Pre-processing raw sequences

Before you start the alignment and analysis processes, it us useful to perform some initial quality checks on your raw data. You may also need to pre-process the sequences to trim them or remove adapters. Here we will assume you have paired-end data from one of GSAF's Illumina sequencers.

Reservations

Use our summer school *reservation* (**CoreNGS-Wed**) when submitting batch jobs to get higher priority on the **ls6** normal queue ***today***:

```
sbatch --reservation=CoreNGS-Wed <batch_file>.slurm
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Wed
```


FASTQ Quality Assurance tools

The first order of business after receiving sequencing data should be to check your data quality. This often-overlooked step helps guide the manner in which you process the data, and can prevent many headaches.

FastQC

FastQC is a tool that produces a quality analysis report on **FASTQ** files.

Useful links:

- **FastQC** report for a Good Illumina dataset
- **FastQC** report for a Bad Illumina dataset
- **Online documentation for each FastQC report**

First and foremost, ***the FastQC "Summary" should generally be ignored***. Its "grading scale" (**green** - good, **yellow** - warning, **red** - failed) incorporates assumptions for a particular kind of experiment, and is not applicable to most real-world data. Instead, look through the individual reports and evaluate them according to your experiment type.

The **FastQC** reports I find most useful, and why:

1. Should I trim low quality bases?
   - consult the **Per base sequence quality** report
     - based on *all* sequences
2. Do I need to remove adapter sequences?
   - consult the **Adapter Content** report
3. Do I have other contamination?
   - consult the **Overrepresented Sequences** report
     - based on the 1st 100,000 sequences, trimmed to 75bp
4. How complex is my library?
   - consult the **Sequence Duplication Levels** report
   - but remember that different experiment types are expected to have vastly different duplication profiles

For many of its reports, **FastQC** analyzes only the first ~100,000 sequences in order to keep processing and memory requirements down. Consult the **Online documentation for each FastQC report** for full details.

Running FastQC

Make sure you're in an **idev** session. If you're in an **idev** session, the **hostname** command will display a name like **c455-021.ls6.tacc.utexas.edu**. But if you're on a login node the **hostname** will be something like **login2.ls6.tacc.utexas.edu**.

If you're on a login node, start an **idev** session like this:

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Wed
```

FastQC is available as part of **BioContainers** on **ls6**. To make it available:

```
# Load the main BioContainers module then load the fastqc module
module load biocontainers  # make take a while
module load fastqc
```

It has a number of options (see **fastqc --help | more**) but can be run very simply with just a **FASTQ** file as its argument.

```
# Setup (if needed)
export CORENGS=/work/projects/BioITeam/projects/courses/Core_NGS_Tools
mkdir -p $SCRATCH/core_ngs/fastq_prep
cd $SCRATCH/core_ngs/fastq_prep
cp $CORENGS/misc/small.fq .
```

Running fastqc on a FASTQ file

```
# make sure you're in your $SCRATCH/core_ngs/fastq_prep directory
cds
cd core_ngs/fastq_prep
fastqc small.fq
```

Exercise: What did FastQC create?

ls -l shows two new items.

```
-rw-rw-r-- 1 abattenh G-823651 676531 Jun  9 20:53 small_fastqc.html
-rw-rw-r-- 1 abattenh G-823651 464403 Jun  9 20:53 small_fastqc.zip
```

- `small_fastqc.html` is the FastQC report, in HTML format.
- `small_fastqc.zip` is a zipped (compressed) directory of FastQC output files.

Let's unzip the **.zip** file and see what's in it.

```
unzip small_fastqc.zip
```

What was created?

ls -l shows one new item, the **small_fastqc** directory (note the "**d**" in "**drwxrwxr-x**")

```
drwxrwxr-x 4 abattenh G-823651      6 Jun 10  2022
```

ls -l small_fastqc shows the directory contents:

```
drwxrwxr-x 2 abattenh G-823651      4 Jun 10  2022 Icons
drwxrwxr-x 2 abattenh G-823651      9 Jun 10  2022 Images
-rw-rw-r-- 1 abattenh G-823651  77464 Jun 10  2022 fastqc.fo
-rw-rw-r-- 1 abattenh G-823651  25602 Jun 10  2022 fastqc_data.txt
-rw-rw-r-- 1 abattenh G-823651 676531 Jun 10  2022 fastqc_report.html
-rw-rw-r-- 1 abattenh G-823651    419 Jun 10  2022 summary.txt
```

Looking at FastQC output

You can't run a web browser directly from your "dumb terminal" command line environment. The **FastQC** results have to be placed where a web browser can access them. One way to do this is to copy the results back to your laptop, for example by using **scp *from your computer*** (read more at Copying files from TACC to your laptop).

For convenience, we put an example **FastQC** report at this URL:
https://web.corral.tacc.utexas.edu/BioinformaticsResource/CoreNGS/yeast_stuff/Sample_Yeast_L005_R1.cat_fastqc/fastqc_report.html

Exercise: Based on this FastQC output, should we trim this data?

The **Per base sequence quality** report does not look good. The data should probably be trimmed (to 40 or 50 bp) before alignment.

Newer versions of FastQC have slightly different report formats. See this example:
https://web.corral.tacc.utexas.edu/BioinformaticsResource/CoreNGS/reports/wcaar_mqc_report.html

Using MultiQC to consolidate multiple QC reports

FastQC reports are all well and good, but what if you have dozens of samples? It quickly becomes tedious to have to look through all the separate **FastQC** reports, including separate R1 and R2 reports for paired end datasets.

The **MultiQC** tool helps address this issue. Once **FastQC** reports have been generated, it can scan them and create a consolidated report from all the individual reports.

Whats even cooler, is that **MultiQC** can also consolidate reports from other bioinformatics tools (e.g. **bowtie2** aligner statistics, **samtools** statistics, **cutadapt**, **Picard**, and may more). And if your favorite tool is not known by **MultiQC**, you can configure custom reports fairly easily. For more information, see this recent Byte Club tutorial on Using MultiQC.

Here we're just going to create a **MultiQC** report for two paired-end ATAC-seq datasets – 4 FASTQ files total. First stage the data:

```
mkdir -p $SCRATCH/core_ngs/multiqc/fqc.atacseq
cd $SCRATCH/core_ngs/multiqc/fqc.atacseq
cp $CORENGS/multiqc/fqc.atacseq/*.zip .
```

You should see these 4 files in your **$SCRATCH/core_ngs/multiqc/fqc.atacseq** directory:

```
50knuclei_S56_L007_R1_001_fastqc.zip  5knuclei_S77_L008_R1_001_fastqc.zip
50knuclei_S56_L007_R2_001_fastqc.zip  5knuclei_S77_L008_R2_001_fastqc.zip
```

Now make the **BioContainers MultiQC** accessible in your environment.

Make sure you're in an **idev** session. If you're in an **idev** session, the **hostname** command will display a name like **c455-020.ls6.tacc.utexas.edu**. But if you're on a login node the **hostname** will be something like **login1.ls6.tacc.utexas.edu**.

If you're on a login node, start an **idev** session like this:

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGSday3
```

```
# Load the main BioContainers module if you have not already
module load biocontainers  # may take a while

# Load the multiqc module and ask for its usage information
module load multiqc
multiqc --help | more
```

```
export CORENGS=/work/projects/BioITeam/projects/courses/Core_NGS_Tools
mkdir -p $SCRATCH/core_ngs/multiqc/fqc.atacseq
cd $SCRATCH/core_ngs/multiqc/fqc.atacseq
cp $CORENGS/multiqc/fqc.atacseq/*.zip .
```

Even though **multiqc** has many options, it is quite easy to create a basic report by just pointing it to the directory where individual reports are located:

```
cd $SCRATCH/core_ngs/multiqc
multiqc fqc.atacseq
```

Exercise: How many reports did multiqc find?

Based on its execution output, it found 4 reports

```
[WARNING]         multiqc : MultiQC Version v1.12 now available!
[INFO   ]         multiqc : This is MultiQC v1.7
[INFO   ]         multiqc : Template    : default
[INFO   ]         multiqc : Searching 'fqc.atacseq/'
[INFO   ]          fastqc : Found 4 reports
[INFO   ]         multiqc : Compressing plot data
[INFO   ]         multiqc : Report      : multiqc_report.html
[INFO   ]         multiqc : Data        : multiqc_data
[INFO   ]         multiqc : MultiQC complete
```

Exercise: What was created by running multiqc?

One file was created (**multiqc_report.html**) and one directory (**multiqc_data**).

You can see the resulting **MultiQC** report here: https://web.corral.tacc.utexas.edu/BioinformaticsResource/CoreNGS/reports/atacseq/multiqc_report.html.

And an example of a **MultiQC** report that includes both standard and custom plots is this is the Tag-Seq post-processing **MultiQC** report produced by the Bioinformatics Consulting Group: https://web.corral.tacc.utexas.edu/BioinformaticsResource/CoreNGS/reports/mqc_tagseq_trim_JA21030_SA21045_mouse.html

Trimming sequences

There are two main reasons you may want to trim your sequences:

- As a quick way to remove 3' adapter contamination, when extra bases provide little additional information
  - For example, 75+ bp ChIP-seq reads – 50 bases are more than enough for a good mapping, and trimming to 50 is easier than adapter removal, especially for paired end data.
  - You would not choose this approach for RNA-seq data, where 3' bases may map to a different exon, and that is valuable information.
    - Instead you would specifically remove adapter sequences.
- Low quality base reads from the sequencer can affect some programs
  - This is an issue with sequencing for genome or transcriptome assembly.
  - Aligners such as **bwa** and **bowtie2** seem to do fine with a few low quality bases, soft clipping them if necessary.

There are a number of open source tools that can trim off 3' bases and produce a **FASTQ** file of the trimmed reads to use as input to the alignment program.

FASTX Toolkit

The **FASTX Toolkit** provides a set of command line tools for manipulating both **FASTA** and **FASTQ** files. The available modules are described on their website. They include a fast **fastx_trimmer** utility for trimming **FASTQ** sequences (and quality score strings) before alignment.

Make sure you're in an **idev** session. If you're in an **idev** session, the **hostname** command will display a name like **c455-021.ls6.tacc.utexas.edu**. But if you're on a login node the **hostname** will be something like **login3.ls6.tacc.utexas.edu**.

If you're on a login node, start an **idev** session like this:

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Wed
```

FASTX Toolkit is available as a **BioContainers** module.

```
module load biocontainers  # takes a while
module spider fastx
module load fastxtools
```

Here's an example of how to run **fastx_trimmer** to trim all input sequences down to 50 bases.

Where does **fastx_trimmer** read its input from? And where does it write its output? Ask the program for its usage.

```
# will fastx_trimmer give us usage information?
fastx_trimmer --help

# no, it wants you to use the -h option to ask for help:
fastx_trimmer -h
```

The **usage**: its help information

```
fastx_trimmer [-h] [-f N] [-l N] [-t N] [-m MINLEN] [-z] [-v] [-i INFILE] [-o OUTFILE]
```

Because the **[-i INFILE] [-o OUTFILE]** options are shown in **brackets [ ]**, reading from a file and writing to a file are optional. That means that by default the program reads its input data from ***standard input*** and writes trimmed sequences to ***standard output***:

Set up directory for working with FASTQs

```
export CORENGS=/work/projects/BioITeam/projects/courses/Core_NGS_Tools

# Create a $SCRATCH area to work on data for this course,
# with a sub-direct[1ory for pre-processing raw fastq files
mkdir -p $SCRATCH/core_ngs/fastq_prep

# Make a symbolic links to the original yeast data:
cd $SCRATCH/core_ngs/fastq_prep
ln -s -f $CORENGS/yeast_stuff/Sample_Yeast_L005_R1.cat.fastq.gz
ln -s -f $CORENGS/yeast_stuff/Sample_Yeast_L005_R2.cat.fastq.gz
```

Trimming FASTQ sequences to 50 bases with fastx_trimmer

```
# make sure you're in your $SCRATCH/core_ngs/fastq_prep directory
cd $SCRATCH/core_ngs/fastq_prep
zcat Sample_Yeast_L005_R1.cat.fastq.gz | fastx_trimmer -l 50 -Q 33 > trim50_R1.fq
```

- The **-l 50** option says that base 50 should be the **l**ast base (i.e., trim down to 50 bases)
- The **-Q 33** option specifies how base **Q**ualities on the 4th line of each **FASTQ** entry are encoded.
  - The **FASTX Toolkit** is an older program written in the time when Illumina base qualities were encoded differently, so its default does not work for modern **FASTQ** files.
  - These days Illumina base qualities follow the Sanger **FASTQ** standard (Phred score + 33 to make an ASCII character).

Exercise: compressing fastx_trimmer output

How would you tell **fastx_trimmer** to compress (**gzip**) its output file?

Type **fastx_trimmer -h** (**h**elp) to see program documentation
You could supply the **-z** option like this:

```
zcat Sample_Yeast_L005_R1.cat.fastq.gz | fastx_trimmer -l 50 -Q 33 -z > trim50_R1.fq.gz

# or, using the -o option:
zcat Sample_Yeast_L005_R1.cat.fastq.gz | fastx_trimmer -l 50 -Q 33 -z -o trim50_R1.fq.gz
```

Or you could **gzip** the output yourself.

```
zcat Sample_Yeast_L005_R1.cat.fastq.gz | fastx_trimmer -l 50 -Q 33 | gzip > trim50_R1.fq.gz
```

See the 3x+ difference in file sizes when the output is compressed with **ls -lh trim***

Exercise: other fastx toolkit programs

What other **FASTQ** manipulation programs are part of the **FASTX Toolkit**?

Type **fastx_** then tab twice (completion) to see their names.

The **FASTX Toolkit** also has programs that work on **FASTA** files. To see them, type **fasta_** then tab twice (completion) to see their names.

Adapter trimming with cutadapt

Data from RNA-seq or other library prep methods that result in short fragments can cause problems with moderately long (50-100bp) reads, since the 3' end of sequences can be read into (or even through) to the 3' adapter at different read offsets . This ***3' adapter contamination*** can cause the "real" insert sequence not to align because the adapter sequence does not correspond to the bases at the 3' end of the reference genome sequence.

Unlike general fixed-length trimming (e.g. trimming 100 bp sequences to 50 bp), ***specific adapter trimming*** removes differing numbers of 3' bases depending on where the adapter sequence is found.

You must tell any adapter trimming program what your R1 and R2 adapters look like.

The GSAF website describes the flavors of Illumina adapter and barcode sequences in more detail: **Illumina - all flavors (USE with Caution, this is outdated but can be useful for a basic understanding of the adapters, the GSAF primarily only uses UDI's for all projects).**

The **cutadapt** program, available in **BioContainers**, is an excellent tool for removing adapter contamination.

Make sure you're in an **idev** session. If you're in an **idev** session, the **hostname** command will display a name like **c455-021.ls6.tacc.utexas.edu**. But if you're on a login node the **hostname** will be something like **login3.ls6.tacc.utexas.edu**.

If you're on a login node, start an **idev** session like this:

Start an idev session

```
idev -m 120 -N 1 -A OTH21164 -r CoreNGS-Wed
```

```
module load biocontainers
module spider cutadapt

module load cutadapt
cutadapt --help
```

A common application of **cutadapt** is to remove adapter contamination from RNA library sequence data. Here we'll show that for some small RNA libraries sequenced by GSAF, using their documented small RNA library adapters.

When you run **cutadapt** you give it the adapter sequence to trim, and ***the adapter sequence is different for R1 and R2 reads***. Here's what the options look like (without running it on our files yet).

cutadapt command for R1 sequences (GSAF RNA library)

```
cutadapt -m 22 -O 4 -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC <fastq_file>
```

cutadapt command for R2 sequences (GSAF RNA library)

```
cutadapt -m 22 -O 4 -a TGATCGTCGGACTGTAGAACTCTGAACGTGTAGA <fastq_file>
```

Notes:

- The **-m 22** option says to discard any sequence that is smaller than 22 bases (**m**inimum) after trimming.
  - This avoids problems trying to map very short, highly ambiguous sequences.
- the **-O 4** (**O**verlap) option says not to trim 3' adapter sequences unless at least the first 4 bases of the adapter are seen at the 3' end of the read.
  - This prevents trimming short 3' sequences that just happen by chance to match the first few adapter sequence bases.

Figuring out which adapter sequence to use when can be tricky. Your sequencing provider can tell you what adapters they used to prep your libraries. For GSAF's adapter layout, please refer to Illumina - all flavors (USE with Caution, this is outdated but can be useful for a basic understanding of the adapters, the GSAF primarily only uses UDI's for all projects) (you may want to read all the "gory details" below later).

The top strand, 5' to 3', of a read sequence looks like this.

Illumina library read layout

```
<P5 capture> <indexRead2> <Read 1 primer> [insert] <Read 2 primer> <indexRead1> <P7 capture>
```

The **-a** argument to **cutadapt** is documented as the "sequence of adapter that was ligated to the 3' end". So we care about the <Read 2 primer> for R1 reads, and the <Read 1 primer> for R2 reads.

The "contaminent" for adapter trimming will be the <Read 2 primer> for R1 reads. There is only one Read 2 primer:

Read 2 primer, 5' to 3', used as R1 sequence adapter

```
AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC
```

The "contaminent" for adapter trimming will be the <Read 1 primer> for R2 reads. However, there are three different Read 1 primers, depending on library construction:

Read 1 primer depends on library construction

```
TCTACACGTTCAGAGTTCTACAGTCCGACGATCA    # small RNA sequencing primer site
CAGGTTCAGAGTTCTACAGTCCGACGATCA        # "other"
TCTACACTCTTTCCCTACACGACGCTCTTCCGATCT  # TruSeq Read 1 primer site. This is the RC of the R2 adapter
```

Since R2 reads are the reverse complement of R1 reads, the R2 adapter contaminent will be the RC of the Read 1 primer used.

For ChIP-seq libraries where reads come from both DNA strands, the TruSeq Read 1 primer is always used.
Since it is the RC of the Read 2 primer, its RC is just the Read 1 primer back.
Therefore, for ChIP-seq libraries the same **cutadapt** adapter option can be used for both R1 and R2 reads:

Cutadapt adapter sequence for ChIP-seq lib

```
cutadapt -a GATCGGAAGAGCACACGTCTGAACTCCAGTCAC
```

For RNAseq libraries in this class, we use the small RNA sequencing primer as the Read 1 primer.
The contaminent is then the RC of this, minus the 1st and last bases:

Small RNA library Read 1 primer, 5' to 3', used as R2 sequence adapter

```
TCTACACGTTCAGAGTTCTACAGTCCGACGATCA    # R1 primer - small RNA sequencing Read 1 primer site, 5' to 3'
TGATCGTCGGACTGTAGAACTCTGAACGTGTAGA    # R2 adapter contaminent (RC of R1 small RNA sequencing Read 1 primer)
```

Exercise: other cutadapt options

The **cutadapt** program has many options. Let's explore a few.

How would you tell **cutadapt** to trim trailing N's?

```
cutadapt --help | less
```

Then, in the **less** pager, type **/trim** <enter> to look for the first occurrence of the string "trim", then **n** to look for subsequent occurrences.
The relevant option is **--trim-n**

How would you control the accuracy (error rate) of **cutadapt**'s matching between the adapter sequences and the **FASTQ** sequences?

Use the **less** pager to search for terms like "error" or "accuracy".

```
cutadapt --help | less
```

Then, in the **less** pager, type **/error** <enter> to look for the first occurrence of the string "error", then **n** to look for subsequent occurrences.
The relevant option is **-e <floating point error rate>** or **--error-rate=<floating point error rate>**:

```
-e ERROR_RATE, --error-rate=ERROR_RATE
     Maximum allowed error rate (no. of errors divided by
     the length of the matching region) (default: 0.1)
```

Suppose you are processing 100 bp reads with 30 bp adapters. By default, how many mismatches between the adapter and a sequence will be tolerated?

cutadapt's default error rate is 0.1 (10%)
Up to three mismatches will be tolerated when the whole 30 bp adapter is found (10% of 30).

If only 20 of the 30 adapter bases are found, up to two mismatches will be tolerated (10% of 20).

How would you require a more stringent matching (i.e., allowing fewer mismatches)?

Providing **--error-rate=0.05** (or **-e 0.05**) as an option, for example, would specify a 5% error rate, or no more than 1 mismatching base in 20.

cutadapt example

Let's run **cutadapt** on some real human miRNA (micro-RNA) data.

First, stage the data we want to use. This data is from a small RNA library where the expected insert size is around 15-25 bp.

Setup for cutadapt on miRNA FASTQ

```
mkdir -p $SCRATCH/core_ngs/fastq_prep
cd $SCRATCH/core_ngs/fastq_prep
cp $CORENGS/human_stuff/Sample_H54_miRNA_L004_R1.cat.fastq.gz .
cp $CORENGS/human_stuff/Sample_H54_miRNA_L005_R1.cat.fastq.gz .
```

Exercise: How many reads are in these files? Is it single end or paired end data?

```
echo $(( `zcat Sample_H54_miRNA_L004_R1.cat.fastq.gz | wc -l` / 4 ))
# or
zcat Sample_H54_miRNA_L004_R1.cat.fastq.gz | wc -l | awk '{print $1 / 4}'
```

Looking at the **FASTQ** file names, we see this is two lanes of single-end reads (**L004** and **L005**).

The data from lane 4 has 2,001,337 reads, the data from lane 5 has 2,022,237 reads.

Exercise: How long are the reads?

You could just Look at the size of the actual sequence on the 2nd line of any **FASTQ** entry and count the characters....

But you're experts now! So challenge yourself.

Use a combination of **tail** and **head** to extract the 2nd line of the **.gz** file.

Then use the **wc** program, but not with the **-l** option (check **wc --help**).

```
zcat Sample_H54_miRNA_L004_R1.cat.fastq.gz | head -2 | tail -1 | wc -c
```

These are 101-base reads. **wc -c** counts the "invisible" **newline** character, so subtract 1 from the character count it returns for a line.

Here's a way to strip the trailing newline characters from the quality scores string before calling **wc -c** to count the characters. We use the **echo -n** option that tells **echo** not to include the trailing **newline** in its output. We gemerate that text using ***sub-shell evaluation*** (an alternative to backtick evaluation) of that **zcat** ... command:

```
echo -n $( zcat Sample_H54_miRNA_L004_R1.cat.fastq.gz | head -2 | tail -1 ) | wc -c
```

Adapter trimming is a rather slow process, and these are large files. So to start with we're going to create a smaller **FASTQ** file to work with.

```
# Remember, FASTQ files have 4 lines per read
zcat Sample_H54_miRNA_L004_R1.cat.fastq.gz | head -2000 > miRNA_test.fq
```

Now execute **cutadapt** like this:

Setup for cutadapt on miRNA FASTQ

```
export CORENGS=/work/projects/BioITeam/projects/courses/Core_NGS_Tools
mkdir -p $SCRATCH/core_ngs/fastq_prep
cd $SCRATCH/core_ngs/fastq_prep
cp $CORENGS/human_stuff/miRNA_test.fq .
```

Cutadapt command for R1 FASTQ

```
cutadapt -m 20 -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC miRNA_test.fq \
  2> miRNA_test.cuta.log \
  | gzip > miRNA_test.cutadapt.fq.gz
```

Notes:

- Here's one of those cases where you have to be careful about separating ***standard output*** and ***standard error***.
  - **cutadapt** writes its **FASTQ** output to ***standard output*** by default, and writes summary information to ***standard error***.
- In this command we ***first*** redirect ***standard error*** to a log file named **miRNA_test.cuta.log** using the **2>** syntax, in order to capture **cutadapt** diag nostics.
  - Then the remaining ***standard output*** is piped to **gzip**, whose output is the redirected to a new compressed **FASTQ** file.
  - (Read more about **Standard streams and redirection**)

You should see a **miRNA_test.cuta.log** log file when the command completes. How many lines does it have?

```
wc -l miRNA*log
```

Take a look at the first 15 lines.

```
head -15 miRNA_test.cuta.log
```

It will look something like this:

cutadapt log file

```
This is cutadapt 1.18 with Python 3.7.1
Command line parameters: -m 20 -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC miRNA_test.fq
Processing reads on 1 core in single-end mode ...
Finished in 0.06 s (113 us/read; 0.53 M reads/minute).

=== Summary ===

Total reads processed:                     500
Reads with adapters:                       492 (98.4%)
Reads that were too short:                  64 (12.8%)
Reads written (passing filters):           436 (87.2%)

Total basepairs processed:        50,500 bp
Total written (filtered):         10,909 bp (21.6%)
```

Notes:

- The **Total reads processed** line tells you how many sequences were in the original **FASTQ** file.
- **Reads with adapters** tells you how many of the reads you gave it had at least part of an adapter sequence that was trimmed.
  - Here adapter was found in nearly all (98.4%) of the reads. This makes sense given this is a short (15-25 bp) RNA library.
- The **Reads that were too short** line tells you how may sequences were ***filtered out*** because they were shorter than our minimum length (20) after adapter removal (these may have ben primer dimers).
  - Here ~13% of the original sequences were removed, which is reasonable.
- **Reads written (passing filters)** tells you the total number of reads that were written out by **cutadapt**

    - These are reads that were at least 20 bases long after adapters were removed

The **cutadapt --help** output describes its usage as follows:

```
cutadapt -a ADAPTER [options] [-o output.fastq] input.fastq
```

From this we see that the **input.fastq** is a required argument. Clearly, it can be a FASTQ file, and it can be compressed based on this help:

```
Compressed input and output is supported and
auto-detected from the file name (.gz, .xz, .bz2).
Use the file name '-' for standard input/output.
```

And this says that input reads can also be provided on ***standard input***, if that argument is a **hyphen** (**-**). So input data can come:

- from a ***file*** named as an argument:
    - **cutadapt -a CGTAATTCGCG -o small.trim.fq small.fq**
    - and that **input.fastq** file can be provided in one of three compression formats
- from ***standard input*** if the **input.fastq** argument is replaced with a **dash** (**-**)
    - **cat small.fq | cutadapt -a CGTAATTCGCG -o small.trim.fq -**

What about **cutadapt** output (the trimmed reads)? The **brackets** around the usage **-o** option indicate that the resulting trimmed **FASTQ** can be written to a file, but is not by default. This implies that cutadapt by default writes its results to ***standard output***. So output can go

- to a ***file***, using the **-o** option
    - **cutadapt -a CGTAATTCGCG -o small.trim.fq small.fq**
- to ***standard output*** without the **-o** option
    - **cutadapt -a CGTAATTCGCG small.fq 1> small.trim.fq**

Finally, as we've seen, **cutadapt** also writes diagnostic output. Where does it go? The usage line doesn't say anything about diagnostics explicitly. But in the **Output** section of **cutadapt --help**:

```
-o FILE, --output=FILE
     Write trimmed reads to FILE. FASTQ or FASTA format is
     chosen depending on input. The summary report is sent
     to standard output. Use '{name}' in FILE to
     demultiplex reads into multiple files. Default: write
     to standard output
```

Careful reading of this suggests that:

- When the trimmed output is sent to a file with the **-o output.fastq** option,
    - diagnostics are written to ***standard output***
        - so can be redirected to a log file with **1> small.trim.log**
    - **cutadapt -a CGTAATTCGCG -o small.trim.fq small.fq 1> small.trim.log**
- But when the **-o** option is omitted, and output goes to ***standard output***,
    - diagnostics must be written to ***standard error***
        - so can be redirected to a log file with **2> trim.log**
    - **cutadapt -a CGTAATTCGCG small.fq 1> small.trim.fq 2> small.trim.log**

paired-end data considerations

Special care must be taken when removing adapters for paired-end **FASTQ** files.

- For paired-end alignment, aligners want the R1 and R2 fastq files to be in the ***same name order*** and be the ***same length***.
- Adapter trimming can remove **FASTQ** sequences if the trimmed sequence is too short
    - But different R1 and R2 reads may be discarded
    - This leads to mis-matched R1 and R2 **FASTQ** files, which can cause problems with aligners like **bwa**
- **cutadapt** has a protocol for re-syncing the R1 and R2 when the R2 is trimmed.
    - See the **cutadapt** manual for more details (https://cutadapt.readthedocs.org/en/stable/).

running cutadapt in a batch job

Now we're going to run **cutadapt** on the larger **FASTQ** files, and also perform paired-end adapter trimming on some yeast paired-end RNA-seq data.

First stage the 4 **FASTQ** files we will work on:

Setup for cutadapt

```
mkdir -p $SCRATCH/core_ngs/cutadapt
cd $SCRATCH/core_ngs/cutadapt
cp $CORENGS/human_stuff/Sample_H54_miRNA_L004_R1.cat.fastq.gz .
cp $CORENGS/human_stuff/Sample_H54_miRNA_L005_R1.cat.fastq.gz .
cp $CORENGS/custom_tracks/Yeast_RNAseq_L002_R1.fastq.gz .
cp $CORENGS/custom_tracks/Yeast_RNAseq_L002_R2.fastq.gz .
```

Instead of running **cutadapt** on the command line, we're going to submit a job to the TACC batch system to perform ***single-end adapter trimming*** on the two lanes of miRNA data, and ***paired-end adapter trimming*** on the two yeast RNAseq **FASTQ** files.

Paired end adapter trimming is rather complicated, so instead of trying to do it all in one command line we will use one of the handy BioITeam scripts that handles all the details of paired-end read trimming, including all the environment setup.

Paired-end RNA fastq trimming script

The BioITeam has an a number of useful NGS scripts that can be executed by anyone on **ls6**. or **stampede2**. They are located in the **/work/projects/BioITeam/common/script/** directory.

For groups that participate in BRCF pods, the scripts are available in **/mnt/bioi/script** on any compute server.

The name of the script we want is **trim_adapters.sh**. Just type the full path of the script with no arguments to see its help information:

trim_adapters.sh

```
/work/projects/BioITeam/common/script/trim_adapters.sh
```

You should see something like this:

```
trim_adapters.sh 2020_04_20
Trim adapters from single- or paired-end sequences using cutadapt. Usage:

trim_adapters.sh <in_fq> <out_pfx> [ paired min_len adapter1 adapter2 ]

Required arguments:
  in_fq     For single-end alignments, path to input fastq file.
            For paired-end alignemtts, path to the the R1 fastq file
            which must contain the string 'R1' in its name. The
            corresponding 'R2' must have the same path except for 'R1'
  out_pfx   Desired prefix of output files.
Optional arguments:
  paired    0 = single end alignment (default); 1 = paired end.
  min_len   Minimum sequence length after adapter removal. Default 32.
  adapter1  3' adapter. Default GATCGGAAGAGCACACGTCTGAACTCCAGTCAC  (NEB).
            Specifiy 'illumina' for AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC
            (standard Illumina TruSeq3 indexed adapter).
  adapter2  5' adapter. Default TGATCGTCGGACTGTAGAACTCTGAACGTGTAGA (NEB).
            Specifiy 'illumina' for AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTA
            (standard Illumina TruSeq universal adapter).
Environment variables:
  show_only 1 = only show what would be done (default not set)
  keep      1 = keep intermediate file(s) (default 0, don't keep)
  cuta_args other cutadapt options (e.g. '--trim-n --max-n=0.25')

Examples:
  export cuta_args='-O 5'; trim_adapters.sh my.fastq.gz h54_b1 1 40
  trim_adapters.sh my_fastq.gz yeast_b3 1 28 Illumina Illumina
```

Based on this information, here are the 3 **cutadapt** commands we want to execute:

Adapter trimming commands file

```
/work/projects/BioITeam/common/script/trim_adapters.sh Sample_H54_miRNA_L004_R1.cat.fastq.gz H54_miRNA_L004 0 20
/work/projects/BioITeam/common/script/trim_adapters.sh Sample_H54_miRNA_L005_R1.cat.fastq.gz H54_miRNA_L005 0 20
/work/projects/BioITeam/common/script/trim_adapters.sh Yeast_RNAseq_L002_R1.fastq.gz yeast_rnaseq 1
```

Let's put these command into a **cuta.cmds** commands file. But first we need to learn a bit about **Editing files** in Linux.

Exercise: Create cuta.cmds file

Use **nano** or **emacs** to create a **cuta.cmds** file with the 3 **cutadapt** processing commands above. If you have trouble with this, you can copy a pre-made commands file:

```
cd $SCRATCH/core_ngs/cutadapt
cp $CORENGS/tacc/cuta.cmds .
```

Or use this "**cat** to **MARKER**" trick, also known as an heredoc. The **MARKER** tag can be anything; below it is **EOL**.

```
cd $SCRATCH/core_ngs/cutadapt
cat > cuta.cmds << EOL
/work/projects/BioITeam/common/script/trim_adapters.sh Sample_H54_miRNA_L004_R1.cat.fastq.gz H54_miRNA_L004 0 20
/work/projects/BioITeam/common/script/trim_adapters.sh Sample_H54_miRNA_L005_R1.cat.fastq.gz H54_miRNA_L005 0 20
/work/projects/BioITeam/common/script/trim_adapters.sh Yeast_RNAseq_L002_R1.fastq.gz yeast_rnaseq 1
EOL
```

When you're finished you should have a **cuta.cmds** file that is ***3 lines long*** (check this with **wc -l**).

Next create a batch submission script for your job and submit it to the **normal** queue with a maximum run time of 2 hours.

Since batch jobs can't be submitted from an **idev** session, ***make sure you are back on a login node*** (just **exit** the **idev** session).

Create and submit cutadapt batch script

```
cd $SCRATCH/core_ngs/cutadapt
launcher_creator.py -j cuta.cmds -n cuta -t 01:00:00 -a OTH21164 -q normal
sbatch --reservation=CoreNGS-Wed cuta.slurm
showq -u

# or, if you're not on the reservation:
launcher_creator.py -j cuta.cmds -n cuta -t 01:00:00 -a OTH21164 -q development
sbatch cuta.slurm
showq -u
```

How will you know your job is done?

Your **cuta** job will no longer be displayed in the **showq -u** output.

You can also ask the batch system to send you email when the job starts to run and completes. The **launcher_creator.py** has a **-e** option that lets you provide an email on the command line. Or you can set the **EMAIL_ADDRESS** environment variable if you want **launcher_creator.py** to always fill in this field:

```
export EMAIL_ADDRESS="abattenhouse@utexas.edu"
```

All our BioITeam scripts, if they complete without errors, will write a line to their logfile that includes the words "completed successfully!". So another way of checking that each command completed is to search for that text in the logfiles.

Here we use the powerful **grep** (**g**eneral **r**egular **e**xpression **p**rocessor) tool:

```
# to see all lines in any log file with the words 'completed successfully':
grep 'completed successfully!' *.log

# or, to simply count how many lines have the the words 'completed successfully':
grep 'completed successfully!' *.log | wc -l

# or to see only the names of files that have the words 'completed successfully'
grep -l 'completed successfully!' *.log

# to see the names of the files that do NOT have the words 'completed successfully'
# note we only look in *.cuta.log files since the pass0, pass1, pass2 log files will
# never contain those words.
grep -L 'completed successfully!' *.cuta.log
```

You should see several log files when the job is finished:

- **H54_miRNA_L004.cuta.log, H54_miRNA_L005.cuta.log, yeast_rnaseq.cuta.log**
  - these are the main execution log files, one for each **trim_adapters.sh** command
- **H54_miRNA_L004.acut.pass0.log, H54_miRNA_L005.acut.pass0.log**
  - these are **cutadapt** statistics files for the single-end adapter trimming
  - their contents will look like our small example above
- **yeast_rnaseq.acut.pass1.log, yeast_rnaseq.acut.pass2.log**
  - these are **cutadapt** statistics files from trimming the R1 and R2 adapters, respectively.

Take a look at the first part of the **yeast_rnaseq.acut.pass1.log** log file:

```
more yeast_rnaseq.acut.pass1.log
```

It will look something like this:

cutadapt pass1 log file

```
This is cutadapt 1.18 with Python 3.7.1
Command line parameters: -m 32 -a GATCGGAAGAGCACACGTCTGAACTCCAGTCAC --trim-n --paired-output yeast_rnaseq_R2.
tmp.cuta.fastq -o yeast_rnaseq_R1.tmp.cuta.fastq Yeast_RNAseq_L002_R1.fastq.gz Yeast_RNAseq_L002_R2.fastq.gz
Processing reads on 1 core in paired-end legacy mode ...
WARNING: Legacy mode is enabled. Read modification and filtering options
*ignore* the second read. To switch to regular paired-end mode,
provide the --pair-filter=any option or use any of the
-A/-B/-G/-U/--interleaved options.
Finished in 151.54 s (24 us/read; 2.55 M reads/minute).

=== Summary ===

Total read pairs processed:          6,440,847
  Read 1 with adapter:               3,875,741 (60.2%)
  Read 2 with adapter:                       0 (0.0%)
Pairs that were too short:             112,847 (1.8%)
Pairs written (passing filters):     6,328,000 (98.2%)
```

- **cutadapt** started with 6,440,847 read pairs
  - 112,847 reads were discarded as too short after the R1 adapter was removed
    - the same 112,847 reads were discarded from both the R1 and R2 files
  - the remaining 6,328,000 were then subjected to pass2 processing.

The corresponding **yeast_rnaseq.acut.pass2.log** file looks like this:

cutadapt pass2 log file

```
This is cutadapt 1.18 with Python 3.7.1
Command line parameters: -m 32 -a TGATCGTCGGACTGTAGAACTCTGAACGTGTAGA --paired-output yeast_rnaseq_R1.cuta.fastq
-o yeast_rnaseq_R2.cuta.fastq yeast_rnaseq_R2.tmp.cuta.fastq yeast_rnaseq_R1.tmp.cuta.fastq
Processing reads on 1 core in paired-end legacy mode ...
Finished in 83.64 s (13 us/read; 4.54 M reads/minute).

=== Summary ===

Total read pairs processed:          6,328,000
  Read 1 with adapter:                  90,848 (1.4%)
  Read 2 with adapter:                       0 (0.0%)
Pairs that were too short:                   0 (0.0%)
Pairs written (passing filters):     6,328,000 (100.0%)

Total basepairs processed: 1,198,172,994 bp
  Read 1:   639,128,000 bp
  Read 2:   559,044,994 bp
Total written (filtered):  1,197,894,462 bp (100.0%)
```

- **cutadapt** started with 6,328,000 pass1 reads
  - no additional reads were discarded as too short after the R2 adapter was removed
  - so new R1 and R2 files, both with 6,328,000 reads, were produced
    - **yeast_rnaseq_R1.cuta.fastq.gz** and **yeast_rnaseq_R2.cuta.fastq.gz**

Exercise: Verify that both adapter-trimmed yeast_rnaseq fastq files have 6,328,000 reads

```
echo "$((`zcat yeast_rnaseq_R1.cuta.fastq.gz | wc -l` / 4))"
zcat yeast_rnaseq_R2.cuta.fastq.gz | wc -l | awk '{printf("%d\n", $1/4)}'
```

For more on **printf**, which is available in most programming languages, see https://alvinalexander.com/programming/printf-format-cheat-sheet/