MICRO657 – Lectures 7 - 11: Installing and Using the Celera Assembler

The Celera assembler is one of the first assemblers employed to handle the massive amounts of data generated from the human genome project (HMP). Since its original introduction, it has morphed into a newer generation assembler that capitalizes on de Bruijn graph approaches. However, because of its original algorithmic design, it still performs slower than modern assemblers. As a result, development on the Celera assembler ceased in 2015. Here, we will download and install the Celera assembler and use it to assemble a small genome as an example. This will become the baseline for comparison against more modern day assemblers as we move forward.

The Celera assembler downloads page (https://sourceforge.net/projects/wgs-assembler/files/wgs-assembler/wgs-8.3/) provides a number of download options. Note that the latest version is 8.3rc2, which translates to 8.3 stable version 2. Most available bioinformatics program are downloadable as either pre-compiled binaries, or source code that can be compiled for your specific architecture. For pre-compiled binaries, it is use at your own risk, as they may not be compatible with your current O/S and setup. In general, binaries compiled for unix/linus should work on Ubuntu, and binaries compiled for MacOS (Darwin) should work on all Macs. For our exercise here, we will download the source code and compile it from scratch.

To begin, download the wgs-8.3rc2.tar.bz2 file from the download page, and make a note of where it was saved. Once it has completed download, move it to the MICRO875 folder we created previously on your desktop. If you are on a Mac, the file will automatically unzip and expand into a folder called wgs-8.3rc2 right after download. Move this entire folder to the MICRO875 folder. If you are on a PC using Ubuntu, bring up the bash shell and go to the MICRO875 folder on your desktop where you moved the wgs-8.3rc2.tar.bz2. Here, we will need to unzip and expand the file into a folder (folks on a Mac DO NOT need to do this, as Mac does this automatically). To first unzip the file, run the following command at command line:

```
bzip2 –d wgs-8.3rc2.tar.bz2
```

This will unzip the file and replace it with wgs-8.3rc2.tar. Next we will need to expand the file from its native archive (tar = tape archive) to get the folder. Here we will run the following command:

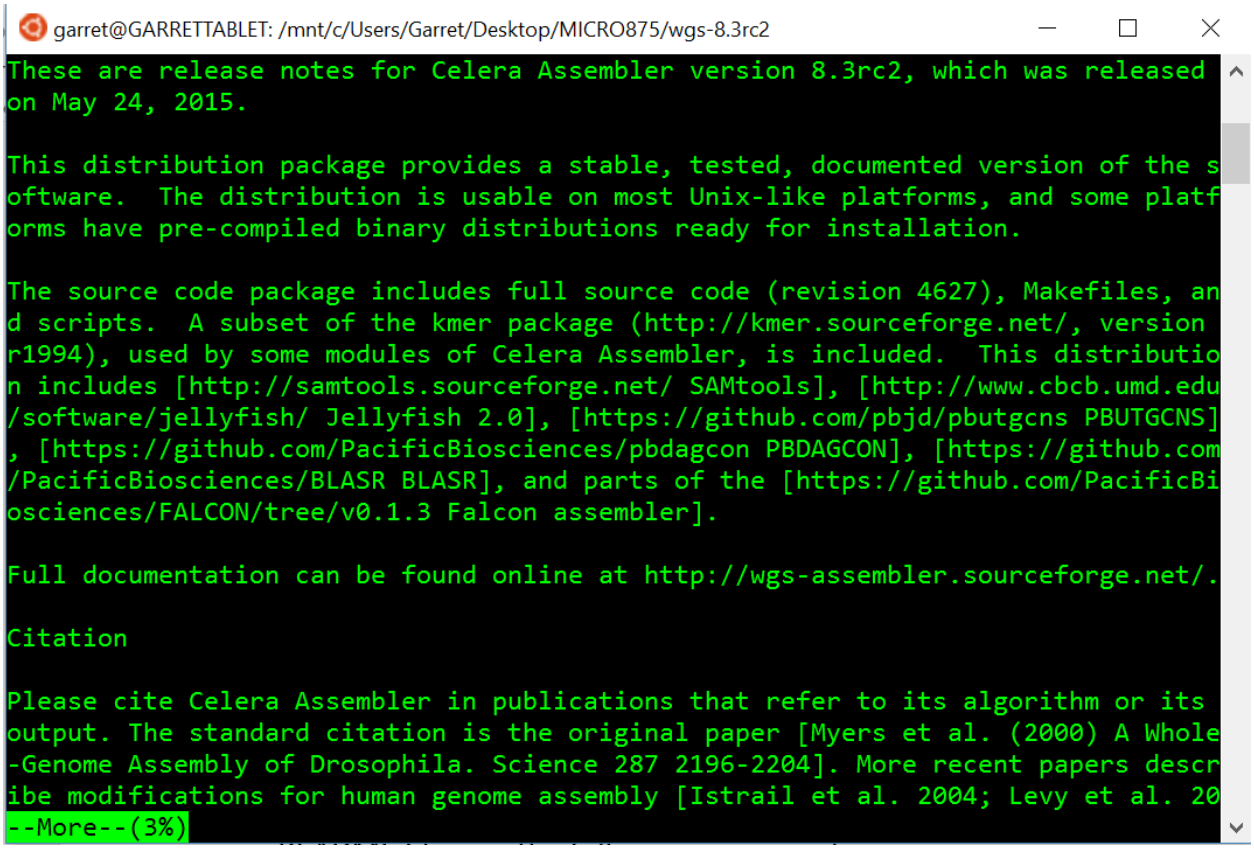
```
tar –xvf wgs-8.3rc2.tar
```

What will happen is that you will see the file being expanded into its original folder path and files. Once completed, a new folder called wgs-8.3rc2 should appear in your folder.

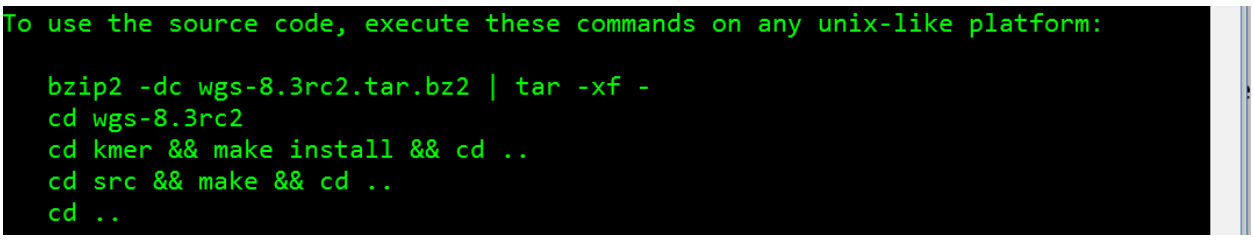
Now, for both Mac and PC folks, go into the wgs-8.3rc2 folder. If you list the contents, you will see that there are a couple of folders (kmer, src) along with a file called README. The README file is very important as it will contain the instructions for compiling the source code into a binary. To view the README file, use the following command:

```
more README
```

You should see something like this:



Use the space bar to scroll through the file until you get to the relevant section describing how you compile the program:



Note that we've already done the first line to unzip and expand the download file (here they have merged both steps into a single command. The second line tells you to go into the wgs-8.3rc2 folder, as we have already done. Next (line 3), it tells you to go into the kmer folder, execute the command "make install" and then go back to the wgs-8.3rc2 folder. If you go ahead and execute these commands, it should go ahead and compile the source code in the kmer file.

Ubuntu: On some installations of Ubuntu, you may get a few errors due to missing installations of key programs that allow compiling. These are "make" and "g++/gcc". If you are missing "make", when you attempt to run "make install", you will get the following error:

```
make: command not found
```

as such, we'll need to install the "make" program. To do so, type the following at command line:

sudo apt-get install make

running the “make install” command may then lead to the following error:

```
make: gcc: command not found
```

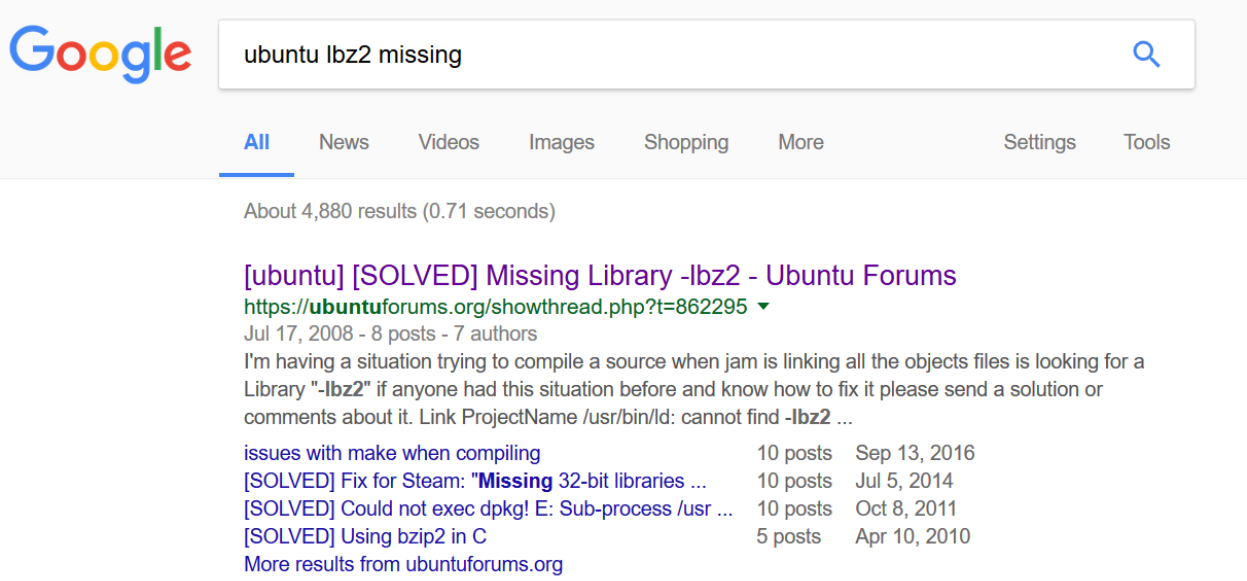
if you get an error like this, it means that the c++ compiler has not been installed, so we’ll have to install it before we can continue. Note that the c++ compiler contains two programs: gcc and g++, so if you ever get this kind of an error, it means that you’ll have to install the compiler. To do this, the following at command line:

sudo apt-get install g++

This will go ahead and install the c++ compiler. Then you can go back and do “make install”. When you do this, you will get the following error:

```
/usr/bin/ld: cannot find -lbz2
collect2: ld returned 1 exit status
```

What this basically means is that the compiler could not fine the library lbz2. Remember that this compiler was last updated in 2015, so it’s likely that it is referencing a library that is old and is no longer included in the 2017 version of Ubuntu. As such, we will have to go and install the library. The easiest way to do this is to do a google search for the library with something like “Ubuntu libz2 missing”. If you do this search, you will note that the first hit that comes up is exactly the scenario that we have come across in our attempt to compile the kmer program:



Reading the post indicates that the way to solve this is to install an old library called libbz2-dev. To do this, run the following command:

sudo apt-get install libbz2-dev

Then run the “make install” command again. At this point, the kmer program should compile with no errors (warnings are fine). Now, going back to our installation instructions, we are told to go into the src folder and compile the complete program (line 4):

```
To use the source code, execute these commands on any unix-like platform:

  bzip2 -dc wgs-8.3rc2.tar.bz2 | tar -xf -
  cd wgs-8.3rc2
  cd kmer && make install && cd ..
  cd src && make && cd ..
  cd ..
```

So, we can go back up into our wgs-8.3rc2 folder, go into the src folder and then simply type make. If you are on a PC with Ubuntu, you will run into another compile error with a second missing library referenced as:

zlib.h : No such file or directory

So, in order to fix this, a google search reveals that we are missing the library zlib1g-dev. To install this, we do something similar to our libbz2-dev installation:

sudo apt-get install zlib1g-dev

Once this is done, we can compile again (type “make”), and it should execute with no errors, and it will create a new folder called “Linux_amd64”.

Mac: If you are a Mac, this should compile perfectly fine, depending on the version of Mac OS X that you have, and will generate a new folder in the wgs-8.3rc2 called “Darwin_amd64”. If you get the following error when you do the “make” in the “src” folder, then you will have to implement a workaround in order to get a clean install.

```
c2/src/AS_MER  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_OVL  -I/Users/g
arretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_OVM  -I/Users/garretsuen/Desktop/MICRO87
5/temp/wgs-8.3rc2/src/AS_OVS  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_
ALN  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_CGB  -I/Users/garretsuen/
Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_BOG  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs
-8.3rc2/src/AS_BAT  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_PBR  -I/Us
ers/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_REZ  -I/Users/garretsuen/Desktop/MI
CRO875/temp/wgs-8.3rc2/src/AS_CNS  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/sr
c/AS_LIN  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_CGW  -I/Users/garret
suen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_TER  -I/Users/garretsuen/Desktop/MICRO875/tem
p/wgs-8.3rc2/src/AS_ENV  -I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/src/AS_REF
-I/Users/garretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/kmer/Darwin-amd64/include -o /Users/g
arretsuen/Desktop/MICRO875/temp/wgs-8.3rc2/Darwin-amd64/obj/gkpStoreDumpFASTQ.o gkpStoreDu
mpFASTQ.C
fastqSimulate-sort.C:236:52: warning: format specifies type 'unsigned long long' but the a
rgument has type 'size_type' (aka 'unsigned long') [-Wformat]
  fprintf(stderr, "Loaded "F_U64" mated reads.\n", reads.size());
                                                  ^~~~~~~~~~~~
fastqAnalyze.C:203:23: error: ordered comparison between pointer and zero ('uint64 (*)[7]'
 and 'int')
    if (freq->tri[ii] > 0)
        ~~~~~~~~~~~~~ ^ ~
1 error generated.
make[1]: *** [fastqAnalyze.o] Error 1
make[1]: *** Waiting for unfinished jobs....
1 warning generated.
make: *** [objs] Error 1
```

The particular error here is the “ordered comparison between pointer and zero” error that is specific to newer compilers. In short, the newer versions of Mac OS X run the newest C/C++ compilers, which are much more strict than the older compilers available back in 2015. As a result, it will throw up errors due to “legacy” code that might cause a run-time error. To resolve this issue, we will have to go and install and old compiler in order to properly compile. At any command prompt, type the following:

brew install gcc@4.9

This will have homebrew install the gcc compiler version 4.9 (note that as of 2017/2018, gcc is at version 7.0). Importantly, it will install the compiler in the following location:

/usr/local/Cellar/gcc@4.9/4.9.4_1/bin

which is what we’ll have to reference to perform our installation.

Once we’ve installed the gcc compiler, we’re going to clean up our previous make install. To do this, exit out of the current directory (you should be in the “src” directory, so at the command prompt type: cd ..), and then enter the “kmer” directory (cd kmer). Then type the following to clean the installation:

make clean

What this will do is remove all of the files that the previous compiler generated so we can do a fresh installation. You should see a number of “rm” commands flash across the screen. Once it is done, we will recompile the install with the new compiler. At the command line, type the following:

make install CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9 CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9

Here we are telling the make command to use the newly installed compilers instead of the default Clang compiler. The CXX and CC flags are variables that point to the newly installed compilers, one for C++ code (CXX) and one for C code (CC). This will then compile the install and we are now ready to perform the master install. Once you are done, get into the “src” folder (cd .. followed by cd src). At the command line, first type:

make clean

then type the following:

make CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9 CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9

Note that this is similar to the previous make command except that we did not specify “install” so as to follow the directions in the README file. This should compile without any errors.

So, if you successfully compiled the program for either PC or Mac, you should now have a new folder called either Linux-amd64 (PC) or Darwin-amd64 (Mac). There will be a folder called “bin” where all of the newly compiled binaries will be stored. As a result, we have now successfully downloaded and compiled the Celera assembler!

Now, in order to properly use the assembler, we have one of two options – run everything in the bin folder of the Linux-amd64 or Darwin-amd64 folders, or figure out a way to allow the binaries to be run from anywhere in the system. To me, the latter approach is much more convenient than having to always go into that bin folder. So, we will go ahead and make this happen. What I usually do, is create a

“bin” folder in my home directory where I store all of my compiled binary files for every program I install. So, go to your home directory and create a new folder called bin. Then, go back to the bin folder in Linux-amd64 or Darwin-amd64 and copy all of the files into your home bin folder. To do this, I ran the following command:

```
cp –R * /home/garret/bin
```

Note that this cp command is a bit different than our regular use. Here, we have used a wildcard (*) to reference all files in the folder. That way, we won’t have to go through and cp each file individually. Lastly, if you did a listing of the bin folder, you’ll notice that there are folders in the bin folder. In order for us to copy folders from one folder to another, we have to use the recursive function (the –R flag). If you execute this successfully, you’ll copy the entire contents of the Linux-amd64/bin or Darwin-amd64/bin folders into your home bin folder (in my case /home/garret/bin). Now that we’ve copied the contents of the bin folder over, we need to think about the other folders in our Linux-amd64 / Darwin-amd64 folder. In this folder, there are three other subfolders besides bin. These include “dep”, “obj”, and “lib”. These are dependencies that programs in the bin folder require in order to run. As such, we will have to preserve the same structure, in that these three subfolders should exist in the same directory where the bin folder is located. Since our bin folder is located in our home directory (/home/garret/bin), we will also need to ensure that the other three subfolders are also found in our home directory (i.e. /home/garret/dep, /home/garret/obj, /home/garret/lib). To do this, we will need to copy those folders over to our home directory in much the same manner as we did for the bin folder. To do this, we can run the following commands from our Linux-amd64 / Darwin-amd64 folder:

```
cp -R dep /home/garret

cp -R obj /home/garret

cp -R lib /home/garret
```

Once this is complete, you should now have a “bin”, “dep”, “obj”, and “lib” folder in your home directory.

Next, we will set up the system such that we can utilize any binary found in the /home/garret/bin folder from anywhere in the system. To do this, we will be adding the /home/garret/bin folder to our system PATH variable. Each time we open up and run the bash shell (i.e. the terminal) specific environment variables are set which allows us to access specific tools such as binaries found in other directories. Environmental variables are defined as beginning with the “$” and the variable in all caps. For example, the variable that defines those directories that binaries can be executed, called the “path” is defined as: $PATH. If you type in $PATH into your terminal it will return the full path to you:

```
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Each directory is split by the “:” symbol. What this means is that anytime and anywhere you type in a command into the command line (e.g. “cd”), it will search each of those directories for a binary that matches that command (the “cd” binary is located in /bin), and execute it. So, what we want to do is add the /home/garret/bin directory to this list so that when we type in any of the celera assembler commands (e.g. meryl), it will search that directory and execute the command regardless of where we are in the system. To do this, we will have to update our $PATH variable by setting it in the profile file

that is read each time you open up the terminal. On a PC with Ubuntu, this file is the .bashrc file that is found in the home directory. On a Mac, it is the .bash_profile file, that is also located in your home directory (note that this file may or may not exist on a Mac. If it does not, we will create it). To accomplish this, run the following command:

PC/Ubuntu: pico .bashrc

Mac: pico .bash_profile

For each of these files, go to the last line and type in the following:

export PATH=/home/garret/bin:$PATH

replacing "/home/garret/bin" with your home directory bin folder.

Save the file by holding down the Ctrl key and hitting the "x" key. It will ask you if you want to save the file (hit 'y') and then it will ask you what name you wish to give the file (it will automatically insert the name you specified in the pico command). Hit enter, and it will save the file.

Close the terminal, and open up a new terminal to load the new profile file. Now, you should be able to run any of the Celera assembler commands from anywhere in the system. To test this, type RunCA at the command prompt, and we should hopefully get the help options.

However, when we do this, we end up getting an interesting error which indicates that it cannot find the program "gatekeeper" in /home/garret/Linux-amd64/bin. This is a rather odd error that is rather specific. So what is going on here? Essentially, what it is saying is that it is looking for the program "gatekeeper" in a subfolder of our home directory called "Linux-amd64/bin". Note that when we compiled the Celera assembler, it placed everything into our Linux-amd64 / Darwin-amd64 folder. We had attempted to recreate the same file structure (i.e. bin, dep, lib, and obj) in our home directory, but it seems that the program is requiring that everything be found in a Linux-amd64 / Darwin-amd64 folder. This is an odd quirk of the Celera assembler in that it hard codes this into the executables. As such, we will necessarily have to ensure that our original Linux-amd64 / Darwin-amd64 is preserved. So, the easiest way to do this is to copy the entire Linux-amd64 / Darwin-amd64 folder into our home/garret/bin folder and then update our $PATH to point to /home/garret/bin/Linux-amd64/bin. It's a bit clunkier, but it does still keeps everything in our home bin folder. So, in order to do this, let's first undo everything that we did before. We will first remove the entire contents of our home bin folder, and then remove the obj, deb, and lib folders in our home directory. To do this, we can type the following at the command line in our /home/garret/bin folder:

rm -R *

Then we can move to our home directory and remove the other folders as follows:

rm -R obj

rm -R dep

rm -R obj

Then, finally we can copy the entire Linux-amd64 / Darwin-amd64 folder into our /home/garret/bin folder. To do this, go to the wgs-8.3rc2 folder and type the following:

cp -R Linux-amd64 /home/garret/bin

This will copy the entire contents of our Linux-amd64 / Darwin-amd64 folder into our home bin folder. Next, we will have to update our path. To do this, use pico or nano to edit your .bashrc or .bash_profile file. At this point, we can simply add /home/garret/bin/Linux-amd64/bin to our path:

export PATH=/home/garret/bin:/home/garret/bin/Linux-amd64/bin:$PATH

Now we can restart our terminal, and type runCA at the command prompt, which will result in the following output:

```
garretsuen@Garret-SurfacePro: ~
garretsuen@Garret-SurfacePro:~$ runCA
usage: runCA -d <dir> -p <prefix> [options] <frg> ...
  -d <dir>          Use <dir> as the working directory.  Required
  -p <prefix>       Use <prefix> as the output prefix.  Required

  -s <specFile>     Read options from the specifications file <specfile>.
                      <specfile> can also be one of the following key words:
                      [no]OBT - run with[out] OBT
                      noVec   - run with OBT but without Vector

  -version          Version information
  -help             This information
  -options          Describe specFile options, and show default values

  <frg>             CA formatted fragment file

Complete documentation at http://wgs-assembler.sourceforge.net/

Assembly name prefix not supplied with -p.
Directory not supplied with -d.
```

which is the help display associated with the program that is provided when you run runCA without the proper file inputs.

After accomplishing all of this, you have now properly downloaded, compiled, and installed the Celera Assembler!

Now that we've installed the Celera assembler, we can go ahead and perform an example assembly. For this, we will follow the example assembly provided by the Celera assembler for the genome of *Yersinia pestis* KIM, which can be found here: http://wgs-assembler.sourceforge.net/wiki/index.php/Yersinia_pestis_KIM_D27,_using_Illumina_paired-end_reads,_with_CA8.2. In this example, we will assemble a paired-end 2x100 bp Illumina dataset for this bacterium. The first task is to create a new directory where the data will be stored, and then

fetching that data from the NCBI's SRA. Here, we are taught to use the "curl" command, which is one of the most useful command line tools for grabbing data directly from an FTP server. To execute these commands, make sure you are in the MICRO875 folder, which is where you will create the READ-sra folder as described, and then fetch the sra file from NCBI's SRA. Note the logic of the FTP path, which is the standard path for all SRA Accession numbers. This is important to know, especially if you find yourself needing to download sra files from other accession numbers reported in literature. Here is the breakdown for the *Y. pestis* KIM accession:

```
ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-
instant/reads/ByRun/sra/SRR/SRR133/SRR133640/SRR133640.sra
```

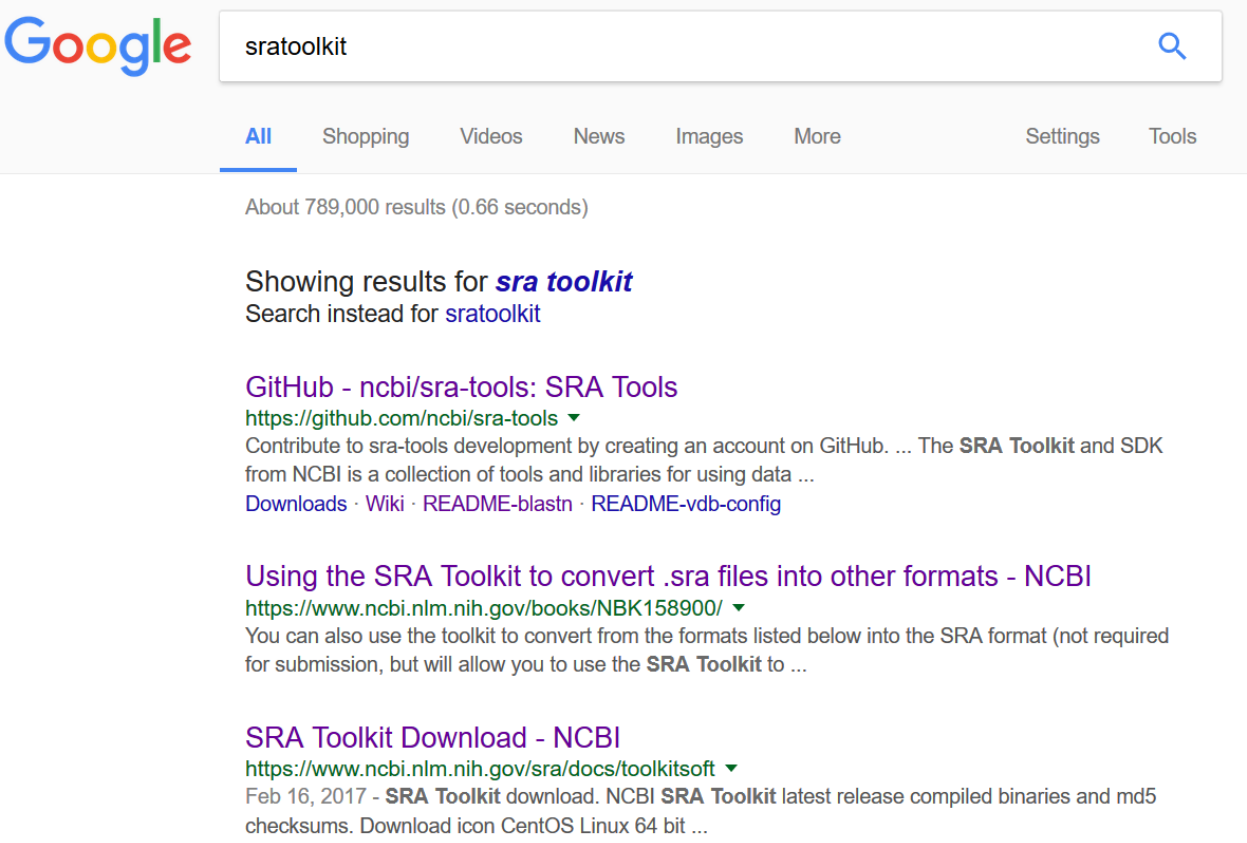
The key pieces here are to note that the accession number itself is SRR133640. The "SRR" part refers to the fact that it was submitted directly to the short read archive itself. You can also submit raw data to both the European and Japanese versions of the SRA, and they will have the three-letter acronym "ERR" and "DRA", respectively. The next thing to note is that each sra file gets its own unique folder, which is the penultimate portion of the FTP listing (/SRR133640/). This is the then preceded by a folder designating the three-letter acronym (SRR) and the first three digits of the accession (133). In this way, all sra files starting with "SRR133" are collected into this folder. Finally, moving one step up, we see that the folder "SRR" contains all of the SRR files. Again, this would be changed to "ERR" or "DRA" if we were accessing data submitted to those respective databases (note that every night, the SRA updates all data from the other two databases). So, if we had a completely different accession such as SRR566089.sra, the ftp string would look like this:

```
ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-
instant/reads/ByRun/sra/SRR/SRR566/SRR566089/SRR566089.sra
```

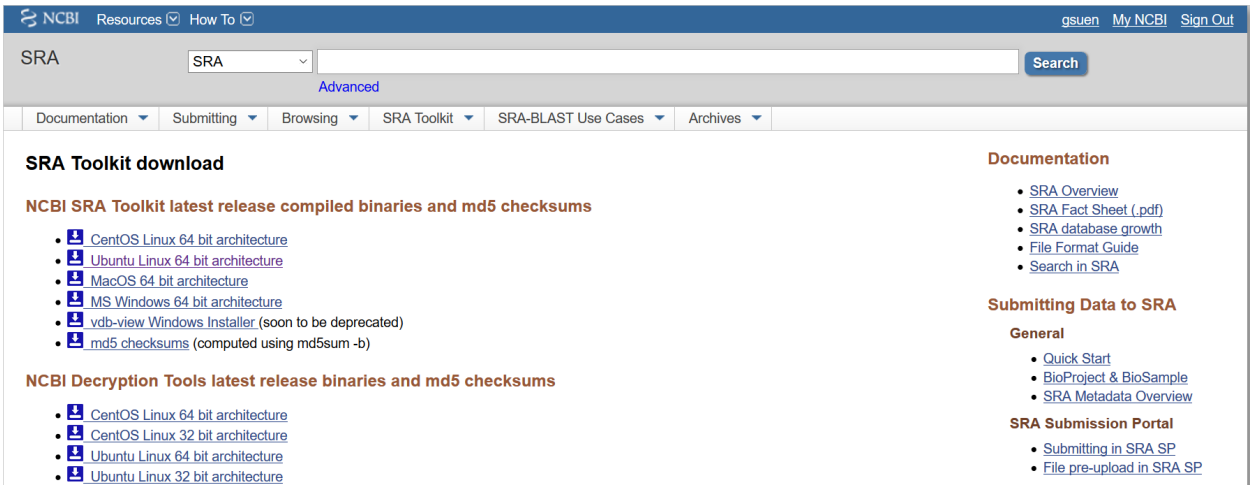
If we were accessing a dataset deposited to the European database (the ENA) under the accession ERR468790.sra, if would look like this:

```
ftp://ftp-trace.ncbi.nlm.nih.gov/sra/sra-
instant/reads/ByRun/sra/ERR/ERR468/ERR468790/ERR468790.sra
```

So, moving along, we see that the next step is to obtain the data we need from the sra file we've downloaded. Since Illumina's default output is a .fastq file, there are specific tools that we can use to pull that information from the .sra file. Note that the .sra file is a special datafile for the SRA that is able to take data from any assembler (e.g. Illumina, 454, Sanger, PacBio, etc) and codify it into a uniform datatype. Thus, we need to pull the information relevant to us from the .sra file. In this case, we will be using a specific command line tool called "fastq-dump" from the sra-toolkit. First, we'll need to download and install the sra-toolkit. In this case, we can simply go and grab the binaries and copy them over to a folder found in our PATH that we set up last time. To do this, we can go search for the sra-toolkit using google. (Note that if you are on an Ubuntu box, the sra-toolkit is actually part of the repository, and so you can install it automatically by typing at the command prompt: sudo apt-get install sra-toolkit. It does not look like the sra-toolkit is available via homebrew, so if you are on a Mac, you will necessarily have to install it by downloading the binaries as follows).



Here, we'll want the 3rd link, as this will contain the binaries we need. Select the version that is appropriate for your system, such as Ubuntu Linux 64 bit architecture, if you are running Ubuntu on Windows 10.



Once you have downloaded the file, you should be able to unzip and untar it to reveal the binaries themselves. You can then copy them to a folder in your path (e.g. the bin folder in your home directory, or /usr/local/bin). Once this is done, we can use the fastq-dump command on the SRR133640.sra file we downloaded, as described in the example. At this point, we can essentially follow the example all the

way through to completion. A couple of notes – fastq-dump is very poorly documented by the NCBI, and so a good site to visit, which describes many of the options is here: https://edwards.sdsu.edu/research/fastq-dump/.

For the first command in the Correct section of the example:

```
cat SRR133640_1.fastq SRR133640_2.fastq | fastq-to-fasta-merged.pl >
SRR133640.merged.fasta
```

Note that they make the fastq-to-fasta-merged.pl Perl script available for download (http://wgs-assembler.sourceforge.net/wiki/index.php/Fastq-to-fasta-merged.pl). On some machines, you may have to invoke the perl command in order for the command to work. This will look like this:

```
cat SRR133640_1.fastq SRR133640_2.fastq | perl fastq-to-fasta-merged.pl >
SRR133640.merged.fasta
```

For the histogram that is shown at the end of the Correct section, this is not in the .histogram file, but a graphic that you can generate in Excel using the data in the .histogram file.

Finally, you will run into an error when you do the assembly, as executed, which is:

sh: 1: qsub: not found

You can do a google search for this error, and a bit of rooting around will reveal that the issue is due to the assembly referencing the Sun Grid Engine (SGE). To solve this, you can remove the references to SGE and it will run with no problems:

```
runCA -p ypestis -d ypestis-raw doOBT=1 unitigger=bogart READS-
sra/SRR133640.frg
```