Read Mapping with bowtie2 Tutorial GVA2019

Overview

Once you know you are working with the best quality data (Evaluating Raw Sequencing data tutorial) that you can be, the first step in nearly every next-gen sequence analysis pipeline is to map sequencing reads to a reference genome. In this tutorial we'll explore these basic principles using **bowtie2** on TACC.

The world of read mappers seems to be settling down a bit after being a bioinformatics Wild West where there was a new gun in town every week that promised to be a faster and more accurate shot than the current record holder. Things seem to have reached the point where there is mainly a trade-off between speed, accuracy, and configurability among read mappers that have remained popular. There are over 50 read mapping programs listed here. Each mapper has its own set of limitations (on the lengths of reads it accepts, on how it outputs read alignments, on how many mismatches there can be, on whether it produces gapped alignments, on whether it supports SOLiD colorspace data, etc.). As evidence of how things are settling down, we're going to just use bowtie2 in this course.

> **Other read mappers**
>
> Previous versions of this class and tutorial have covered using bowtie and bwa. Please consult these tutorials for more specific information on each mapping program. A previous version of this tutorial included a trimmed down version of the bwa tutorial if you just want the 'flavor' of what other read mappers involve.

As this is the first tutorial of the day be sure that you start a new idev session:

```
idev  -m 180 -r CCBB_Day_2 -A UT-2015-05-18
```

Learning Objectives

This tutorial covers the commands necessary to use bowtie2 to map reads to a reference genome, and concepts applicable to many more mappers.

- Become comfortable with the basic steps of indexing a reference genome, mapping reads, and converting output to `SAM/BAM` format for downstream analysis.
- Use **bowtie2** to map reads from an *E. coli* Illumina data set to a reference genome and compare the output.

Theory

Please see the Introduction to mapping presentation for more details of the theory behind read mapping algorithms and critical considerations for using these tools and references correctly.

Mapping tools summary

The tutorial currently available on the Lonestar cluster at TACC is as follows:

Tool	TACC	Version	Download	Manual	Example

Bowtie2	`module load bowtie/2.3.4` You may recall we added this to our .bashrc file yesterday so it is already loaded	2.3.4	link	link	#Bowtie2

Modules also exist on lonestar5 for **bwa**.

Tutorial: *E. coli* genome re-sequencing data

The following DNA sequencing read data files were downloaded from the NCBI Sequence Read Archive via the corresponding European Nucleotide Archive record. They are Illumina Genome Analyzer sequencing of a paired-end library from a (haploid) *E. coli* clone that was isolated from a population of bacteria that had evolved for 20,000 generations in the laboratory as part of a long-term evolution experiment (Barrick et al, 2009). The reference genome i s the ancestor of this *E. coli* population (strain REL606), so we expect the read sample to have differences from this reference that correspond to mutations that arose during the evolution experiment.

Transferring Data

We have already downloaded data files for this example and put them in the path:

```
$BI/gva_course/mapping/data
```

You may recognize this as the same files we used for the fastqc and cutadapt tutorial. If you chose to improve the quality of R2 reads using cutadapt as you did for R1 in the tutorial, you could use the improved reads in this tutorial to see what a difference the improved reads can make for read mapping.

File Name	Description	Sample
`SRR030257_1.fastq`	Paired-end Illumina, First of pair, FASTQ format	Re-sequenced *E. coli* genome
`SRR030257_2.fastq`	Paired-end Illumina, Second of pair, FASTQ format	Re-sequenced *E. coli* genome
`NC_012967.1.gbk`	Reference Genome in Genbank format	*E. coli* B strain REL606

The easiest way to run the tutorial is to copy this entire directory into a new folder called "**GVA_bowtie2_mapping**" on your **$SCRATCH** space and then run all of the commands from inside that directory. See if you can figure out how to do that. When you're in the right place, you should get output like this from the `ls` command.

```
tacc:~$ ls
NC_012967.1.gbk  SRR030257_1.fastq  SRR030257_2.fastq  SRR030257_2.fastq.gz
```

Remember that to copy an entire folder requires the use of the recursive (-r) option.

Still stuck? click here for the correct code

```
cds
cp -r $BI/gva_course/mapping/data GVA_bowtie2_mapping
cd GVA_bowtie2_mapping
ls
```

Useful commands

Often you will have general questions about your sequencing files that you want to answer before or after starting your actual analysis. Here we show you some very handy commands after a warning:

Beware the cat command when working with NGS data

NGS data can be quite large, a single lane of an Illumina Hi-Seq run generates 2 files each with 100s of millions of lines. Printing all of that can take an enormous amount of time and will likely crash your terminal long before it finishes. If you find yourself in a seemingly endless scroll of sequence (or anything else for that matter) remember ctrl+c will kill whatever command you just executed

Reminder about Linux 1 liners

Below are several commands we've already been using, and some new ones put together to improve your skills.

How to look at the top/bottom of files to determine their type

```
head * # will show the top 10 lines of all files to give you an idea of the file type/structure
tail * # will show the last 10 lines of all files to determine if the file is a repetitive structure
```

How to count the total number of lines in a file

```
wc -l *  # can be very useful to determine if it can be printed to screen or opened in a text editor
```

How to determine the total number of sequences in a fastq file

```
wc -l *  # and then divide by 4 using the your knowledge of fastq files

# OR
grep ^@SRR030257 SRR030257_1.fastq | wc -l

# OR
grep --count ^@SRR030257 SRR030257_1.fastq

# OR
grep --count "^+$" SRR030257_1.fastq
```

How to determine how long the reads are in a fastq file

```
sed -n 2p SRR030257_1.fastq | awk -F"[ATCGatcg]" '{print NF-1}'
```

Converting sequence file formats

Occasionally you might download a sequence or have it emailed to you by a collaborator in one format, and then the program that you want to use demands that it be in another format. Why do they have to be so picky? Everybody has own favorite formats and/or those that they are the most familiar with but humans can typically pick the information they need out of comparable formats. Programs can only be written to assume a single type of format (or allow you to specify a format if the author is particularly generous), and can only find things in single locations based on that format.

The `bp_seqconvert.pl` script is a common script written in Bioperl that is a helpful utility for converting between many common sequence formats. On TACC, the Bioperl modules are installed, but the helper script isn't. So, we've put it in a place that you can run it from for your convenience. However, remember that any time that you use the script you must have the bioperl module loaded. We also took care of this for you when we edited your `~/.bashrc` file in the Linux introduction.

Run the script without any arguments to get the help message:

```
module load gcc
module load bioperl
bp_seqconvert.pl
```

Exercises

The file `NC_012967.1.gbk` is in `Genbank` format. The files `SRR030257_*.fastq` are in `FASTQ` format.

- Convert `NC_012967.1.gbk` to `EMBL` format. Call the output `NC_012967.1.embl`.
    - Does EMBL format have sequence features (like genes) annotated?

      Try reading through the program help when you run the bp_seqconvert.pl without any options to see the syntax required

Sill need help?

```
bp_seqconvert.pl --from genbank --to embl < NC_012967.1.gbk > NC_012967.1.embl
head -n 100 NC_012967.1.embl
```

You might get an error or a warning like the following, even if the bp_seqconvert.pl script executed correctly so don't worry.

```
Use of uninitialized value in substitution (s///) at /opt/apps/bioperl/1.6.901/Bio/SeqIO/embl.pm
line 777, <STDIN> line 164674.
Use of uninitialized value in concatenation (.) or string at /opt/apps/bioperl/1.6.901/Bio/SeqIO
/embl.pm line 779, <STDIN> line 164674.
```

From the head command, you should see that yes, EMBL files do maintain gene annotation features.

- Convert only the first 10,000 lines of `SRR030257_1.fastq` to FASTA format.
  - What information was lost by this conversion?

Remember use the | character to have the output of head feed into the bp_seqconvert.pl script.

Click here for the answer

```
head -n 10000 SRR030257_1.fastq | bp_seqconvert.pl --from fastq --to fasta > SRR030257_1.fasta
head SRR030257_1.fastq
head SRR030257_1.fasta
```

The line of ASCII characters was lost. Remember, those are your "base quality scores". Many mappers will use the base quality scores to improve how the reads are aligned by not placing as much emphasis on poor bases.

Mapping with bowtie2

Bowtie2 is a complete rewrite of **bowtie**. It is currently the latest and greatest in the eyes of one very picky professor (and his postdoc) in terms of configurability, sensitivity, and speed. After years of teaching bwa mapping along with bowtie2, bowtie2 alone is now taught as I never recommend anyone use bwa, and based on positive feedback we continue with this set up. For some more details about how read mappers work see the bonus presentation, and if you find a compelling reason to use bwa (or any other read mapper) rather than bowtie2, I'd love to hear from you.

Create a fresh output directory named bowtie2. We are going to create a specific output directory for the bowtie2 mapper **within** the directory that has the input files so that you can compare the results of other mappers if you choose to do the other tutorials.

Commands for making a directory and changing into it

```
mkdir bowtie2
```

First you need to convert the reference file from GenBank to FASTA using what you learned above. Name the new output file `NC_012967.1.fasta` and put it in the same directory as `NC_012967.1.gbk`.

Use the bp_seqconvert.pl script

Click here to get the answer or to check your command

```
bp_seqconvert.pl --from genbank --to fasta < NC_012967.1.gbk > NC_012967.1.fasta
```

Next, we want to make sure the **bowtie2** module is loaded (we use `module spider` to get the current name, which may not be `bowtie/2.3.4` if you re-run this tutorial later):

Remember in our earlier tutorial we discussed the use of lonestar's `module` commands "spider" and "load" to install new functionality and "list", "keyword", and "avail" to find different modules.

click here for the best answer

```
module list bowtie
```

Unexpected output

```
# Currently Loaded Modules Matching: bowtie
#  None found.
# Inactive Modules Matching: bowtie
#   1) bowtie/2.3.4
```

Further, when we try to load bowtie/2.3.4 we get an error message.

```
Lmod has detected the following error:  These module(s) exist but
cannot be loaded as requested: "bowtie/2.3.4"
   Try: "module spider bowtie/2.3.4" to see how to load the module(s).
```

See if you can figure out how to load bowtie using the information above.

Click here for answer

```
module load intel/18.0.2
module load bowtie/2.3.4
```

When we loaded the bioperl module we first loaded the gcc compiler which unloaded several other modules (such as bowtie) which require the intel compiler to function. If you now try to load bioperl you'll see that it loads without requiring the gcc compiler. This was done deliberately to introduce you to another quirk of the module system. Hopefully the error messages were informative enough to help you work through how to get the modules to work.

Despite these quirks, this is still far easier to deal with than issues that can arise when installing other programs on tacc or your personal computer. Here are a few of the possibilities that will work.

In this case all of these methods will work, that may not be true of all programs

```
bowtie2 --version
module list
which bowtie2
```

Note that **`which`** can be very useful for making sure you are running the executable that you think you are running, especially if you install your own programs. In particular make sure that the path matches up to what you expect. The most common situations arise from wanting to run a simplistically named script in your $HOME directory conflicting with something of the same name in the $BI directories or TACC modules.

For many read mappers, the first step is quite often indexing the reference file regardless of what mapping program is used. Put the output of this command into the `bowtie` directory we created a minute ago. The command you need is:

```
bowtie2-build
```

Try typing this alone in the terminal and figuring out what to do from the help show just from typing the command by itself.

The command requires 2 arguments. The first argument is the reference FASTA. The second argument is the "base" file name to use for the created index files. It will create a bunch of files beginning bowtie/NC_012967.1*.

Click here to check your work, or for the answer if needed

```
bowtie2-build NC_012967.1.fasta bowtie2/NC_012967.1
```

Take a look at your output directory using `ls bowtie2` to see what new files have appeared. These files are binary files, so looking at them with `head` or tail isn't instructive and can cause issues with your terminal. If you insist on looking at them (or accidentally do so before you read this) and your terminal begins behaving oddly, simply close it and log back into lonestar with a new terminal, and start a new idev session.

Why do so many different mapping programs create an index as a first step you may be wondering?

Like an index for a book (in the olden days before Kindles and Nooks), creating an index for a computer database allows quick access to any "record" given a short "key". In the case of mapping programs, creating an index for a reference sequence allows it to more rapidly place a read on that sequence at a location where it knows at least a piece of the read matches perfectly or with only a few mismatches. By jumping right to these spots in the genome, rather than trying to fully align the read to every place in the genome, it saves a ton of time.

Indexing is a separate step in running most mapping programs because it can take a LONG time if you are indexing a very large genome (like our own overly complicated human genome). Furthermore, you only need to index a genome sequence once, no matter how many samples you want to map. Keeping it as a separate step means that you can skip it later when you want to align a new sample to the same reference sequence.

Finally, map the reads! The command you need is:

```
bowtie2
```

Try reading the help to figure out how to run the command yourself. Remember these are paired-end reads.

IMPORTANT

This command can take a while (~5 minutes) and is extremely taxing. This is longer than we want to run a job on the head node (especially when all of us are doing it at once). In fact, in previous years, TACC has noticed the spike in usage when multiple students forgot to make sure they were on idev nodes and complained pretty forcefully to us about it. Let's not have this be one of those years. Use the showq -u command to make sure you are on an idev node. If you are not scroll up to the start of this tutorial or see yesterday's tutorial for more information. If you are not sure if you are on an idev node, raise your hand and we'll show(q) -u what you are looking for. Yes, your instructor likes bad puns. My apologies.

Solution

```
bowtie2 -t -x bowtie2/NC_012967.1 -1 SRR030257_1.fastq -2 SRR030257_2.fastq -S bowtie2/SRR030257.sam  # the -t command is not required for the mapping, but it can be particularly informative when you begin comparing different mappers
```

Your final output file is in SAM format. It's just a text file, so you can peek at it and see what it's like inside. Two warnings though:

1. SAM files can be enormously humongous text files (potentially >1 gigabytes). Attempting to open the entire file at once can cause your computer to lock up or your text editor to crash. You are generally safer only looking at a portion at a time using linux commands like **head** or **grep** or **more** or using a viewer like IGV, which we will cover in a later tutorial.
2. SAM files have some rather complicated information encoded as text, like a binary encoded FLAGS field and CIGAR strings. We'll take a look at some of these later, if we have time, or they are covered in the bonus presentation.

Still, you should recognize some of the information on a line in a SAM file from the input FASTQ, and some of the other information is relatively straightforward to understand, like the position where the read mapped. Give this a try:

```
head bowtie2/SRR030257.sam
```

If you thought the answer was the mapping coordinates of the read pairs you were right!

More reading about SAM files

- The official SAM file specification
- http://genome.sph.umich.edu/wiki/SAM

Multithreaded execution

We have actually massively under-utilized Lonestar in this example. We ran the command using only a single processor (a single "thread") rather than the 48 we have available. For programs that support multithreaded execution (and most mappers do because they are obsessed with speed) we could have sped things up by using all 48 processors for the bowtie process.

You need to use the `-p`, for "processors" option. Since we had 48processors available to our job.

click here to check your answer

```
bowtie2 -t -p 48 -x bowtie2/NC_012967.1 -1 SRR030257_1.fastq -2 SRR030257_2.fastq -S bowtie2/SRR030257.sam
```

Try it out and compare the speed of execution by looking at the log files.

1 processor took ~5 minutes, 48 processors took ~ 15 seconds minute. Can you think of any reasons why it was ~ 16x faster rather than 48x faster?

Anytime you use multiprocessing correctly things will go faster, but even if a program can divide the input perfectly among all available processors, and combine the outputs back together perfectly, there is "overhead" in dividing things up and recombining them. These are the types of considerations you may have to make with your data: When is it better to give more processors to a single sample? How fast do I actually need the data to come back?

If you want to launch many processes as part of one job, so that they are distributed one per node and use the maximum number of processors available, then you need to learn about the "wayness" of how you request nodes on Lonestar and possibly edit your launcher.slurm script in the future (more on this on Friday), or make better use of running commands in the background using the **&** symbol at the end of the command line.

One consequence of using multithreading that might be confusing is that the aligned reads might appear in your output SAM file in a different order than they were in the input FASTQ. This happens because small sets of reads get continuously packaged, "sent" to the different processors, and whichever set "returns" fastest is written first. You can force them to appear in the same order (at a slight cost in speed) by adding the `--reorder` flag to your command, but is typically only necessary if the reads are already ordered or you intend to do some comparison between the input and output.

Optional Exercises for your free time

- In the **bowtie2** example, we mapped in `--local` mode. Try mapping in `--end-to-end` mode (aka global mode).
- Do the **BWA** tutorial so you can compare their outputs.
    - Did **bowtie2** or **BWA** map more reads?
    - In our examples, we mapped in paired-end mode. Try to figure out how to map the reads in single-end mode and create this output.
    - Which aligner took less time to run? Are there any options you can change that:
        - Lead to a larger percentage of the reads being mapped? (increase sensitivity)
        - Speed up performance without causing many fewer reads to be mapped? (increase performance)

Next steps...

The next steps are often to view the output using a specific viewer on your local machine, or to begin identifying variant locations where the reads differ from the reference sequence. These will be the next things we cover in the course. Here is a link to help you return to the GVA 2019 course schedule.