

**Title:** Automating the process of mapping NextGen Sequence Data back to a sequenced genome.

## Introduction

NextGen sequencing, which includes Illumina (<http://www.illumina.com/index.html>) and 454 (<http://www.454.com>) sequencing technologies, have emerged as revolutionary methods to sequence genomes at great depths. These technologies have become so widely used in almost all research institutions, that you, as a Bioinformatician, you will be called upon regularly to work on the data sets they produce.

These technologies generate millions of short reads (<120 bp) that are hard or impossible to assemble de-novo (without the use of some sort of reference or map). In this demonstration, you will learn how to use a tool, called Bowtie (<http://bowtie-bio.sourceforge.net/bowtie2/index.shtml>) to align Illumina reads to a reference genome. The sample reads you will use are from an RNA-Seq experiment, meaning the sequence reads are from mRNA that were extracted from the biological samples, converted to cDNA and then sequenced using Illumina. The same tools are used for DNA genomic sequence assembly; the difference is that in RNA-Seq you align the reads to RNA transcripts or CDS (DNA coding regions) of a specific genome. The reason being is that you want to count how many reads hit a particular transcript or CDS (i.e. gene), the number of reads that align to each gene corresponds to the degree of transcription that the gene is undergoing in the biological sample. Usually you have two samples, an experimental and a control, and the read numbers are compared between the experiment versus the control, to see how gene expression has changed in the former.

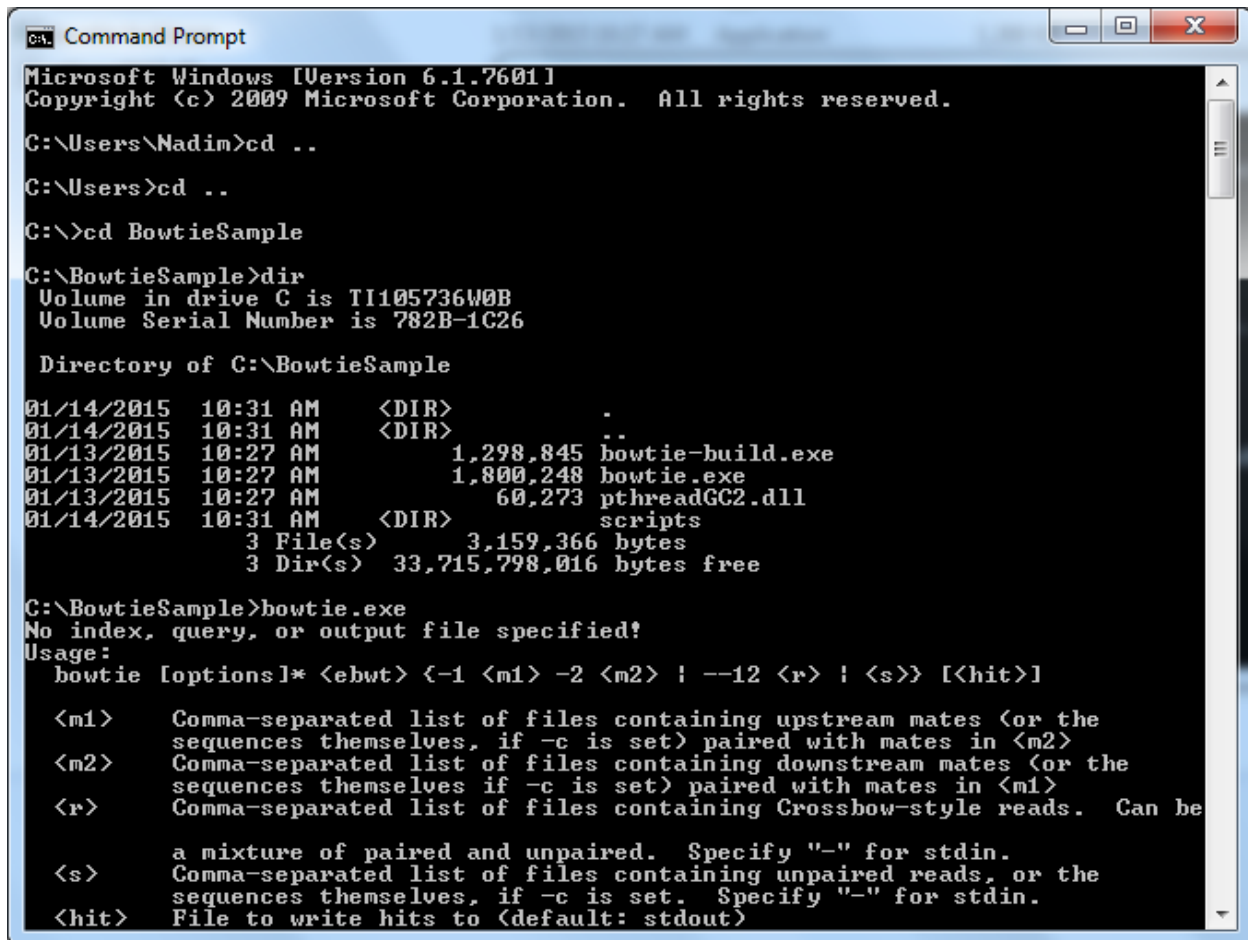
You will also learn in this demonstration how to use a scripting language like Perl or Python to automate this process. You will write a script that automatically calls (executes) Bowtie, and process the data till the final output is achieved.

## Learning Demonstration

- 1- Bowtie can be found in the attached compressed folder, along with the test data that we will use. Bowtie is freely available from here:  
<http://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.2.4/>
- 2- Un-compress the folder onto your computer, you can place it directly onto your C drive for easy access. Bowtie comes as a group of executable files, that don't need to be installed on Windows. All you need to do is call them, as we will see below. In Linux/Mac you'll just have to place them in your working/bin directory.
- 3- Test the installation by opening a command prompt window (see snap shot below), change directory to the uncompressed folder (type "`cd C:\BowtieSample`" and hit enter). BowtieSample being the name of the folder that you uncompressed and placed on your C drive. If you changed the name, or placed it elsewhere just provide

the correct name and full path so your OS can find it.

- 4- You will see the directory change to C:\BowtieSample. You can view a list of files/folders in this directory by using the command *dir* in Windows, or *ls* in Linux/Mac.



```
C:\Users\Nadim>cd ..
C:\Users>cd ..
C:\>cd BowtieSample
C:\BowtieSample>dir
Volume in drive C is TI105736W0B
Volume Serial Number is 782B-1C26

Directory of C:\BowtieSample

01/14/2015  10:31 AM    <DIR>          .
01/14/2015  10:31 AM    <DIR>          ..
01/13/2015  10:27 AM             1,298,845  bowtie-build.exe
01/13/2015  10:27 AM             1,800,248  bowtie.exe
01/13/2015  10:27 AM              60,273  pthreadGC2.dll
01/14/2015  10:31 AM    <DIR>          scripts
                3 File(s)      3,159,366 bytes
                3 Dir(s)   33,715,798,016 bytes free

C:\BowtieSample>bowtie.exe
No index, query, or output file specified!
Usage:
  bowtie [options]* <ebwt> <-1 <m1> -2 <m2> ! --12 <r> ! <s>> [<hit>]

<m1>    Comma-separated list of files containing upstream mates (or the
sequences themselves, if -c is set) paired with mates in <m2>
<m2>    Comma-separated list of files containing downstream mates (or the
sequences themselves if -c is set) paired with mates in <m1>
<r>      Comma-separated list of files containing Crossbow-style reads.  Can be
a mixture of paired and unpaired.  Specify "-" for stdin.
<s>      Comma-separated list of files containing unpaired reads, or the
sequences themselves, if -c is set.  Specify "-" for stdin.
<hit>    File to write hits to (default: stdout)
```

- 5- Now type the command *Bowtie.exe* and you will see the list of arguments it takes.
- 6- Before we start to use Bowtie to align our reads however, we will need a reference genome. The genome sequence has to be indexed (formatted in a specific way) before Bowtie can use it. Pre-indexed genomes are available to download from Bowtie's web site (they include the most common organisms, such as human, rat, mouse...etc.). If you are working with an organism that does not have a pre-indexed genome or you have your own genome you want to work with, you can index it yourself using the included *bowtie-build.exe* tool (included with bowtie, and in the BowtieSample folder).
- 7- We will call the *bowtie-build.exe* and *bowtie.exe* within Python, so open up your favorite IDE or editor (such as IDLE or spyder). And create a new .py file for your code. Save the file inside BowtieSample so it has access to the bowtie executables, or you can provide the path to where the executables are if you save your source code

elsewhere.

- 8- To call tools and other programs on your OS within your Python program, we will need to use specific Python functions (described here: <https://docs.python.org/3.8/library/subprocess.html>). For our purposes we will use the *subprocess.call()* function.
- 9- In your IDE type the command:

```
import subprocess
subprocess.call("bowtie-build.exe strawberry.fa
strawberry_genes");
```

You can see here that I've given *bowtie-build* two arguments: *strawberry.fa* which is the genome sequences that I want indexed (located inside the folder), and then a name for my genome, in this case I just called it *strawberry\_genes*. You can pick whatever name you like for your genome! Just make sure it's descriptive.

Test this out by saving your source file and running the code above, you will notice that Python automatically opens up a command prompt window and executes the statement inside *subprocess.call()*. If you look inside BowtieSample folder now you will notice additional files, all starting with the word "strawberry", these are the index files that bowtie needs!

- 10- Now it's time to actually call bowtie, we can use the *subprocess.call()* command again, add this to your source code:

```
subprocess.call("bowtie.exe strawberry_genes -q
sample1_reads.fq -S Sample1.sam");
```

- 11- You can see from the command above that *bowtie.exe* takes 3 arguments, the 1<sup>st</sup> is the name of your indexed genome, in this case strawberry, the second is the raw reads you want to align against the genome, denoted by the *-q* followed by the name of the file (called *sample1\_reads.fq* in our case), and the 3<sup>rd</sup> is output file name, denoted by *-S*. For the latter I chose to output the results in what is known as SAM format (text format), there is also another binary format (called BAM). For purposes though we need the text version, which will parse out below.
- 12- Open the resulting output (*Sample1.sam*) in Notepad++ or any text editor and check it out. You will notice that each read (represented by its ID) is mapped to a gene (represented by the word gene followed by its number, i.e. gene00001, gene00002..etc.).
- 13- What we need to do next is open the *Sample1.sam* file from within our Python script and parse out the results. Parsing simply means to extract out the relevant information from a file. We can open the file using the open command as we've seen

before:

```
SAMFile = open("Sample1.sam")
```

14- The SAM output file contains comments (denoted by @), read IDs, genes that they mapped to and positions within those genes to name a few). What we need for RNA-Seq analysis though is just the number of reads that hit each gene. To get that we need to count how many times a specific gene ID (denoted by gene####) occurred in the file. If for instance gene00001 is found 1000 times in the file, it means there are a 1000 reads aligned to it! From a biological perspective it also means that they were 1000 mRNAs for that gene in my original biological sample that was sequenced.

15- Now it's time to think about the algorithm to parse out the SAM file. Look at the SAM file again and think about what types of looping structures and/or regular expression you need to count the occurrence of each gene?

16- You've probably come to the conclusion to use a while loop, like we did before, to read in the file line by line and look for lines that have the word gene#### in them!

17- But how are we going to store the gene IDs and their associate read counts? What data structure would be most appropriate? A variable? An array? Or perhaps a dictionary? The latter, as you recall, is an associate array, where each "key" is associate with a "value". That sounds perfect for this situation doesn't it? The key in this case is the gene ID and its value is the read counts.

18- So let's create the variables we need (this should always be done on the top of your source code):

```
myDict={} #This is the dictionary that will store our read
counts for each gene
myKey="" #This variable will store the gene ID's
myValue=0 #This variable will store the read counts
```

19- Now let's start to read in the file, line by line and look for the lines that contain gene####:

```
import re
for line in SAMFile:
    m = re.search("(gene.*?)\s", line)
```

20- Let's look closely at the regular expression (*gene.\*?*)\s, what are we doing here? We are looking for a word that starts with "gene" followed by any number of characters ".\*" till we hit a space "\s". The whole match (gene.\*?) is enclosed in parenthesis. As you recall Python will extract whatever is in the () and store it as an object

(group(1) below). If you had something else to extract, you could place another set of () around it and Python will store that match in *group(2)* and so on. Please refer to the content area for more information on regular expressions.

Now that we have the gene ID extracted and stored we need to place it in the dictionary. What we want to do is create a new dictionary entry with that gene ID and set the read count (value) to 1:

```
if m:
    myKey = m.group(1)
    myDict[myKey] = 1
```

21- As we extract more gene IDs though we'll have to check and see if the dictionary already has it, if it was already parsed out we need to increment the value (the read count) by 1. We can accomplish this by using a conditional statement:

```
if myKey in myDict:
    myDict[myKey]++
```

But what if the dictionary already contains the gene ID? In that case we need to create a new entry for it like we did above and assign its value 1. Basically we need to combine both code snippets above into an if...else statement for this to work:

```
if myKey in myDict:
    myDict[myKey]++
else:
    myKey = m.group(1)
    myDict[myKey] = 1
```

22- After the loop has finished reading all the lines of the file, all genes and their counts should have been extracted and counted, and stored in the dictionary.

23- All we have to do now is print out the results. We could print out to the screen, but it would be better in this case to print out the results to a file. That way we can import it into Excel or any relational database for further processing and analysis:

```
#open a file for the information to be sent to
OUTPUT = open ("ReadCountsS1.txt", "w")
```

```
#create a header line, with the words GeneID and Read counts
separated by a tab
OUTPUT.write("GeneID \t Read counts\n")
```

```
#for each gene in the SAM file, the number of times it is seen
is printed out
for key in myDict:
    myValue = myDict[key]
    OUTPUT.write(key + " \t " + val + "\n")

print ("Output file with read counts has been generated.\n")
OUTPUT.close()
```

24- Open the output file in Excel; it should be located in BowtieSample. This is a tab delimited file, so choose that option when asked by Excel. You will see that Excel generated 2 columns, one for GeneID and one for read counts.

25- Sort the Excel file by read counts, what gene is the most expressed in this sample? Why?

26- You will notice that the genes don't have descriptions; we could blast their sequences and find out what they are though! We will do this in the next learning demonstration.

### **Deliverable:**

Submit your complete source code (.py) file, with comments, along with the Excel file. You do not need to submit any of the test files.