [1] 6
```

For division `/`.

```
6/3
```

```
## [1] 2
```

Operator for exponents `^`. Can also use `**`

```
3^2
```

```
## [1] 9
```

```
3^2  # same as above
```

```
## [1] 9
```

You can use `^0.5` or `sqrt()` function for square roots.

```
9^0.5
```

```
## [1] 3
```

```
sqrt(9)
```

```
## [1] 3
```

to raise something to e^some exponent

```
exp(2)  # this is the performing e^2
```

```
## [1] 7.389
```

For natural log (base e), use `log()`.

```
log(2.7)
```

```
## [1] 0.9933
```

To raise to an arbitrary base use `log( , base)` like the following:

```
log(2.7, 10)  # base 10
```

```
## [1] 0.4314
```

You can also use `log10()` or `log2()` for base 10 or base 2.

While these are all standard operators (except `<-`) common to most programming languages, it is a lot to remember.

A bit on data structures in R

`R` **is vectorized**. It can do its operations on vectors. Indeed there is no data structure in `R` for scalar values at all. This means when you assign `y <- 2` you are in fact generating a vector of length 1.

```
a <- c(2, 6, 4, 5)
b <- c(2, 2, 2, 1)
```

The `c` is short for concatenate you can add the elements of the vectors together. Keep in mind *how* it is doing its addition this way (element-by-element).

```
a + b
```

```
## [1] 4 8 6 6
```

Or multiply the elements of the vector together (this is **NOT** vector multiplication, but element-by-element multiplication. For vector multiplication (inner & outer products) there are specific operators, i.e. `%*%`.

```
a * b
```

```
## [1]  4 12  8  5
```

If you want to take vectors `a` and `b` (imagine these are each columns of read counts for RNAseq from different samples) and put them together in a single vector you use the `c()` (concatenate) function. I just realized that I am using `c` as a variable name (for the new vector), and the function is `c()`. This is entirely by accident and they have no relation to one another.

```
c <- c(a, b)
```

How might you make a vector that repeats vector `a` 3 times?

MANY MANY operations can be vectorized, and R is really good at it!!! Vectors are very useful as way of storing data relevant for all sorts of data analysis.

4.8.7 GETTING HELP in R

There are a number of places where you can get help with R directly from the console. For instance, what if we want to get help on the function `lm()` which is what is used to fit all sorts of *linear models* like regression, ANOVA, ANCOVA etc..

```
?lm
```

This brings up a description of the function 'lm'

For operators we need to use quotes

```
?'*' # for help for operators use quotes
```

sometimes you will need to use `help.search('lm')` This brings up all references to the `lm()` function in packages and commands in `R`. We will talk about packages later.

You can also use `RSiteSearch('lm')`. This is quite a comprehensive search that covers R functions, contributed packages and R-help postings. It is very useful but uses the web (so you need to be online).

You can also use the html version of help using `help.start()`.

Or if using the GUI, just go to the help menu!

4.8.8 Simple functions in base R

`R` has many functions, and indeed everything (including the operators) are functions (even though the details of the function call are sometimes hidden). As we will see it is very easy to also write new functions.

Here are some examples of the useful functions in base `R`.

You can find out the length of the new vector using `length()`:

```
length(c)
```

```
## [1] 8
```

`length()` is an example of a pre-built function in R. Most things in R revolve around using functions to do something, or extract something. We will write our own simple functions soon.

Here are some more common ones that you may use

```
mean(c)
```

```
## [1] 3
```

```
sum(c)
```

```
## [1] 24
```

standard deviation

```
sd(c)
```

```
## [1] 1.773
```

variance

```
var(c)
```

```
## [1] 3.143
```

Correlation coefficient.

```
cor(a, b)
```

This gets the Pearson correlation (there are options to change this to other types of correlations, among the arguments for this function.....).

```
## [1] -0.2928
```

Say we want to keep the mean of `c` for later computation we can assign it to a variable

```
mean_c <- mean(c)
```

We can look at the underlying code of the function (although some times it is buried, in these cases).

We can use a function like `mean()` to add up all of the elements of the vector.

We can also join the two vectors together to make a matrix of numbers.

```
d <- cbind(a, b)
d
```

```
##      a b
## [1,] 2 2
## [2,] 6 2
## [3,] 4 2
## [4,] 5 1
```

We can double check that we really made a matrix:

```
is.matrix(d)
```

This sets up a 'Boolean'. In other words when we ask 'is d a matrix'? it answers TRUE or FALSE.

```
## [1] TRUE
```

```
mode(d)
```

```
## [1] "numeric"
```

```
class(d)
```

```
## [1] "matrix"
```

While the mode of d is still numeric (the most basic "atomic" type of the data), the class of the object we have created is a matrix.

Exercise: Make a new vector q that goes a,b,a*b

4.8.9 Objects in R, classes of objects, mode of objects.

R is an object-oriented language. Everything in R is considered an object. Each object has one or more attributes (which we do not generally need to worry about, but useful for programming.) Most objects in R have an attribute which is the 'class' of the object, which is what we will usually care about. R has a bunch of useful classes for statistical programming. `bioconductor` also has expanded the classes for objects of use for bioinformatics and genomics. We will see these more this afternoon and in the tutorials next week.

The most basic (atomic) feature of an object. NOTE this does not mean the 'mode' of a distribution.

```
mode(c)
```

```
## [1] "numeric"
```

```
class(c)  # class of the object
```

```
## [1] "numeric"
```

In this case for the vector `c` the mode and class of `c` is the same. This is not always going to be the case as we see below.

```
# typeof(c) # internal representation of type, rarely of interest
```

Let's look at some other objects we have generated.

```
mode(mean_c)
```

```
## [1] "numeric"
```

```
class(mean_c)  #
```

```
## [1] "numeric"
```

Despite what we have seen up to now, mode and class are not the same thing. mode tells the basic 'structures' for the objects. integer, numeric (vector), character, logical (TRUE,FALSE) are examples of the atomic structures.

There are many different classes of objects each with their own attributes. The basic ones that we will see are numeric, character, matrix, data.frame, formula, array, list & factor. It is relatively straightforward (but we will not discuss it here) to extend classes as you need.

We can also make vectors that store values that are not numeric.

```
cities <- c("Okemos", "E.Lansing", "Toronto", "Montreal")
class(cities)
```

```
## [1] "character"
```

```
mode(cities)
```

```
## [1] "character"
```

Let's use one of the built-in functions that we used before to look at the "length" of `cities`.

```
length(cities)
```

```
## [1] 4
```

This tells us how many strings we have in the object 'cities' not the length of the string! To get the number of characters in each string we use `nchar()`.

```
nchar(cities)  # This tells us how many characters we have for each string.
```

```
## [1] 6 9 7 8
```

So if we just do this

```
q = "okemos"
length(q)
```

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

```
nchar(q)
```

```
## [1] 6
```

Exercise: How would you compute the total number of characters for all of cities in the object `cities`?

Let's create a second vector storing the names of rivers in each city.

```
rivers <- c("Red Cedar", "Red Cedar", "Don Valley", "Sainte-Laurent")
cities_rivers <- cbind(cities, rivers)
cities_rivers
```

```
##      cities      rivers
## [1,] "Okemos"    "Red Cedar"
## [2,] "E.Lansing" "Red Cedar"
## [3,] "Toronto"   "Don Valley"
## [4,] "Montreal"  "Sainte-Laurent"
```

```
class(cities_rivers)
```

```
## [1] "matrix"
```

```
mode(cities_rivers)
```

```
## [1] "character"
```

In this above example we have made a matrix, but filled with characters, not numerical values.

Another type of object we will need for this workshop (well a variant of it) is called formula Not surprisingly this is used generally to generate a formula for a statistical model we want to fit.

```
model_1 <- y ~ x1 + x2 + x1:x2
model_1
```

This is just the model formula, and we HAVE NOT FIT ANY MODEL YET!!!!!! It just tells us the model we want to fit. That is the object `model_1` has not yet been 'evaluated'.

```
## y ~ x1 + x2 + x1:x2
## <environment: 0x100b504b0>
```

```
# typeof(model_1)
class(model_1)
```

```
## [1] "formula"
```

```
terms(model_1)  # also see all.names() and all.vars
```

```
## y ~ x1 + x2 + x1:x2
## attr(,"variables")
## list(y, x1, x2)
## attr(,"factors")
##    x1 x2 x1:x2
## y   0  0     0
## x1  1  0     1
## x2  0  1     1
## attr(,"term.labels")
## [1] "x1"    "x2"    "x1:x2"
## attr(,"order")
## [1] 1 1 2
## attr(,"intercept")
## [1] 1
## attr(,"response")
## [1] 1
## attr(,".Environment")
## <environment: 0x100b504b0>
```

Let's make a new vector that will store some read counts of transcript expression values (from a single transcript). The first four are from samples of a "wild type" genotype and the second four samples from a "mutant" genotype.

```
counts_transcript_a <- c(250, 157, 155, 300, 125, 100, 153, 175)
```

We may be interested in comparing and contrasting these. So we need to make a variable that stores the information on the two different genotypes. We do this using one of the underlying classes in R, called factors.

We will make a new variable for the mutant and wild type using the gl() (generate levels) function. There are other ways of generating levels for a categorical treatment variable (and R usually can figure this out), but this will work for here.

```
genotype <- gl(n=2, k=4, labels = c("wild_type", "mutant"))
```

This is our first time using arguments within a function.for the gl() the n=2 means we have two treatment levels. k=4 means we 4 samples within each.

One obvious thing we might want to do is make a single object with both variables (genotype, and counts_transcript_a).

Your first thought (given that we just did this a few minutes ago) might be to make a matrix from this.

```
expression_data <- matrix(counts_transcript_a, genotype)
```

But as you see you gget an error message

```
## Error in matrix(counts_transcript_a, genotype) :
##  non-numeric matrix extent
```

Why? Because for an object of class matrix all of the atomic types in it need to be the same. Basically we can not combine numeric and factors.

The way we do this is to instead create a data.frame. A data.frame is a particular representation of another type of object (list) that allows for the storage of heterogeneous data types. Effectively you can think about it like a spreadsheet for purposes of analysis (for now anywaus.

```
expression_data <- data.frame(counts_transcript_a, genotype)
expression_data
```

When you import most of your data, and do most analyses, you will more often than not be using a data frame.

4.8.10 Workspaces, and objects in them

R stores variables, datafiles, functions, vectors, etc in what is called the Workspace. This contains all of the items that you can access directly within your R session. You can list all of the objects in your # workspace using:

```
ls()
```

```
##  [1] "a"              "b"              "c"              "cities"
##  [5] "cities_rivers" "d"              "mean_c"         "model_1"
##  [9] "q"              "rivers"         "x"              "y"
## [13] "z"
```

If you want to remove a particular variable (say x) use the rm() function

```
rm(x)
```

you could remove multiple objects

```
rm(x, y, z)
```

```
## Warning: object 'x' not found
```

If you want to remove all of the objects in your workspace rm(list = ls()). We will learn what this means later, but basically we are making a list that contains all of the objects found by performing ls(), and then removing everything in that list.

Some people like to save their workspaces, not only because it contains all of the commands they have written, but also all of the objects they have created during that session. I personally do not do this unless I have created objects that have taken a long time to compute. Instead I just save the scripts I write.

However if you write your commands directly at the console (like we have been doing, but really you should not do) without a script editor, you should save your workspaces.

```
save.image('file_name')
```

Which will save it to your current working directory (you can find that using `getwd()`).

If you want to load it again

```
load('file_name.RData')
```

You will need to have the correct working directory set, which I will show you how to do shortly.

4.8.11 SCRIPT!

Writing everything at the console can be a bit annoying, so we will use a script editor.

In Mac OS X I personally find the built-in script editor useful You can highlight the text in the script editor and press command (apple) + return to send it to the R console. Or place the cursor at the end of the line that you want to submit to R with command+ return. It also provides syntax highlighting, and shows the syntax & options for functions.

However, for those of you are under the spell of Bill Gates..... While the basic script editor for windows does not have much functionality, many people have written excellent script editors. The base script editor in windows will submit a line with ctrl-R(???). There are many windows script editors with syntax highlighting (such as Tinn-R).

For a list of some http://www.sciviews.org/_rgui/ In general we will save R scripts with the extension `.R`

Also there is R-studio.

let's type something into our new script

```
x <- c(3, 6, 6, 7)
```

now highlight that line and press ctrl+r (windows), or apple key + return (mac). This should send the highlighted portion to R.

```
x <- c(2, 2, 2, 2)
y <- c(3, 3, 3, 3)
z <- cbind(x, y)
z
```

```
##      x y
## [1,] 2 3
## [2,] 2 3
## [3,] 2 3
## [4,] 2 3
```

We have also just used a new function, `cbind`, which stands for column bind. This will create a new object stitching them together as columns.

4.8.12 Writing our own functions in R

We have now used a few built in functions in R (there are many). Anything where you use `()` is a function in `R`. Like I mentioned, pretty much everything you do in `R` is actually a call to a function.

However, we will often want to compute something for which there is no pre-built function. Thankfully it is very easy to write our own functions in R. You should definitely get in the habit of doing so.

Functions have the following format:

```
aFunction <- function(input variable 1, input variable 2, argument, etc...) {expressions to calculate
```

This is abstract so let me give you a real example. For our read counts, we want to compute the standard error of the mean (a measure of sampling uncertainty), which is ~equal to the sd/sqrt(sample size). How might we do it?

We want to compute it for the numeric vector of read counts.

```
counts_transcript_a
```

We could do it by hand

```
sd_a <- sd(counts_transcript_a)
sample_a <- length(counts_transcript_a)
sd_a/sqrt(sample_a)
```

```
## [1] 23.35857
```

(notice that we the last value was printed, as we did not store it in a variable).

Or we could do it in one line

```
sd(a)/sqrt(length(a))  # notice the function within a function
```

```
## [1] 23.35857
```

But we can also do it so that we can use any vector input we wanted by writing a function.

```
StdErr <- function(vector) {
    sd(vector)/sqrt(length(vector))
}
```

Now type `StdErr`

```
StdErr
```

```
## function(vector) {
##     sd(vector)/sqrt(length(vector))
## }
## <environment: 0x100b504b0>
```

This just repeats the function. If you want to edit the function just type `edit(StdErr)`.

Let's use our new function

```
StdErr(counts_transcript_a)
```

```
## [1] 23.35857
```

But now we can use this for any set of samples that we need to calculate the SEM. In this case transcript 'b'.

```
counts_transcript_b <- c(75, 85, 82, 79, 77, 83, 96, 62)
StdErr(counts_transcript_b)
```

```
## [1] 3.414452
```

Exercise: Write your own function to do something simple, like calculate the co-efficient of variation (CV) which is the sd/mean.

It takes some practice but learning to write small discrete functionscan be extremely helpful for R.

One thing to keep in mind, is that it is very easy to call one function from within another. It is generally considered good practice to write functions that do one thing, and one thing only. It is way easier to find problems (debug).

One of the great things that we can (and need to do) often is to compute the mean (or SEM, etc..) for each transcript by sample. We can do this for the data set we made

```
expression_data
```

I will not explain it at the moment, but we can use one of the apply like functions to compute the mean and SEM for each genotype (works the same whether it is 2 or 2000).

```
with(expression_data, tapply(X=counts_transcript_a, INDEX=genotype, FUN=mean))
```

```
## wild_type    mutant
##  215.50    138.25
```

And then for the SEM

```
with(expression_data, tapply(X=counts_transcript_a, INDEX=genotype, FUN=StdErr))
```

```
# wild_type    mutant
#  35.83876  16.34715
```

The `with()` just makes it a bit easier to utilize the variables within the `expression_data` object. There are a number of other ways of doing it, primarily using the `$` (extract) operator (for lists including data frames). You will also see people use the `attach()`. Avoid using attach at all costs.

4.8.13 Using source() to load your functions

One of the great things about writing simple functions, is that once you have them working, you can keep using them over and over. However, it is generally a pain(and bad practice) to have to include the text of the function in every script you write (what if there is a bug...). Instead, R has a function source() which allows you to 'load' a script that contains functions you have written (and other options you may want), so that you can use them.

We will likely see `source()` in later tutorials, so watch for it.

4.8.14 Regular Sequences

Sometimes we want regular sequences or to create objects of repeated numbers or characters. R makes this easy.

If you want to create regular sequences of integers by units of 1

```
one_to_20 <- 1:20
one_to_20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
twenty_to_1 <- 20:1
twenty_to_1
```

```
##  [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

for other more complicated sequences, use the `seq()` function

```
seq1 <- seq(from = 1, to = 20, by = 0.5)
seq1
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5
## [29] 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0
```

or

```
seq1 <- seq(1, 20, 0.5)
seq1
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5
## [29] 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0
```

This shows that for default options (in the correct order) you do not need to specify things like 'from' or 'by'

Exercise: Make a sequence from -10 to 10 by units of 2

What if you want to repeat a number or character a set number of times?

```
many_2 <- rep(2, times = 20)
```

Works for characters as well

```
many_a <- rep("a", times = 10)
```

We can even use this to combine vectors

```
seq_rep <- rep(20:1, times = 2)
seq_rep
```

```
##  [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1 20 19 18
## [24] 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

What if you wanted to repeat a sequence of numbers (1,2,3) 3 times?

```
rep_3_times <- rep(c(1, 2, 3), times = 3)
# or
rep(1:3, times = 3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

What if we wanted to perform this to create a matrix

```
matrix(rep(20:1, 4), 20, 4)
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]   20   20   20   20
##  [2,]   19   19   19   19
##  [3,]   18   18   18   18
##  [4,]   17   17   17   17
##  [5,]   16   16   16   16
##  [6,]   15   15   15   15
##  [7,]   14   14   14   14
##  [8,]   13   13   13   13
##  [9,]   12   12   12   12
## [10,]   11   11   11   11
## [11,]   10   10   10   10
## [12,]    9    9    9    9
## [13,]    8    8    8    8
## [14,]    7    7    7    7
## [15,]    6    6    6    6
```

```
## [16,]    5    5    5    5
## [17,]    4    4    4    4
## [18,]    3    3    3    3
## [19,]    2    2    2    2
## [20,]    1    1    1    1
```

4.8.15 Indexing, extracting values and subsetting from the objects we have created

Often we will want to extract certain elements from a vector, list or matrix. Sometimes this will be a single number, sometimes a whole row or column.

We index in R using `[ ]` (square brackets).

NOTE `R` indexes starting with 1, not 0!

```
a <- 1:20
b <- 5 * a
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
b
```

```
##  [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85
## [18]  90  95 100
```

```
length(a)
```

```
## [1] 20
```

```
length(b)
```

```
## [1] 20
```

If we want to extract the 5th element from `a`.

```
a[5]
```

```
## [1] 5
```

If we want to extract the 5th and 7th element from 'b'

```
b[c(5, 7)]
```

```
## [1] 25 35
```

IF we want to extract the fifth through 10th element from 'b'

```
b[5:10]
```

```
## [1] 25 30 35 40 45 50
```

How about if we want all but the 20th element of 'a'?

```
a[-20]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

Indexing can also be used when we want all elements greater than (less than etc...) a certain value. Under the hood this is generating a logical/boolean (T vs. F).

```
b[b > 20]
```

```
##  [1]  25  30  35  40  45  50  55  60  65  70  75  80  85  90  95 100
```

Or between certain numbers

```
b[b > 20 & b < 80]
```

```
##  [1] 25 30 35 40 45 50 55 60 65 70 75
```

Exercise: generate a vector with 20 elements create a 'sub' vector that has elements 1:5, 16:20 create a 'sub' vector with odd elements 1,3,5,...,19.

```
c <- a + b
q_matrix <- cbind(a, b, c)
q_matrix
```

`cbind()` 'binds' column vectors together into a matrix (also see `rbind()`).

```
##         a   b   c
##  [1,]  1   5   6
##  [2,]  2  10  12
##  [3,]  3  15  18
##  [4,]  4  20  24
##  [5,]  5  25  30
##  [6,]  6  30  36
##  [7,]  7  35  42
##  [8,]  8  40  48
##  [9,]  9  45  54
## [10,] 10  50  60
## [11,] 11  55  66
## [12,] 12  60  72
## [13,] 13  65  78
## [14,] 14  70  84
## [15,] 15  75  90
## [16,] 16  80  96
## [17,] 17  85 102
## [18,] 18  90 108
## [19,] 19  95 114
## [20,] 20 100 120
```

(What happens if we ask for the length of q.matrix?...)

```
length(q.matrix)
```

We can instead ask for number of rows or columns

```
nrow(q_matrix)
```

```
## [1] 20
```

```
ncol(q_matrix)
```

```
## [1] 3
```

Or just use `dim(q_matrix)` (dimensions) to get both # rows and # columns. R always specifies in row by column format.

```
## [1] 20  3
```

Say we want to extract the element from the 3rd row of the second column (b)?

```
q_matrix[3, 2]
```

```
##  b
## 15
```

How about if we want to extract the entire third row?

```
q_matrix[3, ]
```

```
##  a  b  c
##  3 15 18
```

We can also pull things out by name

```
q_matrix[, "c"]
```

```
##  [1]   6  12  18  24  30  36  42  48  54  60  66  72  78  84  90  96 102
## [18] 108 114 120
```

This is an example of indexing via 'key' instead of numerical order.

The at `@` is used to extract the contents of a slot in an object.. We will not use it much for this class, but it is essential for object oriented programming in R (S4) objects. objectName@slotName.

More often we will use the dollar sign `$`, which is used to extract elements of an object of class list (including data frames).. We will use this a lot to extract information from objects (scuh as information from our models, like coefficients) object.name$element.name.

For more information `?'$'`.

4.8.16 Where to go from here?

There are a huge number of resources for R. Everyone has favorite tutorials and books. Here are but a few.

I have a few screencasts that you can access. I also have a number of tutorials. I have way more R resources for a graduate class I teach in computational statistical approaches which I am happy to share as well.

The R site also has access to numerous tutorials and books or other documents.

For more advanced programming check out Hadley Wickham's online book.

Here is a reasonably decent R wikibook

I really like the book art of R programming.

4.8.17 A few advanced topics... For your own amusement (not nescessary for now, but helps for more advanced R programming).

Objects have attributes. The one we have thought about most is the class of the object, which tells us (and R) how to think about the object, and how it can be used or manipulated (methods). We have also looked at dim() which is another attribute Here is a list of common ones: class, comment, dim, dimnames, names, row.names and tsp.

We can set attributes of objects in easy ways like

```
x <- 4:6
names(x) <- c("observation_1", "observation_2", "observation_3")
x
```

```
## observation_1 observation_2 observation_3
##             4             5             6
```

You can see the attributes in a bunch of ways

```
str(x)
```

```
##  Named int [1:3] 4 5 6
##  - attr(*, "names")= chr [1:3] "observation_1" "observation_2" "observation_3"
```

```
attributes(x)
```

```
## $names
## [1] "observation_1" "observation_2" "observation_3"
```

Same as above, but we will be able to use this to set attributes of the object x as well

```
attr(x, "names")
```

```
## [1] "observation_1" "observation_2" "observation_3"
```

```
y <- cbind(1:5, 11:15)
attributes(y)
```

```
## $dim
## [1] 5 2
```

```
colnames(y) <- c("vec1", "vec2")
comment(y) <- c("the first column is pretend data", "the second column is yet more pretend data ")
str(y)
```

```
##  int [1:5, 1:2] 1 2 3 4 5 11 12 13 14 15
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:2] "vec1" "vec2"
##  - attr(*, "comment")= chr [1:2] "the first column is pretend data" "the second column is yet more
```

```
attributes(y)
```

```
## $dim
## [1] 5 2
##
## $dimnames
## $dimnames[[1]]
## NULL
##
## $dimnames[[2]]
## [1] "vec1" "vec2"
##
##
## $comment
## [1] "the first column is pretend data"
## [2] "the second column is yet more pretend data "
```

‘‘‘r

Calling a function like summary() will do very different things for different object classes. We will use this call alot for data frames and output from statistical models, etc..

```
summary(x)  # numeric vector
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0     4.5     5.0     5.0     5.5     6.0
```

```
summary(string_1)  # character string
```

```
##    Length     Class      Mode
##         1 character character
```

The call to `summary()` is generic, which first looks at the class of the object, and then uses a class specific method to generate a summary of x.

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0     4.5     5.0     5.0     5.5     6.0
```

```
summary.default(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0     4.5     5.0     5.0     5.5     6.0
```

but..

```
summary.lm(x)  # Since this was looking for an object of class lm
```

```
## Error: $ operator is invalid for atomic vectors
```

To see all of the methods used when you call the generic `summary()` for S3 classes.

```
methods(summary)
```

```
##  [1] summary.aov             summary.aovlist
##  [3] summary.aspell*         summary.connection
##  [5] summary.data.frame      summary.Date
##  [7] summary.default         summary.ecdf*
##  [9] summary.factor          summary.glm
## [11] summary.infl            summary.lm
## [13] summary.loess*          summary.manova
## [15] summary.matrix          summary.mlm
## [17] summary.nls*            summary.packageStatus*
## [19] summary.PDF_Dictionary* summary.PDF_Stream*
## [21] summary.POSIXct         summary.POSIXlt
## [23] summary.ppr*            summary.prcomp*
## [25] summary.princomp*       summary.proc_time
## [27] summary.srcfile         summary.srcref
## [29] summary.stepfun         summary.stl*
## [31] summary.table           summary.tukeysmooth*
##
##    Non-visible functions are asterisked
```

4.8.18 Syntax style guide

Generally it is advisable to use a consistent way of scripting. For any given programming language there are syntax style guide. The Style guide for my class. You can also check out the R style guide from Google.

4.8.19 Random bits

Note about using q() on the Mac R GUI in v2.11.+ The programming team decided the default behaviour was potentially 'dangerous', and people may lose their files, so they have changed it to command + q to quit instead. If you are an old-fogey like me and like to use q(), you have a couple of options. base::q() # This will work, but it is annoying.

you can set your `.Rprofile` to have the following line.

```
options(RGUI.base.quit=T)
```

and the next time you run R the old `q()` will work.

If you do not know how to create or edit .Rprofile, come speak with me...

4.8.20 session info

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()
```

```
## R version 3.0.1 (2013-05-16)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.5
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.1 formatR_0.10   stringr_0.6.2  tools_3.0.1
```

```
Sys.time()
```

```
## [1] "2014-08-07 10:39:49 EDT"
```

R indexing begins at 1 (not 0 like Python) Negative values of indexes in R

mean something very different. for instance

a[-1] # this removes the first element of a, and prints out all of the remaining elements.

```
[1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

‘‘‘

```
# As far as I know all classes of objects are mutable, which means you can
# write over the name of the objects, values within the objects, and
# slots....

# Indexing on a character string does not work in R
string_1 <- "hello world"
string_1[1]
```

```
## [1] "hello world"
```

```
# instead you need to use the substr() function
substr(x = string_1, start = 1, stop = 1)
```

```
## [1] "h"
```

```
# similarly
length(string_1)  # this gives an output of 1
```

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

```
nchar(string_1)  # this gives the 11 characters
```

```
## [1] 11
```

TOC

Section 1: What is R; R at the console; quiting R

Section 2: R basics; R as a calculator; assigning variables; vectorized computation in R

Section 3: pre-built functions in R

Section 4: Objects, classes, modes - Note: should I add attributes?

Section 5: The R workspace; listing objects, removing objects (should I add attach and detach?)

Section 6: Getting Help in R

Section 7: Using A script editor for R

Section 8: Writing simple functions in R

Section 8b: Using source() to call a set of functions

Section 9: Regular sequences in R

Section 10: Extracting (and replacing), indexing & subsetting (using the index). Can also be used for sorting.

Advanced stuff to learn on your own...

..... setting attributes of objects.... (names, class, dim)

..... environments (see ?environments)

4.9 Control Flow and loops in R

4.9.1 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"
```

4.9.2 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"
```

4.9.3 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.

4.9.4 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
```

4.9.5 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"
```

4.9.6 So for the for loop we would do the following:

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

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

4.9.7 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)
```

4.9.8 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.

4.10 Variant calling and exploration of polymorphisms

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

4.10.1 ## 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/
```

4.10.2 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
```

4.10.3 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")
```

4.10.4 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)

4.11 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 Starting up a custom operating system instead; use AMI ami-7607d01e.

4.11.1 Switching to root

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

```
sudo bash
```

4.11.2 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.

4.11.3 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.

4.11.4 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

4.11.5 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
```

4.12 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-T

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/
```

4.13 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
```

4.14 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_r

#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("ch
}
```

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
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
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
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.

4.15 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 Starting up a custom operating system).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_melanogaste
gunzip Drosophila_melanogaster.BDGP5.75.dna.toplevel.fa.gz
curl -O ftp://ftp.ensembl.org/pub/release-75/gtf/drosophila_melanogaster/Drosophila_melanogaster.BDGP
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.to
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
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
    #Count the number of reads mapping to each feature using HTSeq
    htseq-count --format=bam --stranded=no --order=pos ${sample}/accepted_hits.bam ${annotation} > $
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.

4.16 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.

4.17 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 Starting up a custom operating system.

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.

4.18 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
```

4.19 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
```

4.20 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.

4.21 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}
```

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_al
```

4.21.1 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  > ${sa
    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
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.

4.22 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.

4.22.1 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

4.22.2 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.

4.22.3 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 Starting up a custom operating system).

Log in with Windows or from Mac OS X.

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

4.22.4 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.

4.22.5 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
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
```

4.22.6 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
```

4.22.7 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.

4.22.8 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_0104
```

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
java -Xmx500m -Djava.awt.headless=true -cp ~/mauve_linux_snapshot_2012-06-07/Mauve.jar  org.gel.mauve
```

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.

4.22.9 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.fn
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_0104
curl -O ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_O104_H4_2011C_3493_uid176127/NC_018
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\..*/

# 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/
```

4.22.10 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.

4.23 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
```

4.23.1 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
```

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.

4.23.2 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?

4.24 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.

4.24.1 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.

4.24.2 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?

4.24.3 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=Sw
```

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
```

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.

4.24.4 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
```

4.24.5 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.

4.25 So you want to get some sequencing data in NCBI?

This is a set of tutorials for working with the NCBI and MG-RAST databases – specifically, to download project specific information.

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-ngs2014.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.

4.25.1 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. Go to the web portal and look up each FASTA
2. Go to 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 focus on using APIs.

4.25.2 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.

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
```

Note that the NCBI knows a lot about this genome. Check it out here.

If I want to access other kinds of data associated with this genome, I would try the following command:

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

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.

4.25.3 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* on your EC2 instance:

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

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

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

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.

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.

To run the script:

```
python fetch-genomes.py interesting-genomes.txt genbank-files
```

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.

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=
```

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.

4.25.4 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*.

4.25.5 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. I've done that before.

You could also use existing packages to parse Genbank files. I have the most experience with BioPython. To begin with, let's just use BioPython so you can get to using existing scripts without writing scripts.

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

```
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.

To run this script on the Genbank file for CP000962:

```
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.

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

Here is some documentation on GEO in NCBI, here.

4.26 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.ipyn
```

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.

—

4.27 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_
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

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)

4.28 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.

4.28.1 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 Starting up a custom operating system).

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.

4.28.2 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
```

4.28.3 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/
```

4.29 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.

4.29.1 Logging in

Log in and type:

```
sudo bash
```

to change into superuser mode.

4.29.2 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.

4.29.3 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.

4.29.4 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 s
```

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?

4.30 Amazon Web Services instructions

4.30.1 Start up an EC2 instance

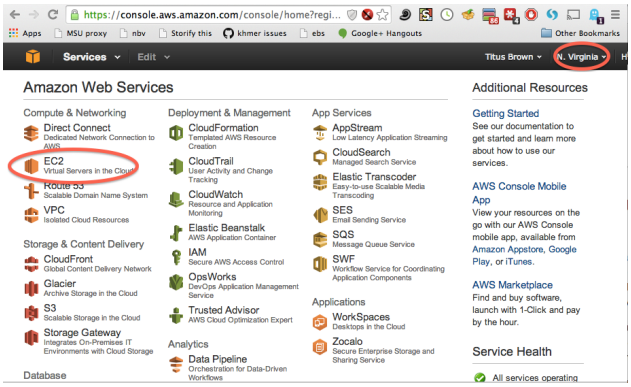
Here, we're going to startup an Amazon Web Services (AWS) Elastic Cloud Computing (EC2) "instance", or computer.

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

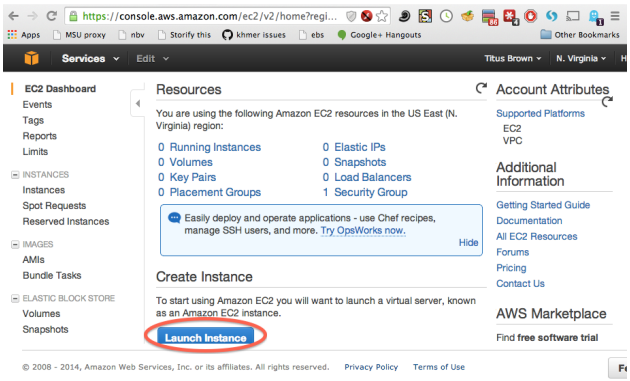
Select 'My Account/Console' menu option 'AWS Management Console."

Log in with your username & password.

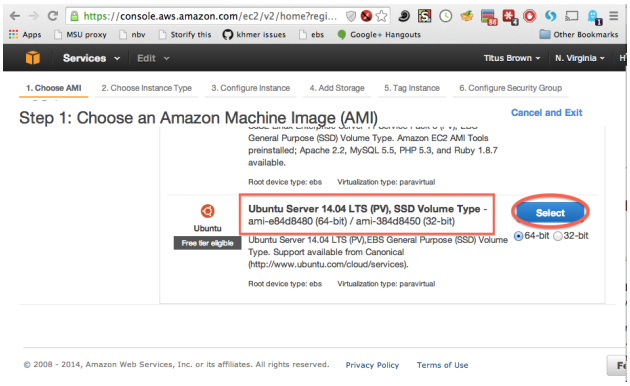
Make sure it says North Virginia in the upper right, then select EC2 (upper left).



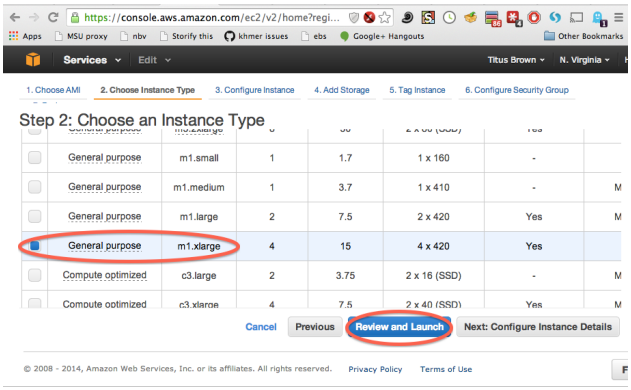
Select “Launch Instance” (midway down the page).



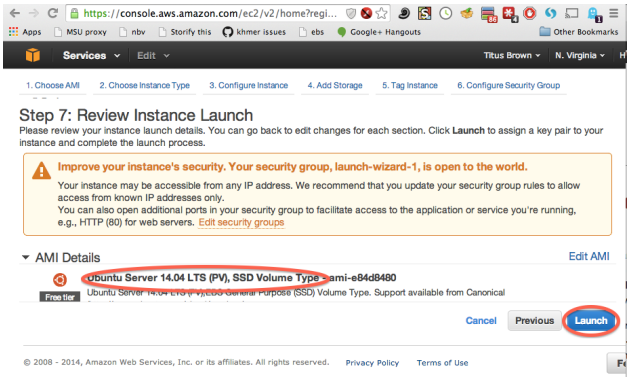
Next, scroll down the list of operating system types until you find Ubuntu 14.04 LTS (PV) – it should be at the very bottom. Click ‘select’. (See Starting up a custom operating system if you want to start up a custom operating system instead of Ubuntu 14.04.)



Scroll down the list of instance types until you find “m1.xlarge”. Select the box to the left, and then click “Review and Launch.”

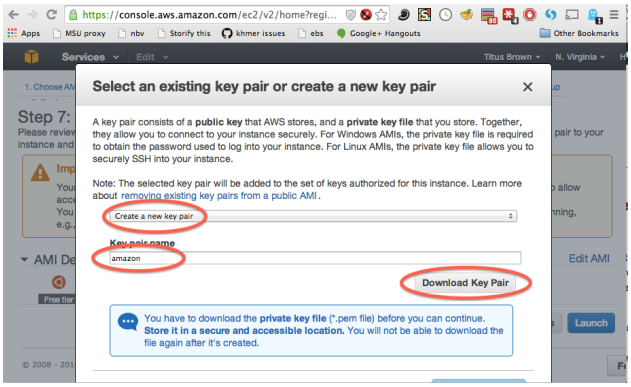


Ignore the warning, check that it says "Ubuntu 14.04 LTS (PV)", and cick "Launch".

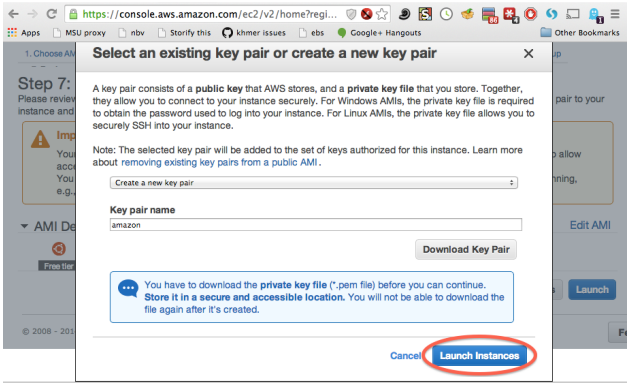


The *first* time through, you will have to "create a new key pair", which you must then name (something like 'amazon') and download.

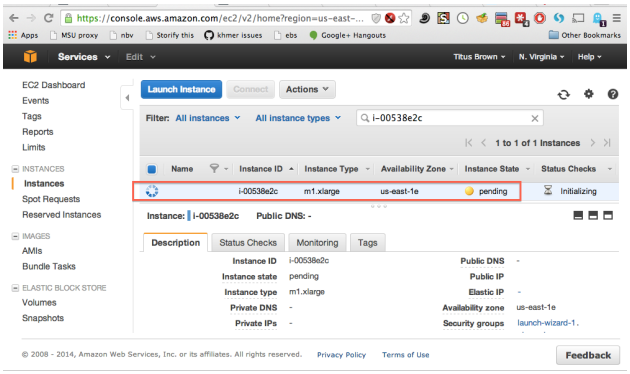
After this first time, you will be able to select an existing key pair.



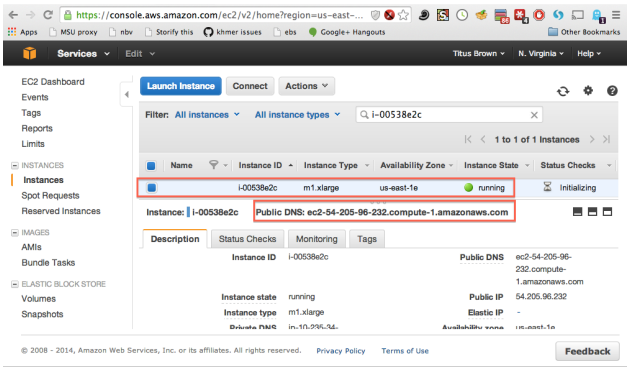
Select "Launch Instance."



Select "view instance" and you should see a "pending" line in the menu.



Wait until it turns green, then make a note of the "Public DNS" (we suggest copying and pasting it into a text notepad somewhere). This is your machine name, which you will need for logging in.



Then, go to Logging into your new instance "in the cloud" (Windows version) or Logging into your new instance "in the cloud" (Mac version)

You might also want to read about Terminating (shutting down) your EC2 instance.

4.30.2 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 'root', 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
```

(but you have to replace the stuff after the '@' sign with the name of the host).

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.

You should now see a text line that starts with something like `ubuntu@ip-10-235-34-223:~$`. You're in! Now type:

```
sudo bash
cd /root
```

to switch into superuser mode (see: http://xkcd.com/149/) and go to your home directory.

This is where the rest of the tutorials will start!

If you have Dropbox, you should now visit Installing Dropbox on your EC2 machine.

You might also want to read about Terminating (shutting down) your EC2 instance.

To log out, type:

```
exit
logout
```

or just close the window.

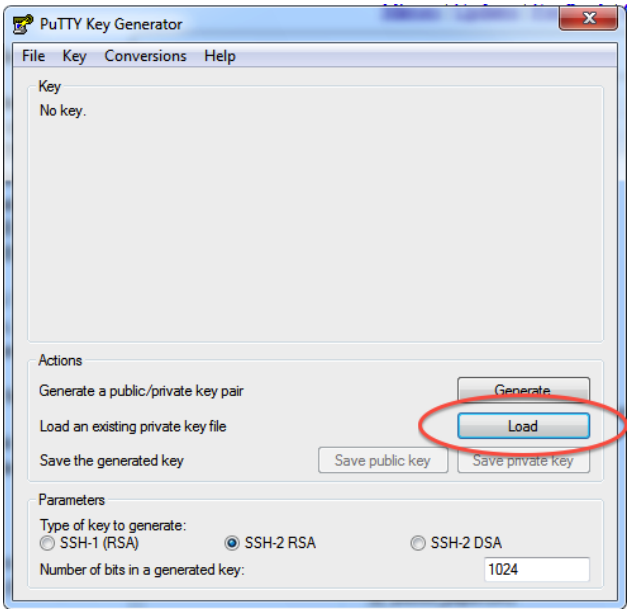
4.30.3 Logging into your new instance "in the cloud" (Windows version)

Download Putty and Puttygen from here: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

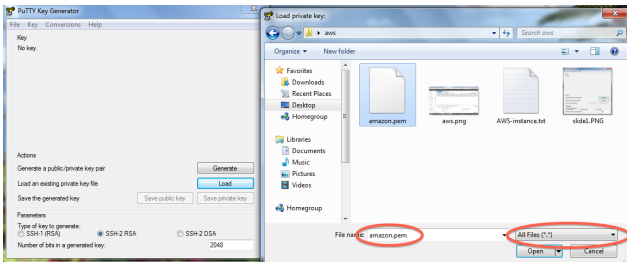
Generate a ppk file from your pem file

(You only need to do this once!)

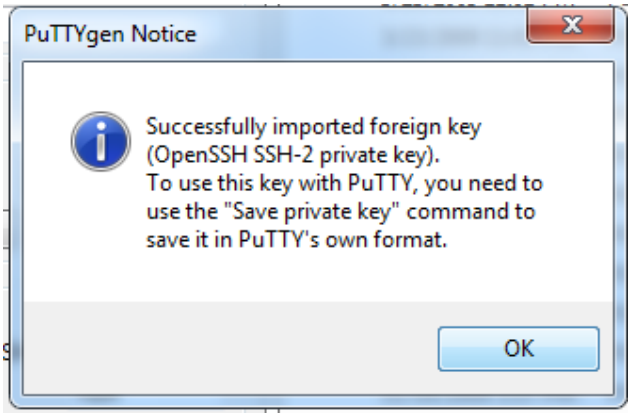
Open puttygen; select "Load".



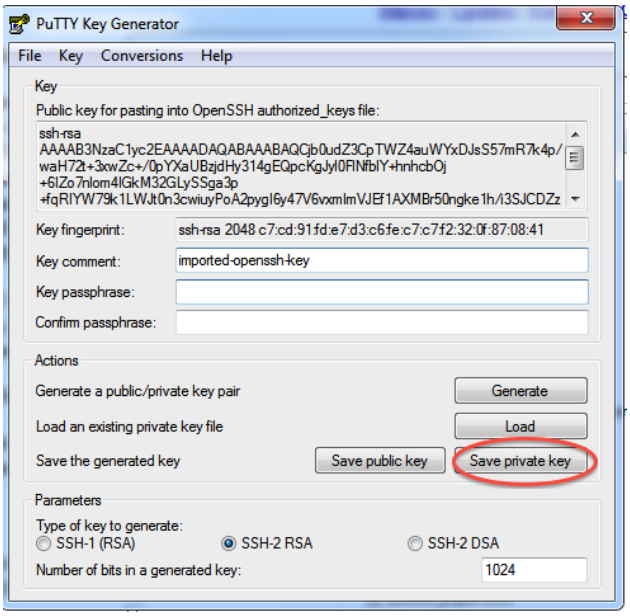
Find and load your '.pem' file; it's probably in your Downloads folder. Note, you have to select 'All files' on the bottom.



Load it.

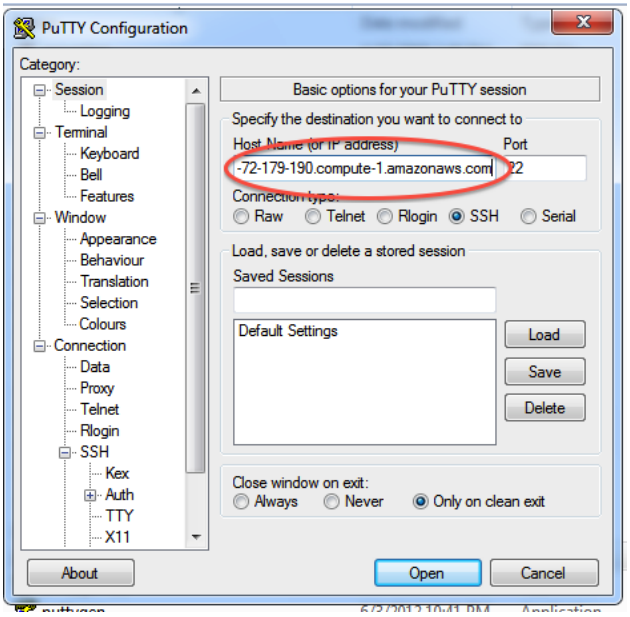


Now, "save private key". Put it somewhere easy to find.

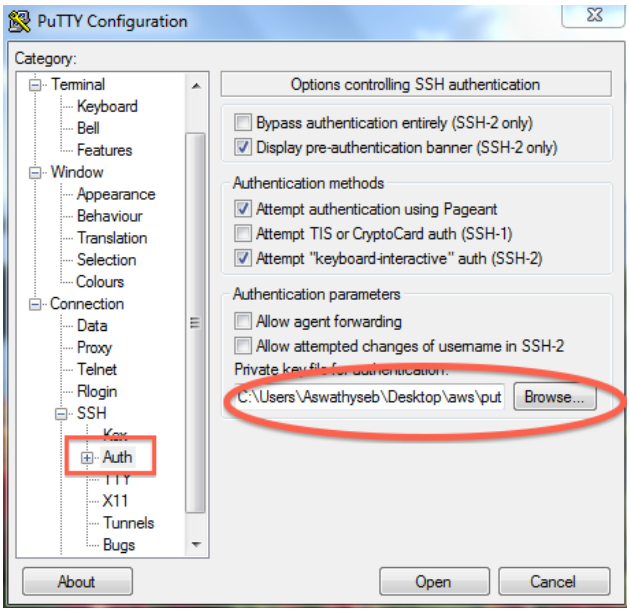


Logging into your EC2 instance with Putty

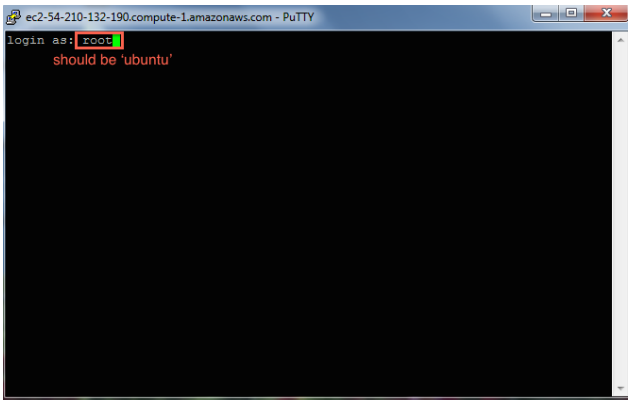
Open up putty, and enter your hostname into the Host Name box.



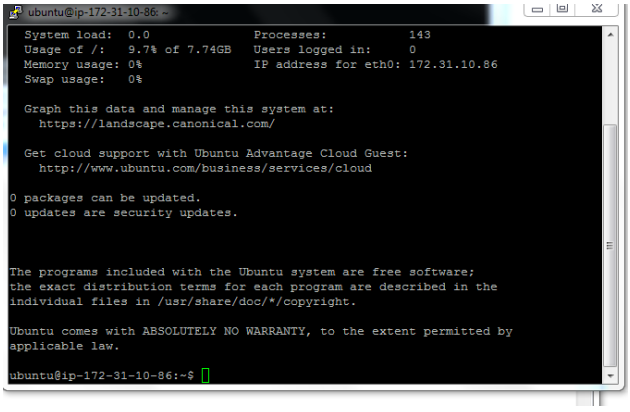
Now, go find the 'SSH' section and enter your ppk file (generated above by puttygen). Then select 'Open'.



Log in as “ubuntu”.



Declare victory!



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.

You should now see a text line that starts with something like `ubuntu@ip-10-235-34-223:~$`. You’re in! Now type:

```
sudo bash
cd /root
```

to switch into superuser mode (see: http://xkcd.com/149/) and go to your home directory.

This is where the rest of the tutorials will start!

If you have Dropbox, you should now visit Installing Dropbox on your EC2 machine.

You might also want to read about Terminating (shutting down) your EC2 instance.

To log out, type:

```
exit
logout
```

or just close the window.

4.30.4 Installing Dropbox on your EC2 machine

IMPORTANT: Dropbox will sync everything you have to your EC2 machine, so if you are already using Dropbox for a lot of stuff, you might want to create a separate Dropbox account just for the course.

Start at the login prompt on your EC2 machine:

```
sudo bash
cd /root
```

Then, grab the latest dropbox installation package for Linux:

```
wget -O dropbox.tar.gz "http://www.dropbox.com/download/?plat=lnx.x86_64"
```

Unpack it:

```
tar -xvzf dropbox.tar.gz
```

Make the Dropbox directory on /mnt and link it in:

```
mkdir /mnt/Dropbox
ln -fs /mnt/Dropbox /root
```

and then run it, configuring it to put stuff in /mnt:

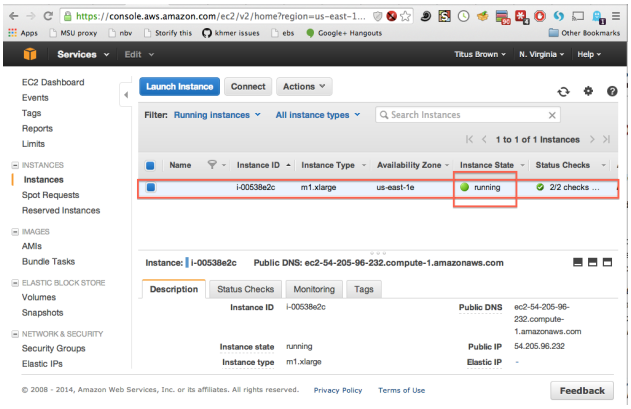
```
HOME=/mnt /root/.dropbox-dist/dropboxd &
```

When you get a message saying "this client is not linked to any account", copy/paste the URL into browser and go log in. Voila!

Your directory '/root/Dropbox', or, equivalently, '/mnt/Dropbox', is now linked to your Dropbox account!

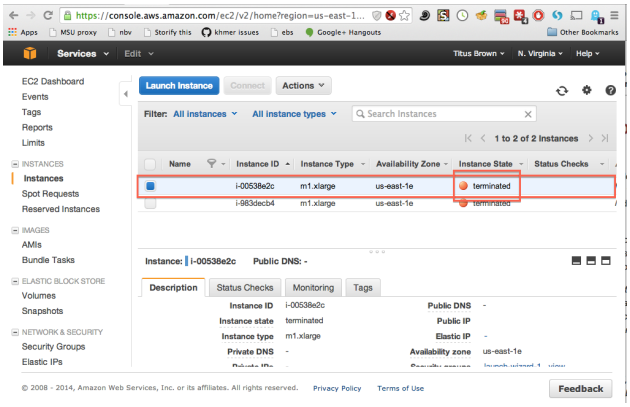
4.30.5 Terminating (shutting down) your EC2 instance

While your instance is running, Amazon will happily charge you on a per-hour basis – check out the pricing for more information. In general, you will want to shut down your instance when you're done with it; to do that, go to your EC2 console and find your running instances (in green).



Then, select one or all of them, and go to the 'Actions...' menu, and then select 'Terminate'. Agree.

After a minute or two, the console should show the instance as "terminated".



4.30.6 Storing data persistently with Amazon EBS Volumes

If you want to save your data across instances – that is, if you want to have persistent data – Amazon can do that for you, too. Amazon is happy to rent disk space to you, in addition to compute time. They'll rent you disk space in a few different ways, but the way that's most useful for us is through what's called Elastic Block Store(EBS). This is essentially a hard-disk rental service.

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

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

(Learn more from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html)

In this and the following tutorials, you will be instructed to create new Amazon EBS Volume, to create Amazon EBS Snapshot from EBS Volume and to restore EBS Volume from EBS Snapshot.

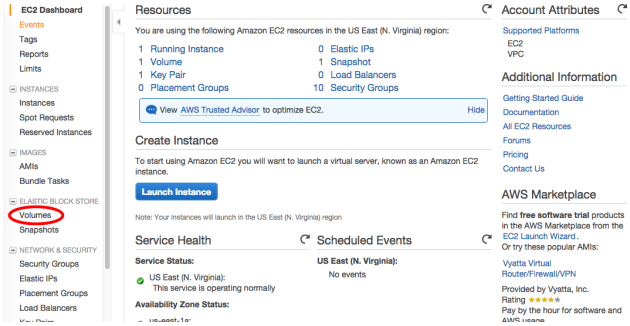
Prerequisites

This tutorial assumes you've already set up an account on Amazon Web Services, and that you've completed the EC2 tutorial to set up an Amazon instances.

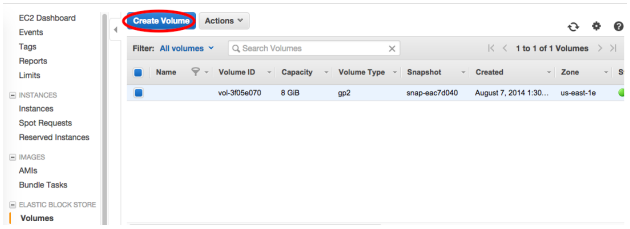
Creating an Amazon EBS Volume

Firstly open the Amazon EC2 console at 'https://console.aws.amazon.com/ec2>' and make sure it says North Virginia in the upper right.

At the AWS Management Console, on the left menu bar, click "Volumes".

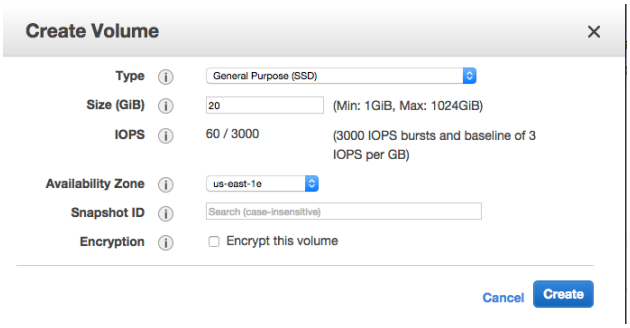


Click "Create Volume".

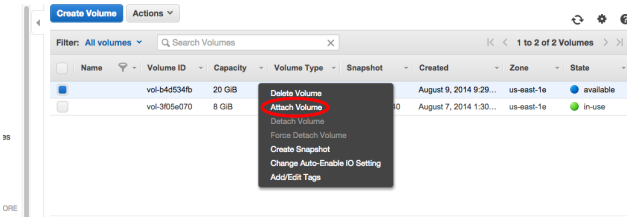


Enter the desired size, and select the zone in which your instance is running. The volume and instance must be in the same zone. Otherwise, the volume cannot be attached to your instance.

Then click "Create".

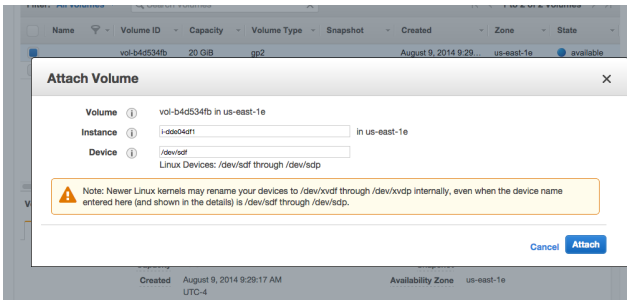


Wait for your volume to finish being created, then click "Attach Volume".



Select the desired running instance. You may leave the Device at the default: /dev/sdf. This is the name that your EC2 instance can use to connect to the disk space. (If this is the second volume you want to attach to this instance, this may be different, like /dev/sdg.)

Click "Attach".



When attachment is complete, connect to your instances via SSH.

If the volume is newly created, you must format the volume.

WARNING: ONLY DO THIS ONCE, WHEN YOU FIRST CREATE THE VOLUME. OTHERWISE, YOU WILL LOSE ALL YOUR DATA. YOU SHOULD NOT DO THIS IF THE VOLUME IS RESTORED FROM A SNAPSHOT AND YOU WANT TO USE THE DATA ON IT.

```
ubuntu@ip-10-31-232-122:~$ sudo mkfs -t ext4 /dev/xvdf
```

Note1: here we use "xvdf" instead of "sdf".

Note2: The device may be different. If this is the second volume you attached to the same instance, the device ID may be /dev/xdg, so here just use "xvdg" instead of "sdg". In this situation, for all the commands below, replace "/dev/xvdf" by "/dev/xvdg".

Then, mount the volume. You'll do this every time you attach the volume to an instance:

```
ubuntu@ip-10-31-232-122:~$ sudo mkdir /data
ubuntu@ip-10-31-232-122:~$ sudo mount /dev/xvdf /data
```

Your drive is now ready to use – it will be available under /data/.

Detaching an Amazon EBS Volume

Any volumes you have attached will automatically detach when you shut down the instance. You can also stop all processes that are using the volume, change out of the directory, and type

```
ubuntu@ip-10-31-232-122:~$ sudo umount -d /dev/xvdf
```

and then detach the volume via the AWS Web site.

4.30.7 Using Amazon EBS Snapshots for sharing and backing up data

Now you have the Amazon EBS Volume to store the data for your instance to use. But it can only be attached to the EC2 instance you created. If you want to share the EBS Volume with the data with other colleagues so they can use the data on the EC2 instance they created, you need to create an Amazon EBS Snapshot and share it with other Amazon EC2 user. You can also create Amazon EBS Snapshots periodically to backup your Amazon EBS Volume.

In this tutorial, you will be instructed to create an Amazon EBS Snapshot from an Volume, to share the EBS Snapshot with others and to restore an Volume from an Snapshot.

Prerequisites

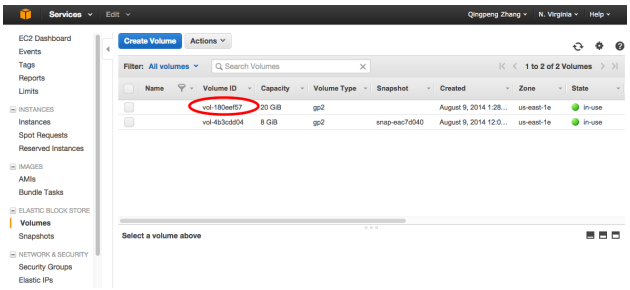
This tutorial assumes you've completed the EC2 tutorial to set up an Amazon EBS Volume.

Creating an Amazon EBS Snapshot from a Volume

Firstly open the Amazon EC2 console at 'https://console.aws.amazon.com/ec2' and make sure it says North Virginia in the upper right.

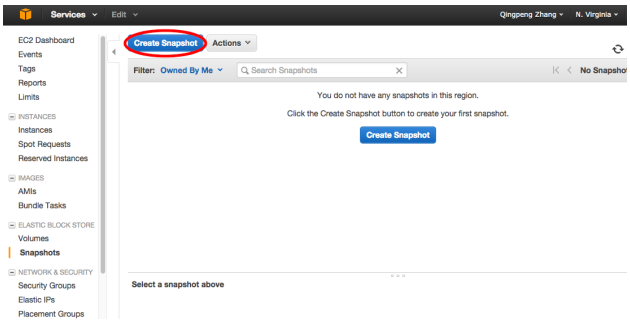
At the AWS Management Console, on the left menu bar, click "Volumes".

Here you can see the 20GiB volume you created in the tutorial "Storing data persistently with Amazon EBS Volumes".



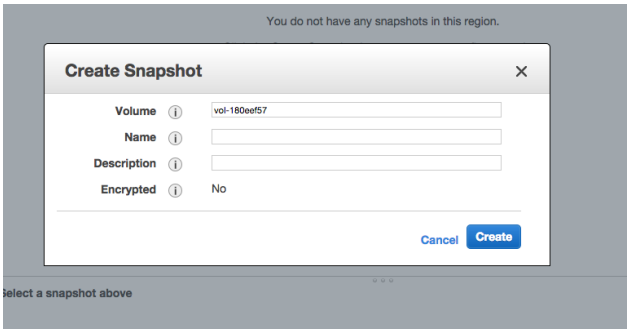
This tutorial will guide you to create a Amazon EBS Snapshot for this volume. In this example, the "Volume ID" of that volume is "vol-180eef57". Record this ID, we will use it later.

Next, on the left menu bar, click "Snapshots" and click "Create Snapshot".



Choose the Volume we want to create Snapshot for. (The ID is vol-180eef57 for this example, as we recored in last step.) You can enter the information for "Name" and "Description" as you like or just leave them blank.

Then click "Create".



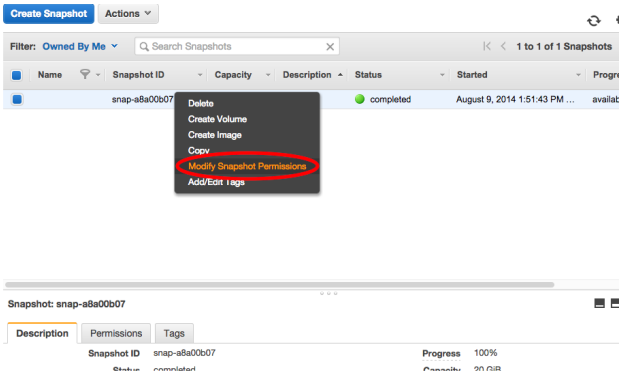
Ok. Now you have created a Snapshot from that 20G Volume.

Sharing an Amazon EBS Snapshot

For now, the snapshot we just created is private, which can only be viewed and used by yourself.

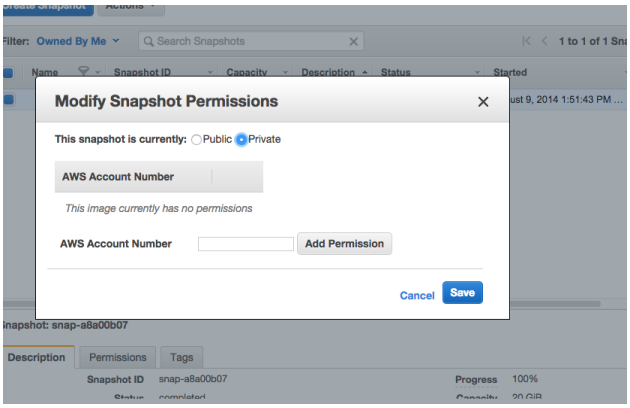
To make it public or share it with other Amazon EC2 user:

Right click the snapshot and click “Modify Snapshot Permissions”.



If you want to make this snapshot public, which means any Amazon EC2 user can have access to this snapshot and all the data in it, click “Public”.

If you just want to share this snapshot with specific person, like your colleague, keep the option as “Private” but put the AWS Account Number (which can be acquired by checking “Account Settings”) of the person in the box and click “Add Permission”.

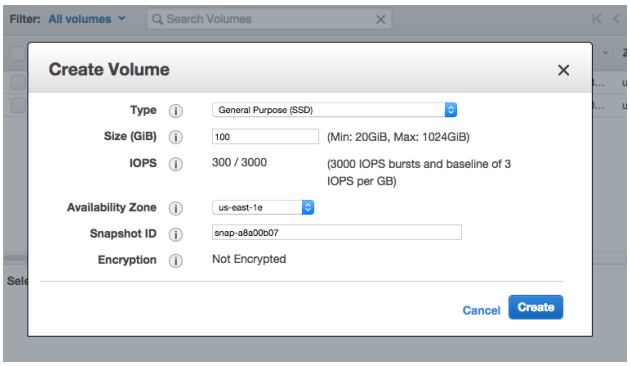


Now you can share the “Snapshot ID” (snap-a8a00b07 in this example) to your colleague so they can restore an EBS Volume from this snapshot and use the data in it.

Restoring an Amazon EBS Volume from a Snapshot

Your colleague shares an Amazon EBS Snapshot with you and you want to use the data on it. You did something terribly wrong to the data on your Volume and you want to restore the data from the backup Snapshot of that Volume. Under these circumstances, you want to restore an EBS Volume from a Snapshot.

It is similar to how you create a new EBS Volume. The only difference is that in the dialog box after you click “Create Volume” button, input the “Snapshot ID” of the snapshot you want to restore. Similarly, also select the zone in which your instance is running.



Ok, now you have the volume available to attach to your running instance.

4.30.8 Transferring Files between your laptop and Amazon instance

For linux/Unix/Mac system, we can use a command-line tool "scp" to transfer files between your laptop and Amazon instance. Also we can use a GUI tool "FileZilla" to do the transfer, which is more user-friendly.

Using scp to transfer data

"scp" means "secure copy", which can copy files between computers on a network. You can use this tool in a Terminal on a Unix/Linux/Mac system.

To upload a file from your laptop to Amazon instance:

```
$scp -i ~/Desktop/amazon.pem ~/Desktop/MS115.fa  ubuntu@ec2-54-166-128-20.compute-1.amazonaws.com:~/d
```

This command will upload a file - MS115.fa in your ~/Desktop/ folder of your laptop to folder ~/data/ on an Amazon instance. Note you still need to use the private key you used to connect to the Amazon instance with ssh. (In this example, it is the amazon.pem file in ~/Desktop/.

Note: You need to make sure that the user "ubuntu" has the permission to write in the target directory. In this example, if ~/data/ was created by user "ubuntu", it should be fine.

Similarly, to download a file from Amazon instance to your laptop:

```
$scp -i ~/Desktop/amazon.pem ubuntu@ec2-54-166-128-20.compute-1.amazonaws.com:/data/ecoli_ref-5m-trim
```

This command will download a file /data/ecoli_ref-5m-trim.fastq.gz from Amazon instance to your ~/Download folder in your laptop.

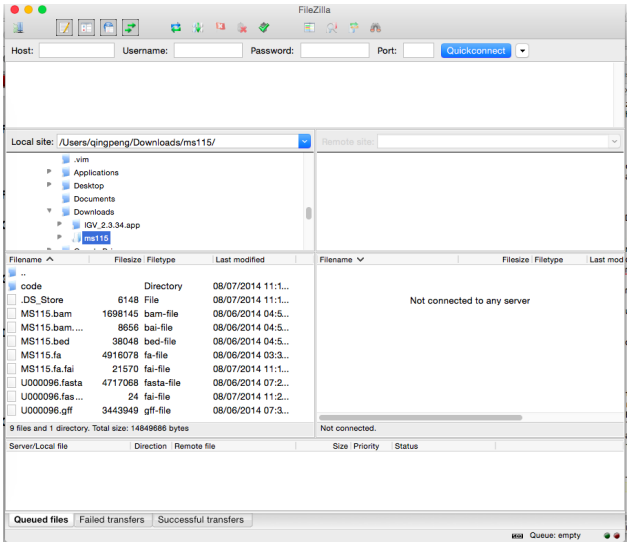
Note: You can use asterisk(*) to download multiple files, like *.fasta.gz.

Using FileZilla to transfer data

If you want a more user-friendly tool to transfer data, FileZilla is a good choice. It is free, it supports Windows/Linux/Mac systems, and it has a good user interface. It supports FTP, SFTP and other file transfer protocols.

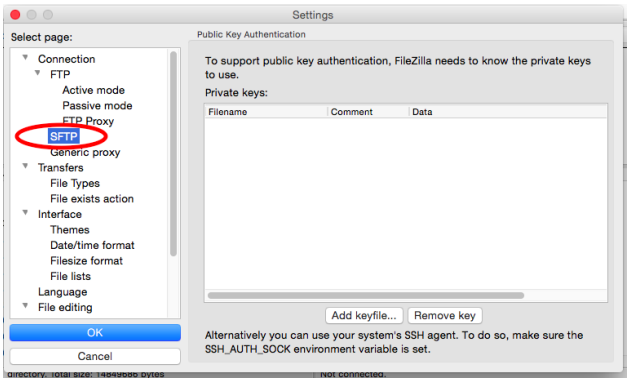
Firstly, go to 'https://filezilla-project.org/' and click "Download FileZilla Client" button to download it.

The interface of FileZilla is like this:

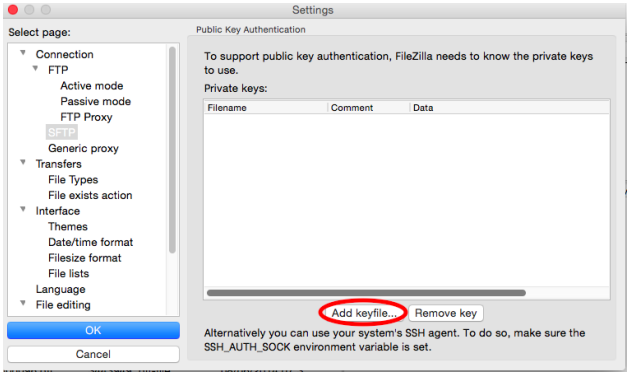


If you want to use FileZila to upload to or download data from a normal FTP server if you have the user and password, just put the information in the "Host", "Username", "Password" box and connect. However for Amazon instance, we use key-pair to log in instead of password for better safety. So it is a little bit more complicated to configure.

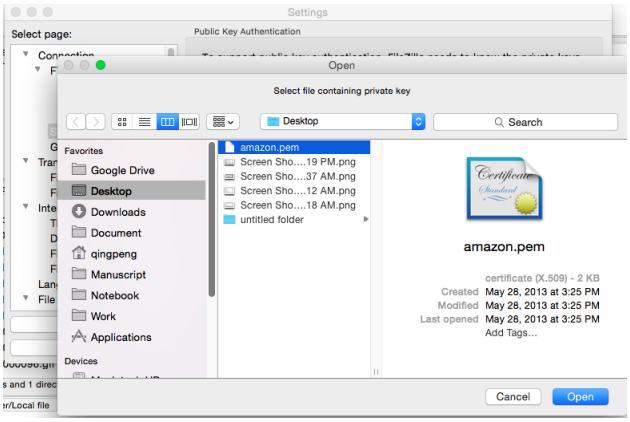
Open "Settings" and click "SFTP":



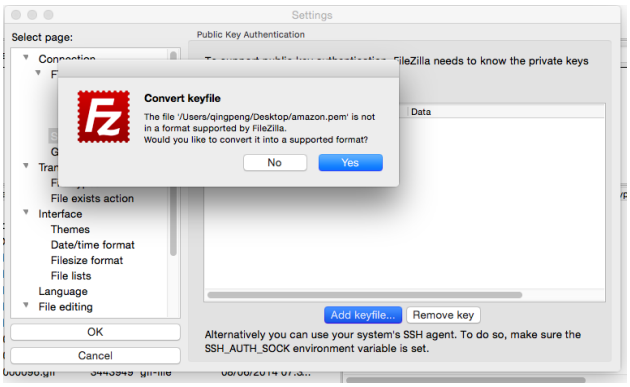
Click "Add keyfile...":



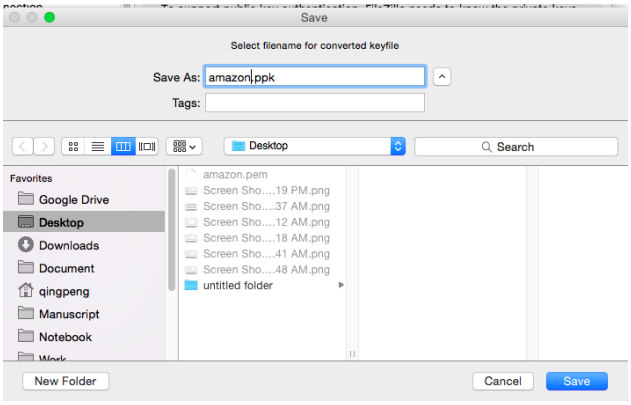
Then select the ".pem" file you used to connect to Amazon instance with ssh.



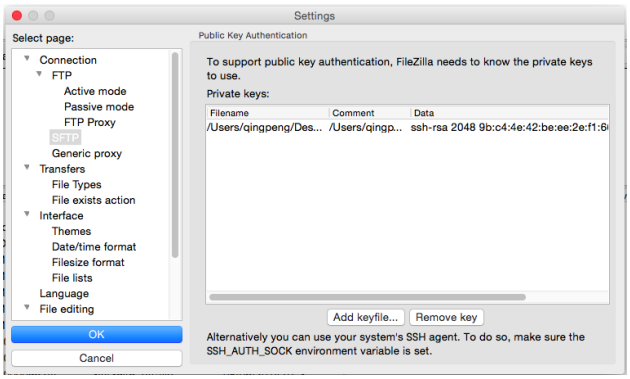
There is a dialog box to ask you if you want to convert the ".pem" file into a supported format. Click "Yes".



Name it with extension as ".ppk" and save it.

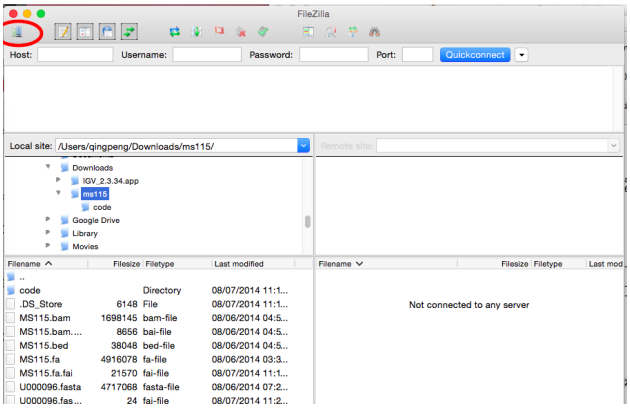


You will see the a private key has been added.

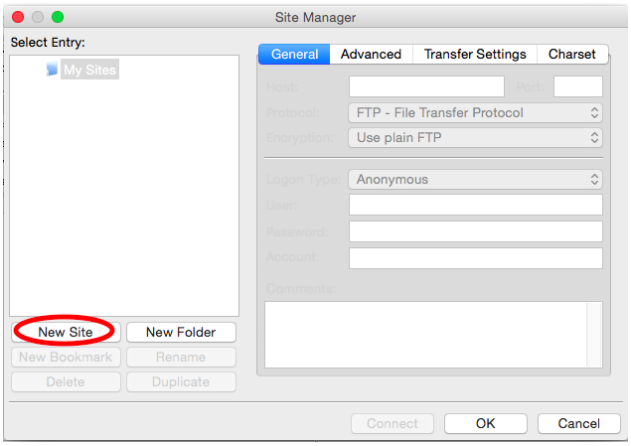


Close “Settings” and go back to the main interface.

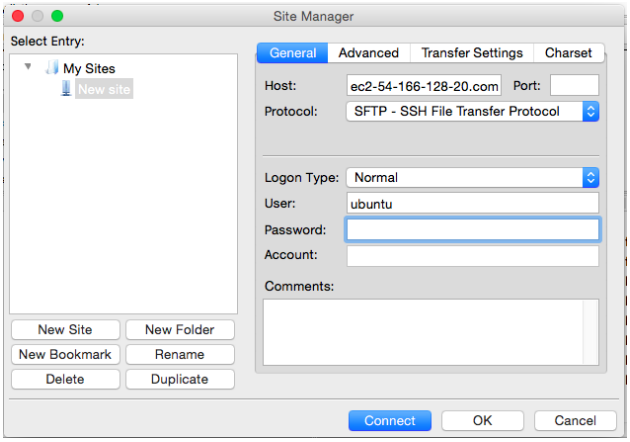
Click button to open the site manager.



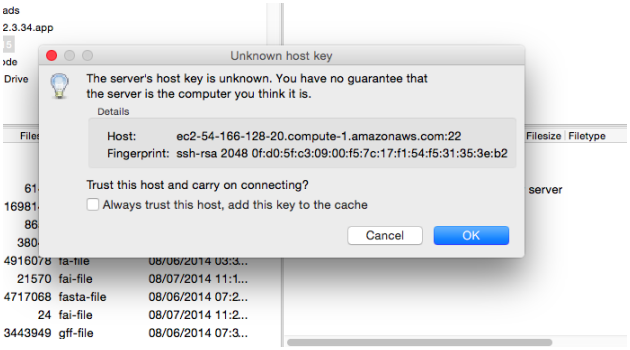
Click “New Site”.



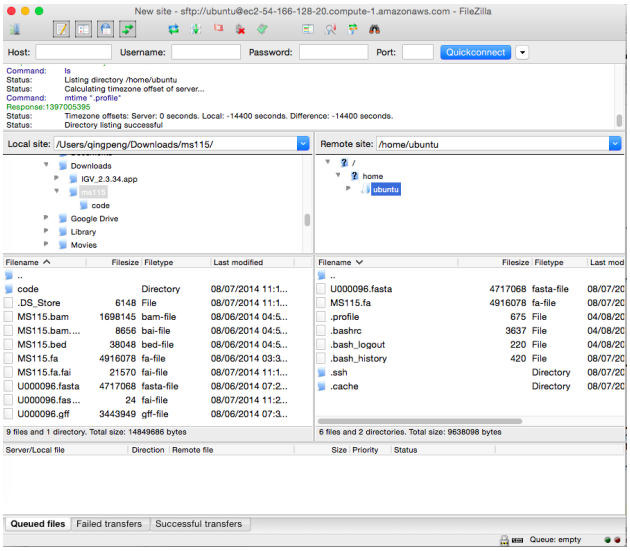
Put the Amazon instance URL like ec2-54-166-128-20.compute-1.amazonaws.com in the “Host” box. Set “Protocol” as “SFTP”, “Logon Type” as “Normal”, “User” as “ubuntu” and leave “Password” as blank. Then click “Connect”.



There will be a dialogue box to ask you about "Unknown host key", just click "Ok".



All right. Now you have logged in the Amazon instance. You can drag and drop to transfer the files between the remote machine and your local laptop.



4.30.9 Uploading files to Amazon S3 to share

"Amazon Simple Storage Service(S3) is storage for the Internet." It is designed to make web-scale computing easier for developers with a highly scalable, reliable, fast, inexpensive data storage infrastructure. Companies like Dropbox, Pinterest, Tumblr store their data on the S3 servers.

(Learn more from http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html)

For personal users like us, S3 is also a reliable, inexpensive service to store data or share data around the world.

This tutorial will instruct you to upload files to Amazon S3 and share them.

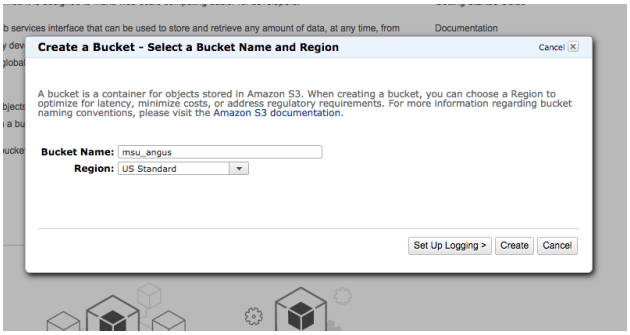
Uploading files to Amazon S3

Go to Amazon S3 Console: https://console.aws.amazon.com/s3/

Click "Create Bucket".

In the dialog box, in the Bucket Name box, enter a bucket name. Here a "bucket" is like a "folder" in concept. The bucket name must be unique across all existing bucket names in Amazon S3. You can not change the name after you create a bucket. Also, the bucket name you chose here will be visible in the URL that points to the objects(files) stored in the bucket. So make sure the bucket name you choose is appropriate.

Leave the "Region" as default and click "Create".

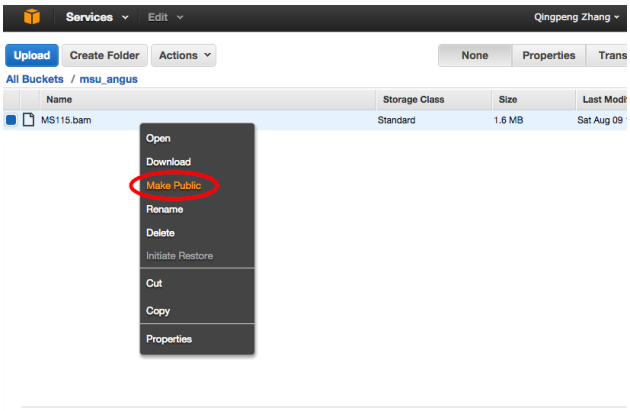


Next you can click the bucket we just created, and it is an empty one.

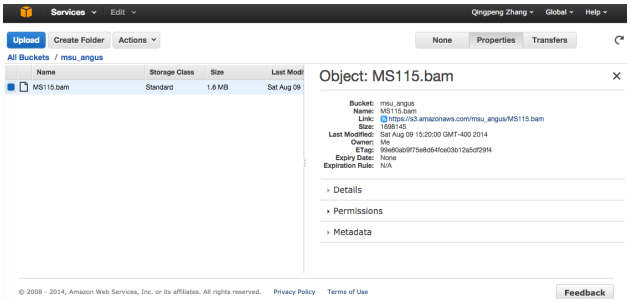
Now we can add files into this bucket(folder) by clicking "Upload".

After you select the files and upload them successfully, you can see them in the current bucket. Right click the file, you can manipulate it in many ways you like, like "Rename", "Delete", "Download", and others.

Out of them, "Make Public" is what you want to click to share the file.



When it is done, highlight the file you just shared and click "Properties" on the upper right. You will see the link of this file, like https://s3.amazonaws.com/msu_angus/MS115.bam in this example.



So that's the public URL of that file. You can share this link to the person you want to share this file with.

Downloading files from Amazon S3

You can use any internet browser to download the file from Amazon S3 directly with the public URL.

In command-line environment, you can use curl to download the file:

```
$curl -O https://s3.amazonaws.com/msu_angus/MS115.bam
```

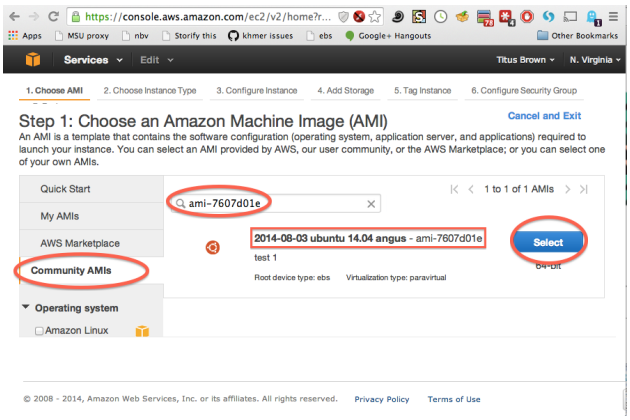
This command wil download the file into current directory.

4.30.10 Starting up a custom operating system

The instructions in Start up an EC2 instance tell you how to start up a machine with Ubuntu Linux version 14.04 on it, but that machine comes with very little software installed. For anything where you are executing actual analyses, you're going to want to have a bunch of basic software installed.

Therefore, we make custom versions of Ubuntu available as well, that come with some software pre-installed. (See Technical guide to the ANGUS course for a list of the packages installed on the ANGUS custom AMI.)

To boot these, go to EC2/Launch and select "Community AMIs" instead of the default Quick Start; then type in the AMI number or name given to you in the tutorial. Below is a screenshot of an example for 'ami-7606d01e'. Then proceed with the rest of Start up an EC2 instance.



4.30.11 Technical guide to the ANGUS course

Packages we install

Install:

```
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 \
        ipython-notebook
```

Creating your own AMI

Boot up the machine you want (Ubuntu 14.04 LTS PV is what we used in 2014), then run the apt-get commands above. Shut it down, and then follow these instructions:

> http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html

I usually name them something like '2014-08-03 ubuntu 14.04 angus', with 'v1', 'v2', 'v3' on the end of 03 (e.g. 2014-08-03.v1) just so I can sort them more easily.

At the end of this, you should be able to boot it up and run the apt-get commands, above, and have them complete instantly (beacuse it's up to date)

Be sure to make it public! (Actions... Modify Image permissions under AMI menu).

4.31 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.

4.31.1 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.

4.32 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.

4.32.1 Need Help?

You can reach the course director, Titus Brown, at titus@idyll.org 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.

4.32.2 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.

4.32.3 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 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.