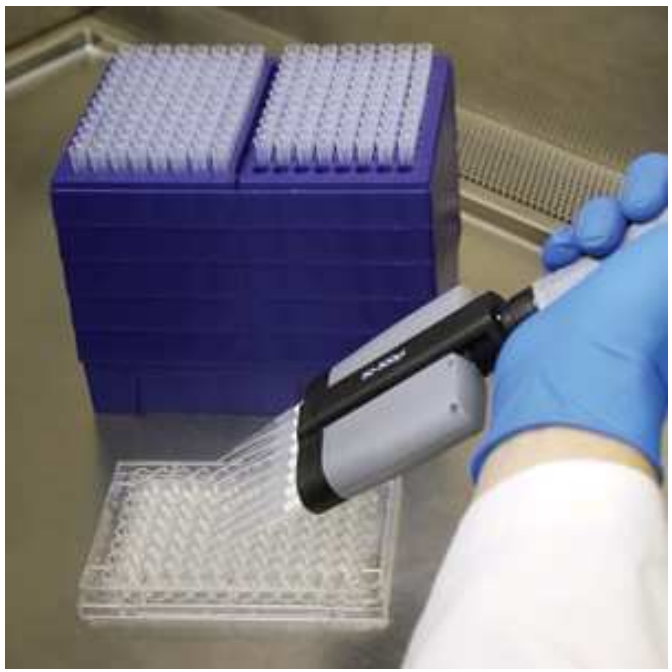
UMass High Performance Computing Center

University of Massachusetts Medical School

February, 2019

Challenges of Genomic Data

It is getting easier and cheaper to produce bigger genomic data every day. Today it is not unusual to have 100 samples getting sequenced for a research project. Say, we have 100 samples sequenced and each sample gave us about 50 million reads. It may easily take half a day to process **just one** library on a desktop computer.



Why Cluster?

Massive data coming from Deep Sequencing needs to be

- stored
- (parallel) processed

It is not feasible to process this kind of data even using a high-end computer.

MGHPCC

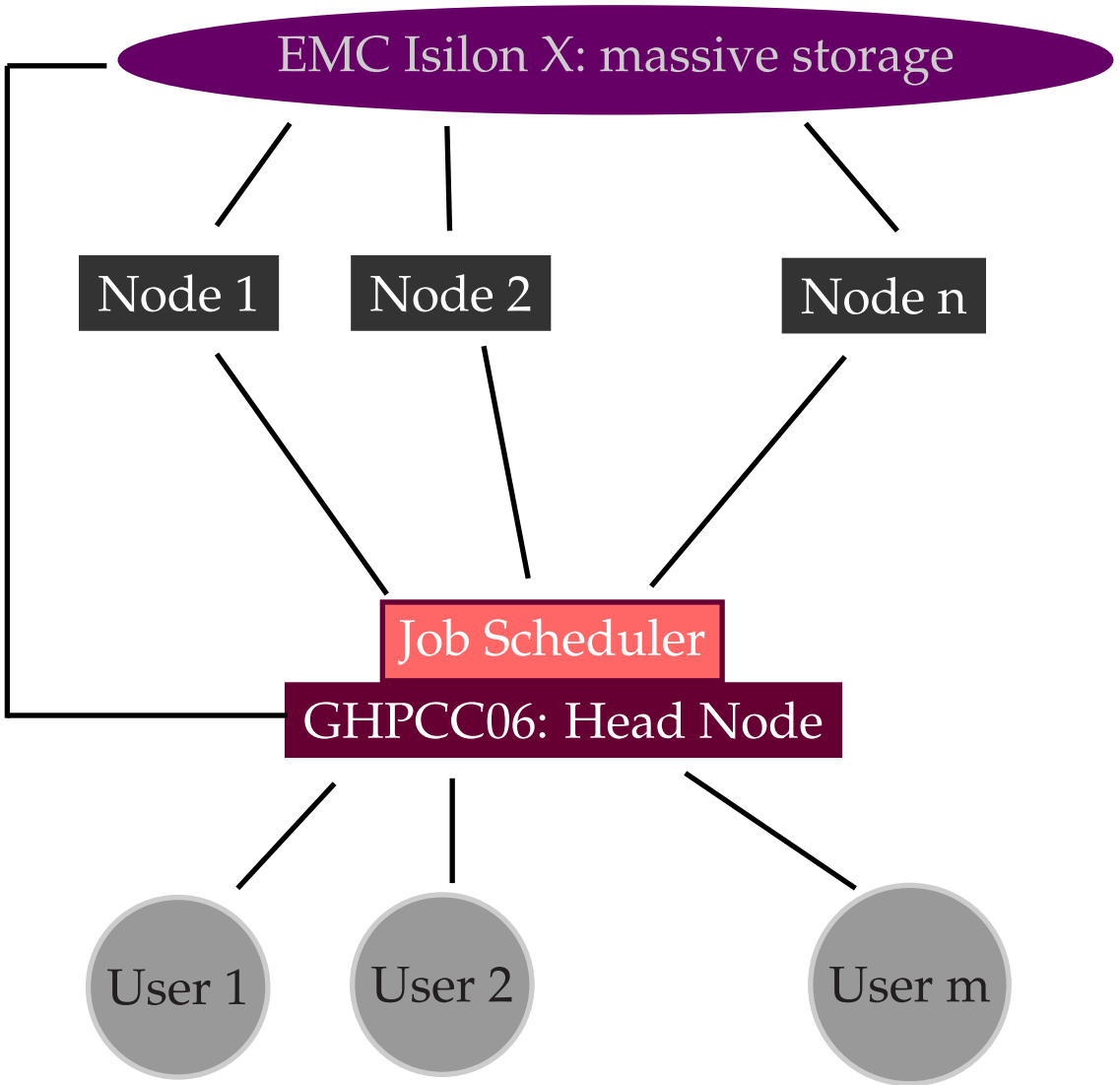
University of Massachusetts Green High Performance Computing Cluster

$$\text{HPCC} \equiv \text{GHPCC} \equiv \text{MGHPCC} \equiv \text{the Cluster}$$

HPC	:	High performance computing
Cluster	:	a number of similar things that occur together
Computer Cluster	:	A set of computers connected together that work as a single unit

MGHPCC has over 10K+ cores available and 400+ TB of high performance storage. It is located in Holyoke MA and provides computing services to the five campuses of UMass.

Overview


EMC Isilon X: massive storage
Node 1
Node 2
Node n
Job Scheduler
GHPCC06: Head Node
User 1
User 2
User m

Storage Organization

Though there are many file systems mounted on the head node, there are three file systems that are important for us.

Type	Root Directory	Contents	Quota
Home Space	/home/user_name = ~	Small Files, executables, scripts	50 GB
Project Space	/project/umw_PI_name	Big files being actively processed	Varies
Nearline Space	/nl/umw_PI_name	Big files for long term storage	Varies

Reaching the Nodes

We do **NOT** use the head node (ghpcc06) to process big data.
We use the cluster nodes to process it.

How do we reach the nodes?

Reaching the Nodes

We do **NOT** use the head node (ghpcc06) to process big data. We use the cluster nodes to process it.

How do we reach the nodes?

We submit our commands as jobs to a *job scheduler* and the job scheduler finds an available node for us having the sufficient resources (cores & memory.)

Job Scheduler

Job Scheduler is a software that manages the resources of a cluster system. It manages the program execution in the nodes. It puts the *jobs* in a (priority) queue and executes them on a node when the requested resources become available.

Job Scheduler

Job Scheduler is a software that manages the resources of a cluster system. It manages the program execution in the nodes. It puts the *jobs* in a (priority) queue and executes them on a node when the requested resources become available.

There are many Job Schedulers available. In MGHPCC,

IBM LSF (**L**oad **S**haring **F**acility)

is used.

Say we have 20 libraries of RNASeq data. We want to align using tophat.

```
tophat ...  library_1.fastq
```

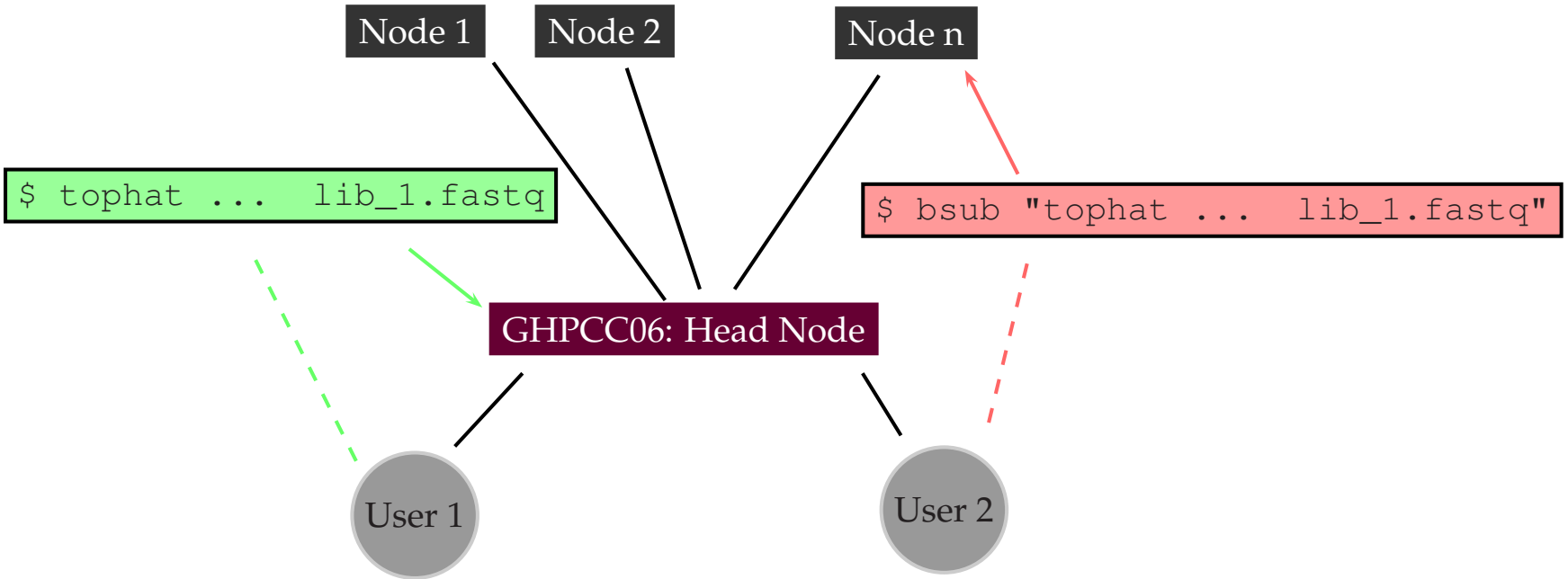
We submit this job to the job scheduler rather than running it on the head node.

Say we have 20 libraries of RNASeq data. We want to align using tophat.

```
tophat ...  library_1.fastq
```

We submit this job to the job scheduler rather than running it on the head node.

Submitting a job vs running on the head node



Our First Job Submission

We use the command **bsub** to submit jobs to the cluster.
Let's submit a dummy job.

```
$ bsub "echo Hello LSF > ~/firstjob.txt"
```

Specifying Resources

After running

```
$ bsub "echo Hello LSF > ~/firstjob.txt"
```

we got the following warning message

```
Job does not list memory required, please specify memory
...
Job runtime not indicated, please specify job runtime
...
Job <12345> is submitted to default queue <long>
```

Why did the job scheduler warn us?

Specifying Resources

Besides other things, each job requires

1. Core(s) processing units
2. Memory

to execute.

The maximum amount of time needed to complete the job must be provided.

There are different queues for different purposes, so the queue should also be specified as well.

Specifying Resources

Cores	:	Number of processing units to be assigned for the job. Some programs can take advantage of multicores .Default value is 1.
Memory Limit	:	The submitted job is not allowed to use more than the specified memory. Default value is 1 GB
Time Limit	:	The submitted job must finish in the given time limit. Default value is 60 minutes.
Queue	:	There are several queues for different purposes. Default queue is the long queue.

Queues

Let's see the queues available in the cluster.

```
$ bqueues
```

We will be using the queues `interactive` , `short` and `long`.

interactive	:	used for bash access to the nodes
short	:	used for jobs that take less than 4 hours
long	:	(default queue) used for jobs that take more than 4 hours.

Specifying Resources

Hence we must provide

1. The number of cores
2. The amount of memory
3. Time limit
4. Queue

when submitting a job unless we want to use the system default values.

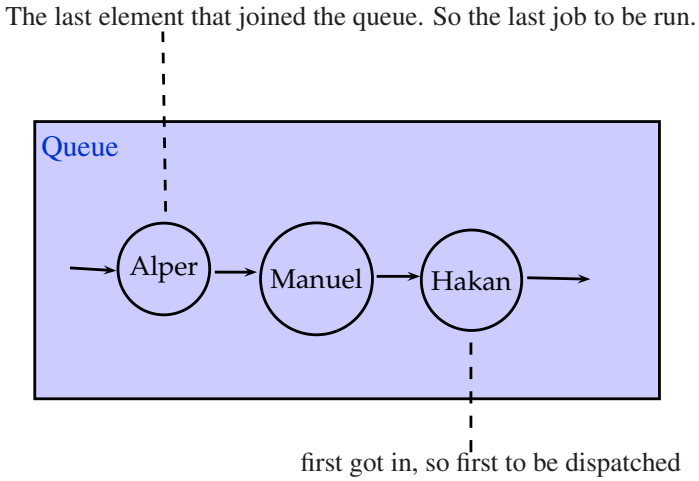
In a system like MGHPCC, where there are over 10K cores and tens of thousands of jobs and hundreds of users, specifying the right parameters can make a big difference!

Job Scheduling

Let's try to understand how a job scheduler works on a hypothetical example. The IBM LSF system works differently but using similar principles.
Suppose, for a moment, that when we submit a job, the system puts our job in a queue.
A queue is a data type that implements **F**irst **I**n **F**irst **O**ut (FIFO) structure.

Job Scheduling

Say first, Hakan then, Manuel and, lastly, Alper submit a job. Then, the queue will look like



What if Hakan's job needs 10 cores and 5 TB of memory in total and 8 hours to run whereas Alper's job only needs one core 1 GB of memory and 20 minutes to run. Also, Alper didn't use the cluster a lot recently but Hakan has been using it very heavily for weeks.

This wouldn't be a nice distribution of resources. A better approach would be prioritizing jobs, and therefore using a priority queue.

Job Scheduling

In a priority queue, each element has a priority score. The first element to be removed from the queue is the one having the highest priority.

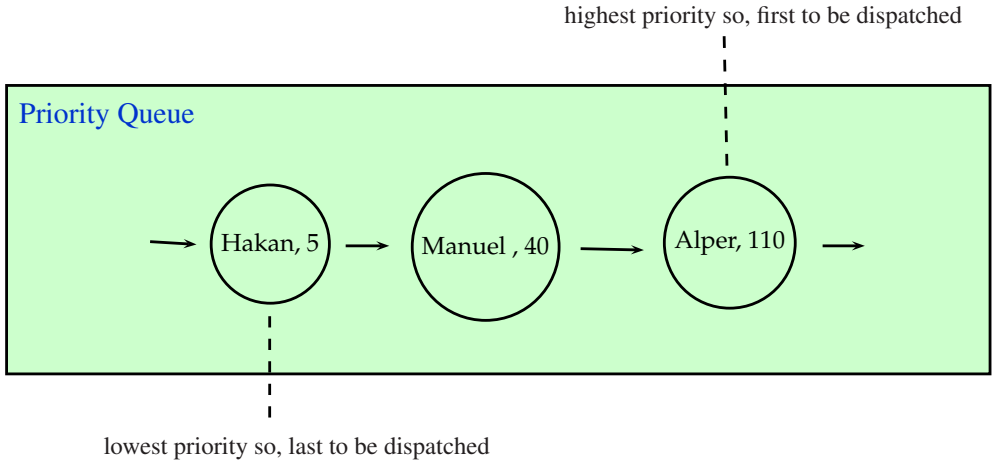
Hakan: I need 10 cores, 5TB of memory, 8 hours of time.
System: A lot of resources requested and heavy previous usage, so the priority score is 5.

Manuel: I need 2 cores, 8 GB of memory and one hour time.
System: A medium amount of resources requested, light previous usage, so the priority is 40.

Alper: I need one core, 1 GB of memory and 20 minutes.
System: Very little amount of resources requested, light previous usage, so the priority is 110.

Job Scheduling

So, in a priority queue, we would have



This is a better and fairer sharing of resources. **Therefore** it is important to ask for the right amount of resources in your job submissions.

If you ask more than you need, then it will take longer to start your job.
If you ask less than you need, then your job is going to be killed.
It is a good idea to ask a little more than you actually need.

A more Sophisticated Job Submission

Let's submit another job and specify the resources this time.
To set

1. We explicitly state that we request a single core, `-n 1`
2. The memory limit to 1024 MB, we add `-R rusage[mem=1024]`
3. Time limit to 20 minutes, we add `-W 20`
4. Queue to short, we add `-q short`

```
$ bsub -n 1 -R rusage[mem=1024] -W 20 -q short "sleep 300"
```

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

We need 4 cores as we'l run our process in 4 threads, so we need `-n 4`.
2 GB = 2048 MB, so we need the parameter `-R rusage[mem=2048]`.
We can **estimate** the running time to be 20 / 4 = 5 hours = 300 mins. So, let's ask for 330 mins to be on the safer side.

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

We need 4 cores as we'l run our process in 4 threads, so we need `-n 4`.
2 GB = 2048 MB, so we need the parameter `-R rusage[mem=2048]`.
We can **estimate** the running time to be 20 / 4 = 5 hours = 300 mins. So, let's ask for 330 mins to be on the safer side.

```
$ bsub -R span[hosts=1] -n 4 -R rusage[mem=2048] -W 330 -q long "~/bin/myscript.pl -p 4"
```

Monitoring Jobs

We will be running jobs that take tens of minutes or even hours.
How do we check the status of our active jobs?

```
$ bjobs
```

Monitoring Jobs

We will be running jobs that take tens of minutes or even hours.
How do we check the status of our active jobs?

```
$ bjobs
```

Let's create some dummy jobs and monitor them.
We run

```
$ bsub "sleep 300"
```

several times. Then

```
$ bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
1499929	ho86w	RUN	long	ghpcc06	c09b01	sleep 300	Oct 6 01:23
1499930	ho86w	RUN	long	ghpcc06	c09b01	sleep 300	Oct 6 01:23
1499931	ho86w	RUN	long	ghpcc06	c09b01	sleep 300	Oct 6 01:23

Monitoring Jobs

We can give a name to a job to make job tracking easier.
We specify the name in the `-J` parameter.

```
$ bsub -J lib_1 "sleep 300"
```

```
$ bsub -J lib_2 "sleep 300"
```

```
$ bsub -J lib_3 "sleep 300"
```

Canceling Jobs

```
$ bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
1499929	ho86w	RUN	long	ghpcc06	c09b01	sleep 300	Oct 6 01:23

We give the `JOBID` to `bkill` to cancel the job we want.

```
$ bkill 1499929
```

Creating Logs

It can be helpful to have the output and specifications of the jobs in separate files.
Two log files can be created: the standard error output and the standard output of the command run.
The standard output file is created using the `-o` parameter and the standard error output is be created using the `-e` parameter.

```
$ bsub -o output.txt -e error.txt "echo foo 1>&2; echo bar"
```

Using Bash on the Computing Nodes

Can I get a computing node (other than the head node) for myself temporarily?

Using Bash on the Computing Nodes

Can I get a computing node (other than the head node) for myself temporarily?

Yes. The interactive queue can be used for that.

```
$ bsub -q interactive -W 120 -Is bash
```

Determining Resources

How do we determine the queue, time limit, memory and number of cores?

Queue: Use the interactive queue for bash access. The time limit can be 8 hours maximum. If your job requires less than 4 hours, use the **short** queue, if it requires more than 4 hours, you need to submit it to the **long** queue.

Time Limit: This depends on the software and the size of data you are using. If you have a time estimate, request a bit more than that.

Determining Resources

Memory: Depends on the application. Some alignment jobs may require up to 32 GB whereas a simple gzip can be done with 1 GB of memory.

Number of Cores: Depends on the application. Use 1 if you are unsure. Some alignment software can take advantage of multicore systems. Check the documentation of the software you are using.

Advised Practice

- **Do not use the head node for *big jobs!***
  Do not run programs on the head node that will take longer than 5 minutes or that will require gigabytes of memory. Instead submit such commands as jobs. You can also use the interactive queue for command line access to the nodes. **This is mandatory!**
- Remember that MGHPCC is a shared resource among the five campuses of UMass!
- Keep in mind that you are probably sharing the same nearline and project space quota with your lab members. Be considerate when using it!
- Keep your password secure.
- Backup your data.

Most Important MGHPCC Policy

Do not use the head node for *big jobs!*
Do not run programs on the head node that will take longer than 5 minutes or that will require gigabytes of memory. Instead submit such commands as jobs. You can also use the interactive queue for command line access to the nodes. **This is mandatory!**

On the head node (ghpcc06), using alignment software, samtools, bedtools and etc, R, Perl , Python Scripts and etc. for deep sequencing data is a **very bad** idea!
You are likely to get a warning and / or termination of your jobs if you do so.

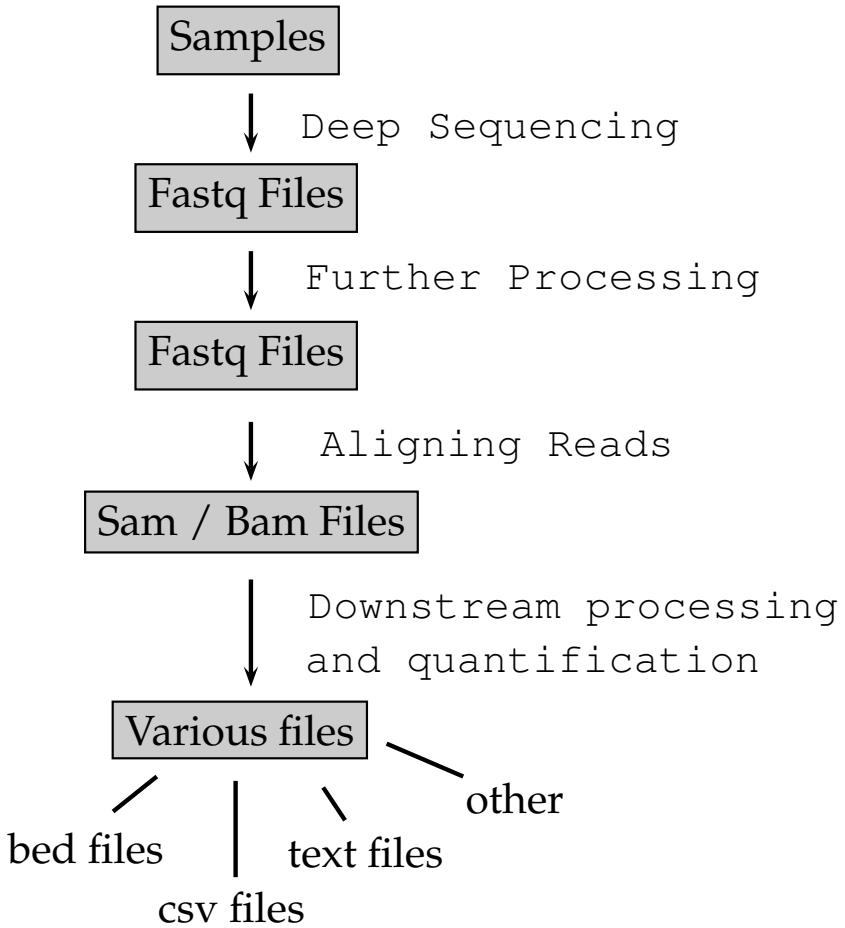
For questions: hpcc-support@umassmed.edu

Advised Practice

- Keep your files organized
- Do not put genomic data in your home folder. Process data in the project space and use nearline for long term storage
- Delete unnecessary intermediate files
- **Be considerate when submitting jobs and using disk space. The cluster is a shared resource.**
- Do not process big data in the head node. Always submit jobs instead.

For more detailed information, see
http://wiki.umassrc.org/

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:

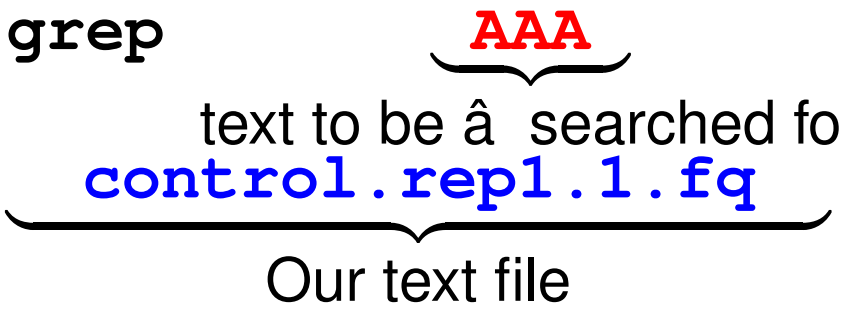
- Viewing: less
- Searching: grep
- Table Processing: awk
- Editors: nano, vi, sed

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.



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 $\rightarrow$ ˆ

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

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

'{print($2)}': awk statement

sample.sam: 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)

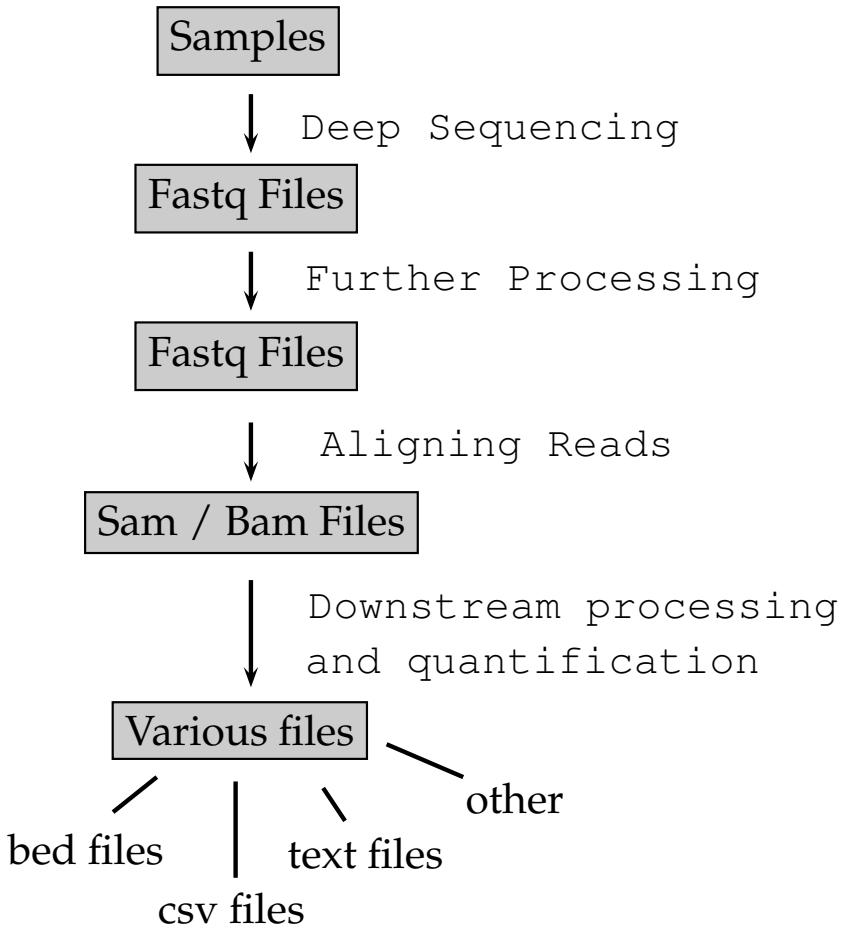
Let's find the number of chromosomes in the mm10.fa file. Each chromosome entry begins with ">", we get them by

```
$ 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

Fastq File $\xrightarrow{\text{Aligner}}$ 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	* | [!-()+-$\langle\rangle$ -~][!-~]*	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	*|=|[!-()+-$\langle\rangle$-~][!-~]*	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.