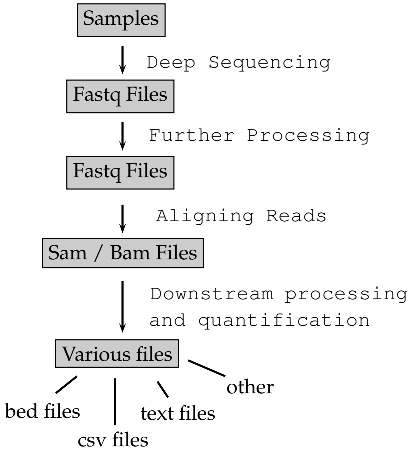
Genomic Files

University of Massachusetts Medical School

November, 2016

A Typical Deep-Sequencing Workflow



Deep Sequencing Data pipelines involve a lot of text processing.

This is an oversimplified model and your workflow can look different from this!

Toolbox

Unix has very useful tools for text processing.
Some of them are:

- Viewing: less
- Searching: grep
- Table Processing: awk
- Editors: nano, vi, sed

Searching Text Files

Problem

Say, we have our RNA-Seq data in fastq format. We want to see the reads having three consecutive A's. How can we save such reads in a separate file?

grep is a program that searches the standard input or a given text file *line-by-line* for a given text or pattern.

$$\texttt{grep}\quad \underbrace{\texttt{AAA}}_{\text{text to be searched for}}\quad \underbrace{\texttt{control.rep1.1.fq}}_{\text{Our text file}}$$

For a colorful output, use the `--color=always` option.

```
$ grep AAA control.rep1.1.fq --color=always
```

Using Pipes

We don’t want grep print everything all at once.
We want to see the output line-by-line.
Pipe the output to `less`.

```
$  grep AAA control.rep1.1.fq --color=always | less
```

Using Pipes

We don’t want grep print everything all at once.
We want to see the output line-by-line.
Pipe the output to `less`.

```
$  grep AAA control.rep1.1.fq --color=always | less
```

We have escape characters but less don’t expect them by default. So

```
$ grep AAA control.rep1.1.fq --color=always | less -R
```

Unix Pipes

Unix pipes direct the (standard) output of the LHS of | to the RHS of | as standard input.

```
$ command_1 | command_2 | ··· | command_n
```

The (standard) output of **command_i** goes to **command_i+1** as (standard) input.

Exercise

Submit two dummy jobs to the long queue and three short dummy to the short queue. Then get a list of your jobs that have been submitted to the **long** queue **only**.

Hint 1: Use `sleep 300` to create a dummy job.
Hint 2: Use **bsub** to submit a job. Remember that `-q` parameter is used to specify the queue.
Hint 3: Recall that **bjobs** can be used to list your jobs in the cluster.
Hint 4: Use what you have learned so far and put the pieces together.

Exercise

Submit two dummy jobs to the long queue and three short dummy to the short queue. Then get a list of your jobs that have been submitted to the **long** queue **only**.

Hint 1: Use `sleep 300` to create a dummy job.
Hint 2: Use **bsub** to submit a job. Remember that `-q` parameter is used to specify the queue.
Hint 3: Recall that **bjobs** can be used to list your jobs in the cluster.
Hint 4: Use what you have learned so far and put the pieces together.

```
$ bsub -q short "sleep 300"
```

```
$ bsub -q long "sleep 300"
```

Exercise

Submit two dummy jobs to the long queue and three short dummy to the short queue. Then get a list of your jobs that have been submitted to the **long** queue **only**.

Hint 1: Use `sleep 300` to create a dummy job.
Hint 2: Use **bsub** to submit a job. Remember that `-q` parameter is used to specify the queue.
Hint 3: Recall that **bjobs** can be used to list your jobs in the cluster.
Hint 4: Use what you have learned so far and put the pieces together.

```
$ bsub -q short "sleep 300"
```

```
$ bsub -q long "sleep 300"
```

```
$ bjobs | grep long
```

Exercise

Submit two dummy jobs to the long queue and three short dummy to the short queue. Then get a list of your jobs that have been submitted to the **long** queue **only**.

Hint 1: Use `sleep 300` to create a dummy job.
Hint 2: Use **bsub** to submit a job. Remember that `-q` parameter is used to specify the queue.
Hint 3: Recall that **bjobs** can be used to list your jobs in the cluster.
Hint 4: Use what you have learned so far and put the pieces together.

```
$ bsub -q short "sleep 300"
```

```
$ bsub -q long "sleep 300"
```

```
$ bjobs | grep long
```

Homework: Read the manual page of `bqueues` and find a way to do this without using a pipe.

What about saving the result?

We can make grep print all the reads we want on the screen.

But how can we save them? View them better?

For this we need to redirect the **standard output** to a textfile.

```
$ grep AAA control.rep1.1.fq > ~/AAA.txt
```

Standard Input, Output and Error

When a process is started, by default, several places are setup for the process to read from and write to.

- **Standard Input:** This is the place where process can read input from. It might be your keyboard or the output of another process.
- **Standard Output:** This is the place where the process writes its output.
- **Standard Error:** This is the place where the process writes its error messages.

By default, all these three places point to the terminal. Consequently, standard output and error are printed on the screen and the standard input is read from the keyboard.

Redirecting Standard Input, Output and Error

- We can redirect the standard output using ">".
  Let's have the output of echo to a text file.

```
$ echo echo hi > out.txt
```

- We can redirect the standard input using "<".
  Let's use the file we created as input to bash.

```
$ bash < out.txt
```

- We can redirect the standard error using "2>".
- We can redirect both the standard output and error using "&>".

Fastq Files

As the ultimate product of sequencing, for each fragment of DNA, we get three attributes.

- Sequence Identifier
- Nucleotide Sequence
- Sequencing quality per nucleotide

The sequencing information is reported in **fastq** format. For each sequenced read, there are four lines in the corresponding fastq file.

Fastq Example

`@61DFRAAXX100204:2`	←	Identifier
`ACTGGCTGCTGTGG`	←	Nucleotide Sequence
+	←	Optionally Identifier + description
`789::=<<==;9<==<;;`	←	Phred Quality
`@61DFRAAXX100304:2`	←	Identifier
`ATAATGAGTATCTG`	←	Nucleotide Sequence
+	←	Optionally Identifier + description
`4789;:=<=:«=:`	←	Phred Quality
⋮	⋮	⋮

Some aligners may not work if there are comments after the identifier (read name).

There are 4 rows for each entry. This is a simplified example and the actual sequences and the identifiers in a fastq file are longer.

Phred Quality Score

The sequencer machine is not error-free and it computes an error probability for each nucleotide sequenced.
Say, for a particular nucleotide position, the probability of reporting the wrong nucleotide base is P, then

$$Q_{Phred} = -10 \times \log_{10} P$$

is the *Phred Quality Score* of the nucleotide position.

Phred Quality Score

The sequencer machine is not error-free and it computes an error probability for each nucleotide sequenced.
Say, for a particular nucleotide position, the probability of reporting the wrong nucleotide base is P, then

$$Q_{Phred} = -10 \times \log_{10} P$$

is the *Phred Quality Score* of the nucleotide position.

The above formula is for Sanger format which is widely used today.
For Solexa format, a different formula is used.

Phred Quality Score

Q_{Phred} is a number. But we see a character in the fastq file. How do we make the conversion?

There are two conventions for this.

1. Phred 33
2. Phred 64

ASCII

ASCII TABLE	
Decimal	**Character**
0	NULL
⋮	⋮
33	!
34	"
⋮	⋮
64	@
65	A
⋮	⋮
90	Z
⋮	⋮
97	a
⋮	⋮
122	z
⋮	⋮
127	DEL

ASCII printable characters start at the position 33. The capital letters start at position 65.

Phred 33: The character that corresponds to $Q_{Phred} + 33$ is reported.

Phred 64: The character that corresponds to $Q_{Phred} + 64$ is reported.

Phred Example

Suppose that the probability of reporting the base in a particular read position is $\frac{1}{1000}$. Then

$$Q_{Phred} = -10 \times \log_{10} \frac{1}{1000} = -10 \times \log_{10} 10^{-3} = 30$$

Using Phred 33: 30+33 = 63 $\rightarrow$?

Using Phred 64: 30+64 = 94 $\rightarrow$ ^

Exercise

From a big fastq, you randomly pick one million nucleotides with Phred 33 quality reported as **I**. In how many nucleotides, amongst a total of one million nucleotides, would you expect to be a sequencing error?

Exercise

From a big fastq, you randomly pick one million nucleotides with Phred 33 quality reported as **I**. In how many nucleotides, amongst a total of one million nucleotides, would you expect to be a sequencing error?

In the ASCII table, the decimal number corresponding to **I** is 73. For Phred 33, we have

$$73 - 33 = 40 = -10 \times \log_{10} P \quad \rightarrow P = 10^{-4}$$

We have 1 million nucleotides with a probability 10^{-4} of sequencing error. So, we expect to see $10^{6} \times 10^{-4} = 100$ reads with a sequencing error.

grep: filtering out

Say we want to find reads that **don't** contain AAA in a fastq file, then we use the `-v` option to filter out reads with AAA.

```
$ grep -v AAA file.fastq
```

More on Text Filtering

Problem

*How can we get **only** the nucleotide sequences in a fastq file?*

Problem

How can we get only particular columns of a file?

awk

`awk` is an interpreted programming language desgined to process text files. We can still use awk while staying away from the programming side.

$$\texttt{\textbf{awk}}\ \underbrace{\text{'}\{\texttt{\textbf{print(\$2)}}\}\text{'}}_{\text{awk statement}}\quad \underbrace{\texttt{\textbf{sample.sam}}}_{\text{columns sep. by a fixed character (def: space)}}$$

Some Awk Built-in Variables

Content	**Awk variable**
Entire Line	$0
Column 1	$1
Column 2	$2
⋮	⋮
Column i	$i
Line Number	NR

Example

Say, we only want to see the second column in a sam file,

```
$ awk '{print($2)}' sample.sam
```

Getting nucleotide sequences from fastq files

In fastq files, there are 4 lines for each read. The nucleotide sequence of the reads is on the second line respectively. We can get them using a very simple modular arithmetic operation,

```
$ awk '{if(NR % 4== 2)print($0)}' file.fq
```

NR = line number in the given file.

Exercise

Using `awk`, get the sequencing quality from the fastq file.

Exercise

Using `awk`, get the sequencing quality from the fastq file.

```
$ awk '{if(NR % 4== 0)print($0)}' file.fq
```

Unix pipes

awk can be very useful when combined with other tools.

Problem

*How many reads are there in our fastq file that don't have the seqeunce **GC**?*

```
$  awk '{if(NR % 4== 2)print($0)}' file.fq | grep -v GC
```

gives us all such reads. How do we find the number of lines in the output?

Find sequences ending with AAA

Let’s find all the sequences in our fastq file that ends with AAA using awk.

```
$ awk '{if(NR % 4== 2){if(substr( $0, length($0)-2, length($0) )=="AAA") print($0)}}'\
file.fq
```

Exercise

Using awk, find all sequences **starting** with AAA in a fastq file.

Exercise

Using awk, find all sequences **starting** with AAA in a fastq file.

```
$ awk '{if(NR % 4== 2){if(substr( $0, 1, 3 )=="AAA") print($0)}}'\
file.fq
```

wc

`wc:` gives us the number of lines, words, and characters in a line.

with the `-l` olption, we only get the number of lines.
Hence

```
$   awk '{if(NR % 4== 2)print($0)}' file.fq | grep -v GC | wc -l
```

gives us the number of reads that don't contain the sequence `GC` as a subsequence.

Example

Fasta File Format:
>Chromosome (or Region) Name
Sequence (possibly separated by new line)
>Chromosome (or Region) Name
Sequence (possibly separated by newline)

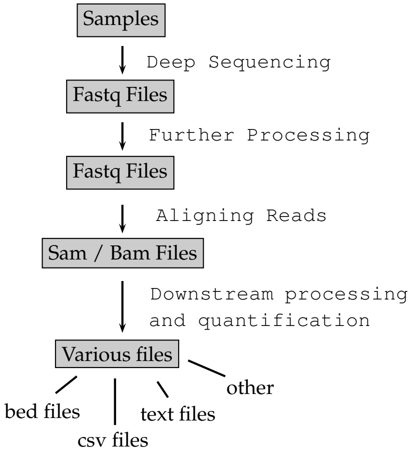
Let's find the number of chromosomes in the mm10.fa file. Each chromosome entry begins with ">", we get them by

```
$ grep ">" mm10.fa
```

Then we count the number of lines

```
$ grep ">" mm10.fa | wc -l
```

SAM / BAM Files



When a fastq file is aligned against a reference genome, a sam or a bam file is created as the ultimate output of the alignment. These files tell us where and how reads in the fastq file are mapped.

Sequence Aligners

$$\text{Fastq File} \xrightarrow{\text{Aligner}} \text{Sam / Bam File}$$

Short (Unspliced) Aligners	Spliced Aligners
Bowtie2	Tophat
BWA	STAR

miRNA Data:
Continuous reads so **Bowtie2** or **BWA** would be a good choice.

RNA-Seq Data:
Contains splice junctions, so **Tophat** or **STAR** would be a good choice.

Contents of A Sam / Bam File

Say a particular read is mapped somewhere in the genome by an aligner.

- Which chromosome?
- What position?
- Which strand?
- How good is the mapping?
- Are there insertions , deletions or gaps?

are some of the fundamental questions we ask on the alignment. A sam / bam file contains answers for these questions and possibly many more.

Sam is a text, Bam is a binary format

Recall:
A **text file** contains printable characters that are meaningful for us. It is big.
A **binary file** (possibly) contains nonprintable characters. Not meaningful to humans. It is small.

Sam File: Text file, tab delimited, big
Bam File: Binary file, relatively small

A bam file is a compressed version of the sam file and they contain the same information.

It is good practice to keep our alignment files in bam format to save space. A bam file can be read in text format using samtools.

Mandatory Fields of a Sam File

Col	Field	Type	Regexp/Range	Brief Description
1	QNAME	String	[!-?A-~] $\{1,255\}$	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	* | [!-()+-⟨⟩ -~][!-~]*	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping Poaition
5	MAPQ	Int	$[0,2^{8}-1]$	Mapping Quality
6	CIGAR	String	*|([0-9]+[MIDNSHPX=])+	CIGAR string
7	RNEXT	String	*|=|[!-()+-⟨⟩-~][!-~]*	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template Length
10	SEQ	String	*|[A-Za-z=.]+	segment Sequence
11	QUAL	String	[!-\]+	Phred Qual. of the Seq.

These are followed by optional fields some of which are standard and some others are aligner specific.

More detailed information on Sam format specification can be found at:
http://samtools.github.io/hts-specs/SAMv1.pdf

How do we convert sam files to bam files and bam files to sam files?

Use samtools.
Samtools is a software used to view and convert sam / bam files.

```
$ samtools command options
```

Don't have samtools?

Installing Software

What if we need a software that we dont't have in the mghpc?

You can only install software **LOCALLY!**

Installing Software

What if we need a software that we dont't have in the mghpc?

You can only install software **LOCALLY!**

There may be an easier way out!

the module system

The Module System in MGHPC

Many useful bioinformatics tools are already installed!
You need to *activate* the ones you need for your account.

To see the available modules:

```
$ module avail
```

To load a module, say samtools version 0.0.19:

```
$ module load samtools/0.0.19
```

If you can't find the software among the available modules, you can make a request to the admins via
ghpcc@list.umassmed.edu

Converting Sam to Bam

```
$ samtools view -Sb sample.sam > sample.bam
```

By default, the input is in bam format. Using `-S`, we tell that the input is in sam format.
By default, the output is in sam format, by `-b`, we tell that the output is in bam format.

Converting Bam to Sam

```
$ samtools view -h sample.bam > output.sam
```

We need to provide the parameter -h to have the headers in the sam file.

More on grep

Let's find all reads in a fastq file that **end** with **AAA**.
For this, we can use `grep -E` with *regular expressions*.

More on grep

Let's find all reads in a fastq file that **end** with **AAA**.
For this, we can use `grep -E` with *regular expressions*.

```
$ grep -E "AAA$" control.rep1.1.fq --color=always
```

More on grep

Let’s find all reads in a fastq file that **end** with **AAA**.
For this, we can use `grep -E` with *regular expressions*.

```
$ grep -E "AAA$" control.rep1.1.fq --color=always
```

Let’s find all reads in a fastq file that **begin** with *AAA*.

More on grep

Let’s find all reads in a fastq file that **end** with **AAA**.
For this, we can use `grep -E` with *regular expressions*.

```
$ grep -E "AAA$" control.rep1.1.fq --color=always
```

Let’s find all reads in a fastq file that **begin** with *AAA*.

```
$ grep -E "^AAA" control.rep1.1.fq --color=always
```

The character $ matches the end of a line and ^ matches the beginning of a line.

Exercise

Find all sequences in a fastq file that does **NOT** begin with a CA and that **does** end with an A**.**

Exercise

Find all sequences in a fastq file that does **NOT** begin with a CA and that **does** end with an A.

```
$ awk '{if(NR % 4 == 2){print($0)}}' file.fq | grep -v -E "^CA"\
 | grep -E "A$"
```

Exercise

Find all sequences in a fastq file that does **NOT** begin with a CA and that **does** end with an A.

```
$ awk '{if(NR % 4 == 2){print($0)}}' file.fq | grep -v -E "^CA"\
 | grep -E "A$"
```

Try doing this using awk only.