Mapping tutorial

Overview

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 run some common mapping tools 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. We're going to (mainly) stick to just two or three in this course.

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.).

Learning Objectives

This tutorial covers the commands necessary to use several common read mapping programs.

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

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 correctly.

Table of Contents

Mapping tools summary

The three tools that we show detailed instructions for in this tutorial and their versions currently available on the Lonestar cluster at TACC:

Tool	TACC	Version	Download	Manual	Example
Bowtie	`module load bowtie/0.12.8`	0.12.8	link	link	#Bowtie
BWA	`module load bwa/0.6.2`	0.6.1; 0.6.2	link	link	#BWA
Bowtie2	`module load bowtie/2.0.2`	2.0.2	link	link	#Bowtie2

Modules also exist at the current time for: `SHRiMP` and `SOAP`.

Example: *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 is 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.

Data

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

```
$BI/ngs_course/intro_to_mapping/data
```

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 to 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.

```
login1$ ls
NC_012967.1.gbk  SRR030257_1.fastq  SRR030257_2.fastq
```

```
cds
cp -r $BI/ngs_course/intro_to_mapping/data intro_to_mapping
cd intro_to_mapping
```

Exercises

- What basic linux commands could help us quickly peek at the files that were just copied to get an idea of their contents?

  ```
  head
  tail
  wc -l
  ```

- How many sequences are in the file `SRR030257_1.fastq`?

  ```
  grep ^@SRR030257 SRR030257_1.fastq | wc -l
  grep --count ^@SRR030257 SRR030257_1.fastq
  ```

  Or you could just count the number of lines and divide by 4!

  ```
  wc -l SRR030257_1.fastq
  ```

- How many bases long are the reads in `SRR030257_1.fastq`?

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

... and there must be more clever ways than this.

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?

The `bp_seqconvert.pl` script that is installed as part of Bioperl is one 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 `~/.profile_user` file in the Linux introduction.

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

```
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?

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

      ```
      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 funny ASCII characters was lost. Those are your "base quality scores".

Extra reading

http://en.wikipedia.org/wiki/FASTQ_format
http://en.wikipedia.org/wiki/GenBank
The first portion of a Genbank file contains information about "features" of the genome, like genes. The second part contains the actual bases of the reference sequence. Therefore, a Genbank essentially has an embedded FASTA file inside it. There are also a lot of nice statistics and metadata, like the size of the sequence and its base composition in the GenBank header.

Mapping with bowtie

Load the bowtie module

```
module load bowtie
```

What version of bowtie was loaded?

```
bowtie --version
module list
```

Create a fresh output directory. (We are going to create a different output directory for each mapper that we try **within** the directory that has the input files.)

```
mkdir bowtie
```

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`.

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

Next, index the reference file. Put the output of this command into the `bowtie` directory. The command you need is:

```
bowtie-build
```

Try typing this alone in the terminal and figuring out what to do from the help.

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

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*.

Take a look at your output directory using `ls bowtie` to see what new files have appeared. These files are binary files, so looking at them with `head` isn't instructive.

Why do we create an index?

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:

```
bowtie
```

Try reading the help to figure out how to run the command yourself. This command takes a while (~5 minutes). 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, TACC noticed the spike in usage last time we taught the class and we got in trouble.

So, you will want to submit the full job to the cluster like you learned in the introduction.

But first, try to figure out the command and start it in interactive mode. Remember these are paired-end reads. **Use `control-c` to stop the job once you are sure it is running without an immediate error!** Then, submit your command that is working to the TACC queue.

Submit to the TACC queue or run in an idev shell

Create a `commands` file and use ***launcher_creator.py*** followed by ***qsub***.

Put this in your `commands` file:

```
bowtie -t --sam bowtie/NC_012967.1 -1 SRR030257_1.fastq -2 SRR030257_2.fastq bowtie/SRR030257.sam
```

What does the `-t` option do?

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 (maybe >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 using a viewer like IGV, which we will cover later.
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.

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 bowtie/SRR030257.sam
```

What do you think the 4th and 8th columns mean?

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 submitted a job that reserved a single node on the cluster, but that node has 12 processors. Bowtie was only using one of those processors (a single "thread")! 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 12 processors for the bowtie process.

It's `-p`, for "processors". Since we had 12 processors available to our job, the better bowtie alignment `commands` file would look like this.

```
bowtie -p 12 -t --sam bowtie/NC_012967.1 -1 SRR030257_1.fastq -2 SRR030257_2.fastq bowtie/SRR030257.sam
```

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

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 *.sge script. 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.

Mapping with BWA

BWA (the Burrows-Wheeler Aligner) is another fast mapping program. It's the successor to another aligner you might have used or heard of called MAQ (Mapping and Assembly with Quality).

The steps of running BWA are very similar to running bowtie.

Load the module:

```
module load bwa
```

There are multiple versions of BWA on TACC, so you might want to check which one you have loaded for when you write up your awesome publication that was made possible by your analysis of next-gen sequencing data.

```
module spider bwa
module list
bwa
```

Create a fresh output directory, so that we don't write over the output from bowtie. Be sure you are back in your main `intro_to_mapping` directory. Then:

```
mkdir bwa
```

Try to figure out how to index and map from the command line help:

```
bwa
```

You will need to run this set of commands (with options that you should try to figure out) in this order:

```
bwa index
bwa aln
bwa samse or sampe
```

What's going on at each step?

Remember to use the option that enables multithreading, if there is one, for each BWA command.

First, run the index command (`index`) on the reference file. This is fast, so you can run it interactively.

BWA doesn't give you a choice of where to create your index files. It creates them in the same directory as the FASTA that you input. So copy the FASTA in your `intro_to_mapping` directory to your new bwa directory:

```
cp NC_012967.1.fasta bwa
```

Then, run the index command using the copied FASTA as input.

```
bwa index bwa/NC_012967.1.fasta
```

Take a look at your output directory using `ls` bwa to see what new files appear after indexing.

Then, run the mapping command (`aln`). Note that you need to map each set of reads in the pairs separately with BWA because of how it separates the initial mapping and the later alignment steps.

Submit to the TACC queue or run in an idev shell

Create a `commands` file and use ***launcher_creator.py*** followed by ***qsub***.

Put this in your `commands` file:

```
bwa aln -t 6 -f bwa/SRR030257_1.sai bwa/NC_012967.1.fasta SRR030257_1.fastq
bwa aln -t 6 -f bwa/SRR030257_2.sai bwa/NC_012967.1.fasta SRR030257_2.fastq
```

Why did we use `-t 6` instead of `-t 12` for multithreading? Both of our commands are going to go to a single node on Lonestar, so they should share the 12 available cores.

Again, take a look at your output directory using `ls` bwa to see what new files have appeared. What is a *.sai file? It's a file containing "alignment seeds" in a file format specific to BWA. Many programs produce this kind of "intermediate" file in their own format and then at the end have tools for converting things to a "community" format shared by many downstream programs.

We still need to extend these seed matches into alignments of entire reads, choose the best matches, and convert the output to SAM format.

Do we use `sampe` or `samse`?

Submit to the TACC queue or run in an idev shell

Create a `commands` file and use ***launcher_creator.py*** followed by ***qsub***.

Put this in your `commands` file:

```
bwa sampe -f bwa/SRR030257.sam bwa/NC_012967.1.fasta bwa/SRR030257_1.sai bwa/SRR030257_2.sai
SRR030257_1.fastq SRR030257_2.fastq
```

Mapping with Bowtie2

Bowtie2 is a complete rewrite of Bowtie. It is currently the latest and greatest (in the eyes of one very picky instructor) in terms of configurability, sensitivity, and speed.

The steps of running bowtie2 are very similar to running bowtie1.

Create a fresh output directory.

```
mkdir bowtie2
```

Load the module (we use `module spider` to get the current name, which may not be `bowtie/2.0.2` when you run this tutorial):

```
module spider bowtie
module load bowtie/2.0.2
```

See if you can do all the steps **on your own**. You want output in a file called `SRR030257.sam` to be in SAM format.

Index the reference file and map the reads:

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

Submit to the TACC queue or run in an idev shell

Use ***launcher_creator.py*** followed by ***qsub*** to submit a commands file containing:

```
bowtie2 -p 12 -x bowtie2/NC_012967.1 -1 SRR030257_1.fastq -2 SRR030257_2.fastq -S SRR030257.sam
```

Exercises

- Did bowtie, BWA, or bowtie2 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.
- What are the limitations of each read mapper?
  - Which would we want to use if we were mapping 454 data?
- Which aligner took less time to run?
  - Are there any options you can change so that more reads are mapped or that speed up performance without causing many fewer reads to be mapped?

Alternative Exercise (MAQ)

- MAQ (the predecessor to BWA) is also available as a module on TACC. You can also try figuring out how to run it on this example data. It was created before SAM format, so you have to use a super-secret ninja command to convert its output to SAM:

  ```
  module load samtools
  $TACC_SAMTOOLS_DIR/misc/maq2sam-short
  ```

  Other than that, you should be able to use the manual to get through the steps.

From here...

From here you can use the output SAM files to continue on to the Introduction to variant calling (SAMtools) or immediately view your mapped reads in the Integrative Genomics Viewer (IGV) tutorial.