Project #1. “What causes antibiotic resistance?” Alignment to reference, variant calling.

Microbial resistance to antibiotics is a major problem in the treatment of bacterial infections. Evolution is a master algorithm at finding ways to survive, and there is often more than one way for bacteria to become resistant to a particular antibiotic. For scientists and doctors, knowing the exact mechanism of resistance that evolved in a bacterial strain is very important. This knowledge can guide the development of new, better drugs, or even help doctors to decide which alternative antibiotics to use if one is failing to treat a patient.

In this project, we will work with real sequencing data from a strain of *E. coli* resistant to the antibiotic ampicillin. Your job is to analyze that sequencing data to locate the mutations responsible for giving *E. coli* its antibiotic resistance property. You will research the genes that are mutated to identify the mechanism of antibiotic resistance in each case, and you will make recommendations for alternative antibiotics a doctor could use to treat each strain.

REMINDER: Any time you enter a command that processes, measures, or analyzes data, copy that command, and any results of it that you think are relevant, including if it didn’t work, into your lab notebook.

0. Basic Linux navigation

Create directories for Project 1 materials, and inside it create a new directory for raw data.

*UNIX TIP: *Unix has an ‘autocomplete’ feature that will help you correctly type names and paths. If you start typing the command below, i.e. if the directory is called** *"raw_data", you can just type "cd r" and then press the tab key, unix will automatically fill in the rest of the directory name, and you can just hit enter. Try it. If there are multiple options in a file that start with the same letters (ie project1 and project2), when you press tab after you start typing, the shell will autocomplete the shared* part, then wait for you to specify the rest, then you can keep typing and tabbing.

1. Where to get the data

There are several ways to download files in the command line, we will use command *wget*. It is very simple:

```
wget [link address]
```

(Hereinafter square brackets means you have to replace the part in brackets with your relevant information (get rid of the brackets too)).

First, we need the reference sequence of the parental (unevolved, not resistant to antibiotics) *E. coli* strain. You can download it from NCBI FTP:
https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/005/845/GCF_000005845.2_ASM584v2/
We need sequence in fasta format (GCF_000005845.2_ASM584v2_genomic.fna.gz) and annotation in .gff format(*_genomic.gff.gz). This is *E.coli* strain K-12 substrain MG1655, laboratory workhorse. More information about this genome can be found at http://www.ncbi.nlm.nih.gov/genome/167.
Then we will need raw Illumina sequencing reads from shotgun sequencing of an *E. coli* strain that is resistant to the antibiotic ampicillin:
https://doi.org/10.6084/m9.figshare.10006541.v3
(1 and 2 refer to forward and reverse, this was a paired end run).
You may want to copy the reads, reference sequence (.fna) and annotation (.gff) in your working directory for this project.

2. Inspect raw sequencing data manually

We will use a software program that analyzes the overall run quality and makes a report. Before that however, it's good practice to manually verify a few things directly. First, you should always open the sequence files and verify that the format is correct. You don't want to open the entire file, you will just do a quick inspection of the first few reads. Use the head command to look at the first 20 lines in your file (replace 'filename.format' with the actual file name).

```
head -20 [filename.format]
```

Take a look at each file. Each read has 4 lines of information, and then the next read starts on the following line. The first line starts with the @ symbol, and contains identifiers and information about this read. The next line contains the actual sequence, then on line three there is a '+', which may sometimes have the identifier and info repeated.

Line 4 contains the quality string, where ASCII characters encode the quality score for each base. The quality score ranges from 0 to about 40; the higher the number, the greater the accuracy of the base call. To get the actual quality score, you need to figure out the value of the symbol, then subtract 33 (this is called 'sanger' scaling type, because it's the same scaling that people use with traditional sanger sequencing). With some older Illumina data (pre version 1.8), you subtract other numbers, like 64 - this is called Phred33 (sanger) or Phred64 scale, respectively. For more info, see: http://drive5.com/usearch/manual/quality_score.html

Use 'cat' to open the entire fasta reference file. Do you notice anything different about it?

```
cat [filename.fasta]
```

*UNIX TIP: *If you accidentally open a huge file and just see characters flying down your screen, matrix style, or if you think a process you are running may be stuck, you can press **ctrl+c** to cancel the last command.*

Another thing you should check directly from the shell is how many reads are in each fastq file. Some reads will get removed during analysis, so it's important to know what you started with. Use word count with the lines flag to see how many lines there are in each fastq file.

```
wc -l [filename.fastq]
```

From the line count, use what you know about the fastq format to calculate the number of reads in each file, and **record in your lab notebook**.

*UNIX TIP: *If you are going to be executing nearly the same command, with a small change, like you will need to here to get the line count of the _2 file, instead of retyping, you can press up and down at the command prompt to scroll through your recent commands, then use the arrow keys and delete to modify them. Also you can use command history to get the list of the commands you entered - it can save you tons of time if you forgot to write your commands down to your lab notebook.*

3. Inspect raw sequencing data with fastqc. Filtering the reads.

Well, let's start working with tools. And our first tool is fastqc, a simple fastq statistics analysis program. There are few options to install it:
a) Use the package manager Advanced Packaging Tool (APT) to install the program directly from the repository.

```
apt-get install fastqc
```

(for Mac OS it can be `brew install fastqc`)

Maybe it will ask you for permissions – in this case use *sudo* command (*sudo apt-get install fastqc*), to run it as a superuser, with extended permissions.

b) Download fastqc from the website (select proper version for your OS) using *wget*, unpack archive (unzip <*downloaded_file.zip*>) and follow installation instructions.

*UNIX TIP: *If you download the tools instead of using package manager, the good practice is to create separate folder for all your tools, and create link for the executable files to the single location in your PATH variable (e.g. /usr/bin, if you have sudo rights):*

```
ln -s ~/path_to_tool_folder/~/Libs/FastQC/fastqc usr/bin
```

If you forget where your tool is, you can try command *locate*.

c) (**Recommended**) Install it via conda. First, we need to obtain conda - I suggest mambaforge for your system (https://github.com/conda-forge/miniforge). If you have a M1 Mac, use the arm64 version. I recommend using mamba - conda add-on that allows you to install tools waaaay faster.
Also you need to indicate correct channel for the tools to install from (usually bioconda - something like `mamba install -c bioconda fastqc`)
Also, for M1 Macs, check this page on how to manage separate, x86 compatible environment.
TL:DR

```
CONDA_SUBDIR=osx-64 conda create -n myenv_x86 python=3.9
conda activate myenv_x86
conda config --env --set subdir osx-64
```

OK, now you are ready to run fastqc. First, make sure the program is working and properly installed by typing the command below. You should see the manual page. Please raise your hand if this is not working.

```
fastqc -h
```

(generally, the -h or --help flags work with a vast majority of programs, and it gives you an idea how to run it).

Run the program fastqc on the two fastq files. You also have to tell fastqc where to put the output files (use '-o . ' to output files to the current directory, as an example below). You have to specify the full root path to each fastq file (ie /home/project1/raw_data/amp_res_1.fastq).

```
fastqc -o . /pathtofile1/file1.fastq /pathtofile2/file2.fastq
```

Check with 'ls' that this generated some files. To look at the reports, check out html files. **Do the basic statistics match what you calculated for the number of reads last time?** On the left, you'll see a navigation window with green (normal), yellow (slightly abnormal), and red (very unusual) circles for several kinds of data analysis. If you have any red circles, **record them in your notebook**, and read the FastQC documentation on the analysis modules to try to learn what they mean.
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/Help/
Mention the QC results in your lab report. What do you think we should do about anything FastQC identified as unusual?

4. (Optional, 1 bonus point) Filtering the reads

There are a large variety of programs designed to improve the overall quality of your sequencing reads before proceeding with downstream analysis. Removing low quality base calls can improve downstream steps of the analysis. We are going to use a trimming program called Trimmomatic.
Install it using *apt-get install* command.

To learn about this command, just type it as shown below, without any input files. After installation, you can run it like so:

```
TrimmomaticPE
```

*UNIX TIP: If you installed program using apt-get utility, sometimes you have no idea where is the actual files. You see locations of all the installed files using *dpkg -L <packagename>* command. We are usually interested in /usr/bin directory, where all the executable files are located. For trimmomatic you can see TrimmomaticPE and TrimmomaticSE for single and paired reads accordingly).

With paired-end data, alignment algorithms often use the fact that matching forward and reverse reads from the same DNA molecule can't be that far apart - that is because we have DNA fragments of certain length ("insert size") during library preparation.
Based on the read mapping positions, aligners can infer real insert size and its standard deviation, and this information can be used as additional evidence in case of multi-mapped reads. Once a forward read is mapped, the aligner knows it doesn't have to scan the reference sequence very far to find where the corresponding reverse read (from the same molecule) could go.

Because aligners use this information, it's very important to make sure that the number of reads in the forward and reverse files stay coordinated. There are MANY trimming programs out there for next generation sequencing data, but only some that can handle paired end data properly. If you use something other than Trimmomatic in the future, be sure you choose carefully!

We will perform several trimming steps to set up the trimming filter - certain quality threshold. Read more about these steps on the Trimmomatic website – you will need to describe trimming, as one of the data processing steps, in your lab report. There is no optimal trimming strategy, and sometimes it can be useful to do more gentle trimming.

Run Trimmomatic in paired end mode, with following parameters:

- Cut bases off the start of a read if quality below 20
- Cut bases off the end of a read if quality below 20
- Trim reads using a sliding window approach, with window size 10 and average quality within the window 20.
- Drop the read if it is below length 20.

Don’t forget to select the correct phred scale - you can figure it out from the first part of the FastQC reports “Basic statistics” - “Encoding”.

You’ll notice that there are 4 output files. The first and third (_1P.fq and _2P.fq) are the trimmed forward and reverse reads, where BOTH the forward and its matching reverse read passed the trimming filter. Two others (_1U.fq and _2U.fq) are the singleton files contains reads where one read passed the trimming filter, but its partner did not. The reads that do not pass are not output.

Trimmomatic will report back stats, on how many paired reads were kept, but you should check the line count (*wc -l*, divided by 4) of the trimmed paired files manually too. **Record this in your notebook.**

To get see how trimming affected the overall quality of your data, repeat the fastqc analysis you did in section 3, but this time on the _1P.fq and _2P.fq files.

What happens if we increase the quality score at all steps to 30? Try to modify the previous command (be sure to name them something distinct, so as not to overwrite your data).

5 . Aligning sequences to reference

To make sense of our reads, we will map them to our reference sequence (the already-solved genome of normal, non-resistant *E. coli*). Mapping works by taking each read and finding its optimal local alignment in the reference. The read is mapped to a location at which it aligns well (i.e., above some arbitrary score threshold), indicating that the read probably corresponds to that part of the *E. coli* genome. Note that, although you might hope that a read will map to a single position in the genome, it could be that a read maps to multiple locations (e.g. in the case of repeat regions in the genome), and it could also be that a read does not map to anywhere in the genome (e.g. in the case of contamination).

There are many alignment programs ('aligners') available. The earliest alignment algorithms (Smith-Waterman and Needleman-Wunsch) are still used today to compare small pieces of DNA one-by-one, but the computing power needed to map millions of short reads to genomes that are millions to billions of basepairs long requires special algorithms to speed up the process.

I suggest you use an aligner called BWA-MEM, which makes use of the Burrows-Wheeler Transform (BWT). BWA-MEM is optimized for 'long' next-generation sequencing reads of 100bp or more, which may contain several mutations, insertions, or deletions. At first step it looks for the maximum exact matches of seeds, and then it extends the seed using fitting or local alignment (depends on alignment score), to map the entire read.

Selecting the right aligner for your data can be challenging, and there are always new ones coming out, so it's important to keep up with the latest developments and pay attention to benchmarking studies (where researchers compare the performance of different algorithms on the same data set).

5.1 Index the reference file

First, you need to install aligner. It is a part of BWA package – use *apt-get install bwa*. Then you need a reference sequence you downloaded from NCBI FTP .

Next, we have to index the reference file. There is a command in bwa to do this. To see the commands available in BWA, just type *bwa*. To see the options and usage for a specific command, type *bwa* and that command. Do this to see how *bwa index* works.

```
bwa index
```

Based on the usage details you see, run *bwa index* on the reference sequence with the default options. **Record the command you used in your lab notebook.** Check that the command worked with ls. You should see a bunch of new files based on the reference.

5.2 Align your reads

The alignment command takes inputs in this order: first, specify the actual reference name in fasta format (just the original reference sequence, as long as you're in the same directory that you indexed in, bwa will find the new index files). Then specify your actual data in fastq format. (We have paired end data so there are two fastq files, separated by a space). Align your trimmed, paired sequences reference with the command below (all one line). As always, replace the sample filenames below with your ACTUAL file names. **Record the command you used in your lab notebook.**

```
bwa mem [reference_file] [forward_reads] [reverse_reads] >
alignment.sam
```

*UNIX TIP: *The '>' symbol is called a redirect. Redirection takes the output of the preceding command and places it into a new file specified after the '>'.*
If we didn't have it here, bwa mem would just output the results to the standard out (the screen in our case). A lot of tools default their output to stdout, so you can pipe it into other tools. But for now we will be going through this pipeline step by step.

Alignment may take a few minutes, in the meantime, read about SAM format:
BWA outputs data in the "SAM" format. We will dive right in with a manual inspection of the data, but to learn more about sam, check out:
https://samtools.github.io/hts-specs/SAMv1.pdf
That's the official website, you may like the help you get from regular users via google better.

Not all reads will be successfully aligned to the reference. The sam file contains all reads, whether they successfully aligned or not. For bioinformatics pipelines, it's important to know what fraction of your reads aligned. If there are a lot reads that failed to align, that could indicate that your DNA was contaminated with some other source, or that something went wrong with the sequence.

5.3. Compress SAM file

For now we are going to use the built-in utilities of samtools, a program designed to read and parse sam files, to decode the sam file for us.

First, you need to compress and sort the sam file with the commands below. A compressed sam file is called a bam file. We can convert a sam file to a bam file as follows: `samtools view -S -b alignment.sam > alignment.bam`

Next, run the command below to get some basic statistics.

```
samtools flagstat alignment.bam
```

What percentage of reads are mapped?

5..4 Sort and index BAM file

BAM files need to be sorted and indexed. This indexing is different from the reference indexing, we are not using BWT here, we just can get quick access to the positions in BAM file. In case of reference we are building FM-index of the BWT, obtaining quick access to all substrings and positions where they appears.For BAM file indexing, given a coordinate-sorted bam file, you can pull reads from desired positions very quickly, instead of having to iterate over the entire file each time you look for a specific coordinate. This gets the data ready for some of the next commands we will use.

a) Sort bam file by sequence coordinate on reference:

```
samtools sort alignment.bam alignment_sorted
```

(for newer versions of samtools use *samtools sort alignment.bam -o alignment_sorted.bam*)

b) Index bam file for faster search:

```
samtools index alignment_sorted.bam
```

Next you will visualize what the data actually looks like in the IGV browser -
It may be useful to install it locally on your computer. But if you use the online version, when you upload the genome, you may need to upload its index file (`*.fasta.fai`) along with it - obtained, for example, with the `samtools faidx` tool.

Select “Genomes”, ”Load Genome from File” and select our reference genome. Then select “File”, “Open from file” and select your BAM file. Explore visualization.

6. Variant calling

The goal now is to go through our data, and for each position in the reference genome, see how many reads have a mutation at the same position - we want to distinguish actual mutations from the sequencing errors. The solution is to make an intermediate file type called an mpileup, because it goes through each position and “piles up” the reads, tabulating the number of bases that match or don’t match the reference.

Mpileup requires a sorted, indexed bam file. Run the basic command below. It may take a few minutes.

```
samtools mpileup -f [reference fasta file] alignment_sorted.bam >
my.mpileup
```

To call actual variants, we will be using a program called VarScan (variant scanner). On the VarScan man page, you should see several commands. We are interested in the *mpileup2snp* command. Go ahead and enter it with -h to bring up the manual.

```
java -jar VarScan.v2.4.0.jar mpileup2snp -h
```

There are lots of ways we could filter our data. VarScan lets the user define their own cutoffs for including data in calling variants, other programs rely on complex statistical procedures to evaluate the likelihood of real mutations.

The only option we are interested in changing today is the **--min-var-frequency** option. This sets the minimum % of non-reference bases at a position required to call it a mutation in the sample.

Run the program with the threshold you decide is best (replace the N with the decimal value corresponding to the percent you chose, ie 50% = 0.50).
The **--variants** flag tells VarScan to only output positions that are above our threshold, and **--output-vcf 1** option tells it we want the output in yet another kind of data format called **vcf** (variant call format). The command may take a few minutes.

```
java -jar VarScan.v2.4.0.jar  mpileup2snp my.mpileup --min-var-freq N
--variants --output-vcf 1 > VarScan_results.vcf
```

7. Variant effect prediction
The next task is to find out where these mutations are and whether they actually change any proteins in the host (mutations can also occur outside of genes in non-coding regions, or they can be synonymous, where they substitute a codon for the same amino acid).
Go back to your IGV browser and add two more “tracks” - your vcf file and annotation in gff format.

Explore all the mutations, find whether each mutation occurs in a gene, whether it is missense (changes the amino acid sequence), nonsense (introduces a frameshift or early stop codon), or synonymous (no amino acid change). For missense or nonsense mutations find out what that gene name is.

7.1* ADDENDUM Automatic SNP annotation**

It is really easy to make mistakes on the previous step. For each mutation you need to pick up the correct strand - sense or antisense, or "+" and "-". (Note that you can switch between strands in IGV using "reverse complement" option). Then, you need to pick the correct reading frame (based on annotation), mind the direction and manually check if there is a missense or nonsense mutation.

Well, so many ways to screw up. Also, it's just unfeasible if we are dealing with hundreds and thousands of mutations (as in human genomes). Thus, we need to use automatic tools for snp annotation. For this project we can use [SnpEff](http://pcingola.github.io/SnpEff/) (short for "SNP effect").

To annotate vcf file, we need:

1) `snpEff.jar` file (program itself)

You can install it via conda/mamba or download from the web site (http://pcingola.github.io/SnpEff/) - it's Java program, doesn't require installation.

2) sequence and annotation of our reference, to build a custom database. Easiest way - use Genbank format that contains both annotation and sequence:
https://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/005/845/GCF_000005845.2_ASM584v2/GCF_000005845.2_ASM584v2_genomic.gbff.gz

Step 1. Create database

a) create empty text file `snpEff.config`, and add there just one string:

```
k12.genome : ecoli_K12
```

b) create folder for the database

```
mkdir -p data/k12
```

c) Put there your .gbk file (unzip and rename to `genes.gbk`)

```
gunzip GCF_000005845.2_ASM584v2_genomic.gbff.gz
cp GCF_000005845.2_ASM584v2_genomic.gbff data/k12/genes.gbk
```

d) create database

```
snpEff build -genbank -v k12
```

e) annotate

```
snpEff ann k12 VarScan_results_0.5.vcf > VarScan_results_annotated.vcf
```

As a result, you will obtain a vcf file with additional field "ANN" (for "annotation"), describing all the effects for each SNP.

8. Write your lab report.

Hoorayy!!!! You've made it to the end. But you're not done yet! Simply knowing the identity of these mutations won't help the doctor decide how to treat a patient.

For every mutation you found that changes the protein sequence, you must research each gene by name to find out what it does (Ecoli Wiki, EcoCyc, Google, and PubMed are good resources). Try to track down how that function could be involved in antibiotic resistance. If one of the proteins is a gene regulator (transcription factor), try to find out what it regulates. Include what you find in your lab report, and cite where you found it. Include this in the results section.

Also, in your results section, make a table showing how many reads you started with, how many were left after trimming, and how many aligned. We don't need to put the images of the per-position read quality before and after trimming in the main text, but it will be nice to refer to them as Supplementary materials..

It would require additional biochemical testing to verify EXACTLY how each mutation changes the function of the protein it's in. However, your job is to try to come up with a working hypothesis that could explain the mechanism of resistance behind the mutations (for instance, if the mutation is in a gene that makes a protein that is the antibiotic's target, you could propose that the mutation changed the target so the antibiotic couldn't bind anymore). Use what you know about the 4 mechanisms of antibiotic resistance to make predictions (see lecture slides to recall) . Not all of the proteins may be involved in our *E coli* strain's resistance, so you only need to make predictions for THREE of the mutations, but you should research all of them so you are using the ones for which you can best make a logical prediction. Include your predictions for the mechanism of antibiotic resistance for three of the genes in your discussion, and explain the logic behind your predictions.

Finally, make a treatment recommendation for someone infected with this strain of *E coli*. Suggest alternative antibiotics with different targets, and/or secondary therapies that might be useful.