Using MultiQC

Using MultiQC to produce consolidated QC Reports

Byte Club, October 18 2017
Anna Battenhouse, CSSB & CCBB.

- Overview
- Code Workshop
  - Setup to follow along
  - Produce a consolidated FastQC report
  - Add a few customizations
  - Tips for working with the MultiQC configuation file
  - Add reports from a bowtie2 alignment
  - Fix the Picard MarkDuplicates sample name
  - Controlling report section order
  - About MultiQC custom data
  - Adding a custom linegraph
  - Adding custom bargraphs
  - Making MultiQC run faster and be less confused
- References
  - Main MultiQC links
  - MultiQC configuration files
  - MultiQC custom data support
- Example Reports from Anna
  - Igor Ponomarev ATAC-seq data
  - Marcotte lab amplicon sequencing

Overview

- **MultiQC** is a tool for aggregating NGS QC reports.
  - It does not produce reports, just ***combines*** them for unified visualization.
- MultiQC "knows" the report formats of many existing NGS tools:
  - **FastQC, cutadapt, bowtie2, tophat, STAR, kallisto, HISAT2, samtools, featureCounts, HTSeq, MACS2, Picard, GATK**
  - … and more!
- MultiQC can also be configured to display other data via two straightforward steps:

1. format the data appropriately (e.g. tab-delimited text files)
2. create appropriate custom data entries in a **multiqc_config.yaml** configuration file

- MultiQC produces neat, *interactive* plots in an HTML file.
  - So it can be used as a basic plotting tool for many kinds of reports and data, not just those produced by NGS tools!

I recommend using Chrome to view MultiQC reports.

The HTML reports generated by MultQC rely *heavily* on JavaScript and other dynamic web content scripting tools, and not all browsers support them equally well.

Code Workshop

ATAC-seq is a transposon-insertion sequencing method where an engineered, activate transposon inserts in accessible ("open") chromatin. It is considered to be a much simpler protocol to standard DNase-seq, and requires less starting material as well.

For data, we will use some ATAC-seq datasets produced in Igor Ponomarev's lab in WCAAR. As a proof-of-concept for future work, they performed the ATAC-seq protocol on 5k and 50k cell nuclei from mouse brain, producing 2 paired-end datasets.

Setup to follow along

Login to **ls5** or **stampede** at TACCTACC. Execute commands to set up access to the **multiqc** binary:

lonestar5 setup for multiqc

```
module load python
export PATH="/work/projects/BioITeam/ls5/opt/multiqc-1.0:$PATH"
export PYTHONPATH="/work/projects/BioITeam/ls5/lib/python2.7/annab-packages:$PYTHONPATH"

# make sure it is working...
multiqc --help
```

stampede setup for multiqc

```
module load python
export PATH="/work/projects/BioITeam/stampede/opt/multiqc-1.0:$PATH"
export PYTHONPATH="/work/projects/BioITeam/stampede/lib/python2.7/annab-packages:$PYTHONPATH"

# make sure it is working...
multiqc --help
```

Produce a consolidated FastQC report

The FastQC took is great for producing detailed reports for every individual fastq file. For example, for Igor's 2 PE datasets, 4 reports are produced from running **fastqc** (http://web.corral.tacc.utexas.edu/iyer/igor/fastqc/).

The shortcoming is that you have to browse through all the individual reports one at a time, which can be tedious for large experiments.

This is where MultiQC's power comes in. You can point MultiQC to a directory where FastQC has been run and it will magically produce a consolidated report.

For example, logged in to **ls5** at TACC, first stage a directory where FastQC has been run:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
ln -s -f /work/projects/BioITeam/projects/byteclub/multiqc/fastqc
```

Now this is all it takes to produce a basic MultiQC report:

```
cd $SCRATCH/byteclub/multiqc
multiqc .
```

When this completes you'll see a new file and directory:

- **multiqc_report.html** – the MultiQC HTML report with its default name
- **multiqc_data** – directory with text files containing MultiQC data used in the report as well as a log file

Here's what this basic FastQC report looks like: http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/01_basic.multiqc_report.html

To view the file you created in a web browser, it must be copied somwhere where a browser can open it. An easy way to do this is to copy it to your laptop like this, for example, changing the user name from **abattenh** and scratch path as appropriate.

```
# from your laptop:
scp -p abattenh@ls5.tacc.utexas.edu:/scratch/01063/abattenh/byteclub/multiqc/multiqc_report.html .
```

Add a few customizations

MultiQC reports can be customized by creating a file called **multiqc_config.yaml** in the directory where you call **multiqc**.

Use your favorite text editor to create a a file called **multiqc_config.yaml** in your **$SCRATCH/byteclub/multiqc** directory as shown below. This will add report title lines and change the names of the MultiQC output files.

multiqc_config.yaml

```
# Titles to use for the report.
title: "ATAC-Seq QC Reports"
subtitle: null
intro_text: "MultiQC reports for Igor's ATAC-Seq proof-of-concept project."
report_header_info:
    - Sequenced by: 'GSAF'
    - Job: 'JA17277'
    - Run: 'SA17121'
    - Setup: '2x150'

# Change the output filenames
output_fn_name: mqc_report.html
data_dir_name: mqc_report_data
```

To catch up, just stage Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc/
cd $SCRATCH/byteclub/multiqc/
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/02_fastq/ .
```

After saving this file, remove the previous MultiQC outputs and re-run the program:

```
cd $SCRATCH/byteclub/multiqc
rm -rf multiqc_data multiqc_report.html
multiqc .
```

If all went well, you should now see a **mqc_report.html** file and a **mqc_report_data** directory. Your newly-generated **mqc_report.html** report file in should look like this (note the new title and header): http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/02_custom.mqc_report.html.

Tips for working with the MultiQC configuation file

Here are a few tips for working with the MultiQC configuration file.

- Always use spaces (not tabs!) in the **multiqc_config.yaml** file.
- Make sure the file is saved with Unix line endings (not Windows or Mac).
- Pay attention to the output when running **multiqc**. It will tell you if there are issues parsing the config file.
- Always delete any previous MultiQC output files before running **multiqc**
  - While their documentation says existing files will just be updated, I have seen MultiQC get confused when previous reports exist.
- It is a good idea to change the name of the MultiQC output files
  - If output files with those names are not created, something went wrong!
- Consult example config files
  - An example **multiqc_config.yaml** file: https://github.com/ewels/MultiQC/blob/master/multiqc_config_example.yaml
  - All **multiqc_config.yaml** defaults: https://github.com/ewels/MultiQC/blob/master/multiqc/utils/config_defaults.yaml
- Avoid running **multiqc** on large complex directory trees.
  - Instead, create a separate directory (or directory tree) only for MultiQC
    - Copy or link the files you want MultiQC to look for there, and use it as MultiQC's target directory.
  - MultiQC will run much faster and have fewer confusions.

Add reports from a bowtie2 alignment

First stage some **mm10** **bowtie2** alignment data:

```
cd $SCRATCH/byteclub/multiqc
rsync -avrP /work/projects/BioITeam/projects/byteclub/multiqc/bowtie2/ bowtie2/
```

Take a look at the contents of the **bowtie2** directory. It contains typical output files from running Anna's **align_bowtie2_illumina.sh** alignment script.

MultiQC will look at *all* files in this directory looking for report formats it understands. Here, reports that MultiQC will recognize as-is include:

- **<prefix>.flagstat.txt** - output from running **samtools flagstat**
- **<prefix>.idxstats.txt** - output from running **samtools idxstats**
- **<prefix>.dupinfo.txt** - output from running **Picard MarkDuplicates**

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc/
cd $SCRATCH/byteclub/multiqc/
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/03_bowtie/ .
```

Now run **multiqc** again:

```
cd $SCRATCH/byteclub/multiqc
rm -rf mqc_report*
multiqc .
```

If all went well, you should now see a **mqc_report.html** file that looks like this: http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/03_bowtie.mqc_report.html, with new sections for **Picard** and **Samtools** reports.

Fix the Picard MarkDuplicates sample name

Notice there is something odd going on in the new **General Statistics** section. We see **M Reads Mapped** entries for samples called **brain_50k_nuclei** and **brain_5k_nuclei**, but **% Dups** entries for samples named **brain_50k_nuclei.sort** and **brain_5k_nuclei.sort**.

To see where a **General Statistics** column comes from, hover over the column header. Doing this tells us that the the **M Reads Mapped** figures came from the **samtools flagstat** report, while the **% Dups** comes from **Picard MarkDuplicates**.

Take a look at one of the **<prefix>.dupinfo.txt** files to see what might be going on. Below I've added line breaks to the command line info for clarity.

brain_5k_nuclei.dupinfo.txt

```
## htsjdk.samtools.metrics.StringHeader
# picard.sam.markduplicates.MarkDuplicates INPUT=[brain_5k_nuclei.sort.bam]
OUTPUT=brain_5k_nuclei.sort.dup.bam
METRICS_FILE=brain_5k_nuclei.dupinfo.txt ASSUME_SORTED=true VALIDATION_STRINGENCY=LENIENT
MAX_SEQUENCES_FOR_DISK_READ_ENDS_MAP=50000 MAX_FILE_HANDLES_FOR_READ_ENDS_MAP=8000
SORTING_COLLECTION_SIZE_RATIO=0.25 TAG_DUPLICATE_SET_MEMBERS=false
REMOVE_SEQUENCING_DUPLICATES=false TAGGING_POLICY=DontTag REMOVE_DUPLICATES=false
DUPLICATE_SCORING_STRATEGY=SUM_OF_BASE_QUALITIES
PROGRAM_RECORD_ID=MarkDuplicates PROGRAM_GROUP_NAME=MarkDuplicates
READ_NAME_REGEX=<optimized capture of last three ':' separated fields as numeric values>
OPTICAL_DUPLICATE_PIXEL_DISTANCE=100 VERBOSITY=INFO QUIET=false COMPRESSION_LEVEL=5
MAX_RECORDS_IN_RAM=500000 CREATE_INDEX=false CREATE_MD5_FILE=false
GA4GH_CLIENT_SECRETS=client_secrets.json
## htsjdk.samtools.metrics.StringHeader
# Started on: Wed Jul 05 23:20:57 CDT 2017

## METRICS CLASS        picard.sam.DuplicationMetrics
LIBRARY UNPAIRED_READS_EXAMINED READ_PAIRS_EXAMINED     SECONDARY_OR_SUPPLEMENTARY_RDS  UNMAPPED_READS
UNPAIRED_READ_DUPLICATES        READ_PAIR_DUPLICATES    READ_PAIR_OPTICAL_DUPLICATES    PERCENT_DUPLICATION
ESTIMATED_LIBRARY_SIZE
brain_5k_nuclei 0       28666117        0       16562322        0       12504025        902024  0.436 195
23118972

## HISTOGRAM    java.lang.Double
BIN     VALUE
1.0     1.016471
...
```

This is a standard metrics file produced by running **Picard MarkDuplicates** with a **METRICS_FILE** option specified. Note the **INPUT** bam filename: **brain_5k_nuclei.sort.bam**. MultiQC parses this report looking for the **INPUT** file, and uses it (minus the **.bam** suffix) as the sample name. So **brain_5k_nuclei.sort.bam** becomes sample name **brain_5k_nuclei.sort**.

The solution is to modify the metrics so that it has **brain_5k_nuclei.bam** as its **INPUT** option. This sounds straightforward but there are a couple of gocha's.

- We don't want to modify the original metrics file, as it is a faithful record of what was executed. So we want to make a *copy* of the metrics file with the modified **INPUT** parameter.
- We want MultiQC to ignore the original metrics file, and only use our "fixed" metrics log.
    - But MultiQC will, by default, look at *all* reports in the specified analysis directory (and its sub-directories), and will then find *two* metrics reports.

For the first part of the solution, we'll create a modified version of the metrics files, but not in the alignment directory, but in a new **for_multiqc** directory.

```
mkdir -p $SCRATCH/byteclub/multiqc/for_multiqc
cd $SCRATCH/byteclub/multiqc/for_multiqc
for f in ../bowtie2/*.dupinfo.txt; do
  bn=`basename $f`
  pfx=${bn%%.dupinfo.txt}
  echo "$f - $pfx"
  cat $f | sed 's/[.]sort//g' > ${pfx}.dupmetrics.txt
done
```

Your **$SCRATCH/byteclub/multiqc/02_bowtie/for_multiqc** directory should have 2 files:

- **brain_50k_nuclei.dupmetrics.txt**
- **brain_50k_nuclei.dupmetrics.txt**

The final piece of the puzzle is to tell MultiQC to ignore the original **<prefix>.dupinfo.txt** files by modifying the **multiqc_config.yaml** file, adding a **fn_ignore_files** list entry.

multiqc_config.yaml

```
# Titles to use for the report.
title: "ATAC-Seq QC Reports"
subtitle: null
intro_text: "MultiQC reports for Igor's ATAC-Seq proof-of-concept project."
report_header_info:
    - Sequenced by: 'GSAF'
    - Job: 'JA17277'
    - Run: 'SA17121'
    - Setup: '2x150'

# Change the output filenames
output_fn_name: mqc_report.html
data_dir_name: mqc_report_data

# Ignore these files / directories / paths when searching for reports
fn_ignore_files:
    - '*.dupinfo.txt'
```

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/04_picard_fixed/ .
```

After making this config file modification, you can now run **multiqc** again:

```
cd $SCRATCH/byteclub/multiqc; rm -rf mqc_report*; multiqc .
```

The resulting report should look like this: , with a cleaned up **General Statistics** table.

Controlling report section order

You may have noticed that MultiQC lists its report sections in fairly random order. If you're Type-A like me, you'll want sections ordered to reflect the sequence of analyses performed. This can be controlled by adding a list of **top_module** entries.

multiqc_config.yaml

```
# Titles to use for the report.
title: "ATAC-Seq QC Reports"
subtitle: null
intro_text: "MultiQC reports for Igor's ATAC-Seq proof-of-concept project."
report_header_info:
    - Sequenced by: 'GSAF'
    - Job: 'JA17277'
    - Run: 'SA17121'
    - Setup: '2x150'

# Change the output filenames
output_fn_name: mqc_report.html
data_dir_name: mqc_report_data

# Ignore these files / directories / paths when searching for reports
fn_ignore_files:
    - '*.dupinfo.txt'

# Modules that should come at the top of the report
top_modules:
    - 'generalstats'
    - 'fastqc'
    - 'samtools'
    - 'picard'
```

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/05_section_order/ .
```

After making this config file modification, you can now run **multiqc** again:

```
cd $SCRATCH/byteclub/multiqc; rm -rf mqc_report*; multiqc .
```

Producing a report like this: http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/05_section_order.mqc_report.html, with a section order that more closely follows workflow processing steps.

About MultiQC custom data

When MultiQC does not know about data produced by a program it doesn't know about, it has a mechanisms for adding custom report sections. The simple way to do this is *declaratively*, (i.e., via configuation parameters) as described below. (You can also write a Python module for very fine-grained control, but that is a *lot* more work.)

To add a section for custom data:

1. Format the data appropriately
    - MultiQC supports a number of data file formats (yaml, comma-separated values, etc.)
        - I recommend using simple tab-delimited text files, with MultiQC's preferred **.tsv** extension.
    - data can be provided as one file per sample (where the sample name is part of the file name)
        - or as a single table-like file containing data for all samples
2. Add two required custom data section entries in the **multiqc_config.yaml** configuration file
    - a **sp** (search path) section for finding report data
        - specifying a wildcard pattern, if data is supplied as one file per sample
        - or a single file name for a consolidated data file
    - each report has a user-named section under a single **custom_data** section.
        - the required **id** attribute must be unique, and ties the **custom_data**, **sp** and **custom_content** sections
        - other important attributes include **description**, **file_format**, and **plot_type**.
        - a **pconfig** sub-section contains plot configuration options
3. Specify the ordering of the custom report section (optional)
    - add a **custom_content** section **order** list entry

See http://multiqc.info/docs/#configuration for more details about the structure of custom data sections in **multiqc_config.yaml**.

MultiQC supports these custom plot types:

- **bargraph**
- **linegraph**
- **scatter**
- **table**
- **generalstats**
- **beeswarm**
- **heatmap**

Plot-type-specific plotting options can be specified in each custom data report's **pconfig** section. See http://multiqc.info/docs/#plotting-functions for more information. While this section is written for Python programming, the options listed in each plot type's "**config**" block can be used in the plot's plot's **pconfi** section (modulo the Python versus YAML formatting differences).

Adding a custom linegraph

Here we'll create a **linegraph** report of insert sizes for the **bowtie2** alignments.

We start with the **<prefix>.insertsz.txt** files produced by Anna's **align_bowtie2_illumina.sh** script. That file has both positive and "negative" insert sizes with read counts for all mates, proper pairs, and R1s and R2s individually. For a data file compatible with MultiQC's **linegraph**, we want only two columns: the positive insert size and count of proper pairs with that size (negative sizes are redundant since each proper pair will have one positive and one negative size entry of the same magnitude).

So a bit of command line reformatting is needed to produce files for MultiQC, which we will save in our **for_multiqc** directory.

```
cd $SCRATCH/byteclub/multiqc/for_multiqc
for f in ../bowtie2/*.insertsz.txt; do
  bn=`basename $f`
  pfx=${bn%%.insertsz.txt}
  echo "$f - $pfx"
  tail -n +2 $f | grep -v -P '^-' | cut -f 1,3 > ${pfx}.bowtie2_isizes.tsv
done
```

Next we edit the **multiqc_config.yaml** configuration file to add appropriate custom data sections:

multiqc_config.yaml

```
# Titles to use for the report.
title: "ATAC-Seq QC Reports"
subtitle: null
intro_text: "MultiQC reports for Igor's ATAC-Seq proof-of-concept project."
report_header_info:
    - Sequenced by: 'GSAF'
    - Job: 'JA17277'
    - Run: 'SA17121'
    - Setup: '2x150'

# Change the output filenames
output_fn_name: mqc_report.html
data_dir_name: mqc_report_data

# Ignore these files / directories / paths when searching for reports
fn_ignore_files:
    - '*.dupinfo.txt'

# Modules that should come at the top of the report
top_modules:
    - 'generalstats'
    - 'fastqc'
    - 'samtools'
    - 'picard'

# --------------------------------
# Custom data
# --------------------------------

custom_data:
    bowtie2_isize:
        id: 'bowtie2_isize_section'
        section_name: 'Bowtie2 insert size'
        description: 'distribution for alignments (bowtie2 --local -X2000 --no-mixed --no-discordant)'
        file_format: 'tsv'
        plot_type: 'linegraph'
        pconfig:
            id: 'bowtie2_isize_plot'
            title: 'Insert sizes for proper pairs'
            xlab: 'Insert size'
            ylab: 'Count'
sp:
    bowtie2_isize_section:
        fn: '*.bowtie2_isizes.tsv'
```

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/06_custom_linegraph/ .
```

Then the usual...

```
cd $SCRATCH/byteclub/multiqc; rm -rf mqc_report*; multiqc .
```

Resulting in a report that includes our inset size distribution data the custom data section we configured: http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/06_custom_linegraph.mqc_report.html, with a new section called **Bowtie2 insert size**.

What's cool is that this "sawtooth" insert size distribution occurs because of the way transposons insert into the major groove of DNA at regular intervals. So this graph shows Igor that his ATAC-seq proof-of-concept experiment worked!

Adding custom bargraphs

Here we'll create two custom **bargraph** reports, one for **bowtie2** mapping qualities and a second showing genome coverage of the alignments.

The data files for both reports are pretty simple, but it took a bit of scripting to create them. So let's just use pre-made copies:

```
cd $SCRATCH/byteclub/multiqc
cp /work/projects/BioITeam/projects/byteclub/multiqc/07_custom_bargraph/for_multiqc/*mapq*      for_multiqc/
cp /work/projects/BioITeam/projects/byteclub/multiqc/07_custom_bargraph/for_multiqc/*genomecov* for_multiqc/
```

There is one mapping quality histogram for each dataset, with category names in the 1st column and counts in the 2nd. The 50k dataset file looks like this:

brain_50k_nuclei.mapq_histogram.tsv

```
q0            137354
1-9            671546
10-19        1081868
20-29        1945926
30-39        1508496
40+            12930272
```

There is just one data file for genome coverage. Unlike the per-sample files, it has a header, with dataset names in the 1st column, followed by category names and their counts in subsequent columns. (I've re-formatted the data below for readability, but remember that all **.tsv** file data must be ***tab***-separated.)

combined_genomecov.tsv

```
sample      none        1-2        3-10       11-50     51+
5k_nuclei   2140984435  237947623  308665107  38729079  4545530
50k_nuclei  2175228345  351105871  186361275  17356704  819579
```

Here we edit the **multiqc_config.yaml** configuration file to add appropriate custom data sections:

multiqc_config.yaml

```
# Titles to use for the report.
title: "ATAC-Seq QC Reports"
subtitle: null
intro_text: "MultiQC reports for Igor's ATAC-Seq proof-of-concept project."
report_header_info:
    - Sequenced by: 'GSAF'
    - Job: 'JA17277'
    - Run: 'SA17121'
    - Setup: '2x150'

# Change the output filenames
output_fn_name: mqc_report.html
data_dir_name: mqc_report_data

# Ignore these files / directories / paths when searching for reports
fn_ignore_files:
    - '*.dupinfo.txt'

# Modules that should come at the top of the report
top_modules:
    - 'generalstats'
    - 'fastqc'
    - 'samtools'
    - 'picard'

# --------------------------------
# Custom data
# --------------------------------
custom_content:
  order:
    - bowtie2_isize_section
    - bowtie2_mapq_section
    - genome_coverage_section
custom_data:
    bowtie2_isize:
        id: 'bowtie2_isize_section'
        section_name: 'Bowtie2 insert size'
        description: 'distribution for alignments (bowtie2 --local -X2000 --no-mixed --no-discordant)'
        file_format: 'tsv'
        plot_type: 'linegraph'
        pconfig:
            id: 'bowtie2_isize_plot'
            title: 'Insert sizes for proper pairs'
            xlab: 'Insert size'
            ylab: 'Count'
    bowtie2_mapq:
        id: 'bowtie2_mapq_section'
        section_name: 'Mapping quality'
        description: 'distribution for aligned reads before filtering'
        file_format: 'tsv'
        plot_type: 'bargraph'
        pconfig:
            id: 'bowtie2_mapq_plot'
            title: 'Mapping quality scores'
            ymax: 60000000
    genome_coverage:
        id: 'genome_coverage_section'
        section_name: 'Genome coverage'
        description: 'of mapped inserts (bedtools genomecov -fs), grouped into coverage count catgories'
        file_format: 'tsv'
        plot_type: 'bargraph'
        pconfig:
            id: 'genome_coverage_plot'
            title: 'Position coverage by coverage count category'
            logswitch: True
            stacking: null
sp:
    bowtie2_isize_section:
        fn: '*.bowtie2_isizes.tsv'
    bowtie2_mapq_section:

        fn: '*.mapq_histogram.tsv'
    genome_coverage_section:
        fn: 'combined_genomecov.tsv'

# file suffixes to remove when generating sample names...
extra_fn_clean_exts:
    - type: 'replace'
      pattern: '.mapq_histogram.tsv'
    - type: 'replace'
      pattern: '.genomecov.tsv'
```

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/07_custom_bargraph/ .
```

Then the usual...

```
cd $SCRATCH/byteclub/multiqc; rm -rf mqc_report*; multiqc .
```

Resulting in a report that includes our new **Mapping quality** and **Genome coverage** sections, that should look like this: http://web.corral.tacc.utexas.edu/iyer/byteclub/multiqc/07_custom_bargraph.mqc_report.html.

Making MultiQC run faster and be less confused

By default, MultiQC scans ***all*** files in the analysis directory you specify. This can take quite a while for complex directory hierarchies with many files that will not be used by MultiQC.

Additionally, MultiQC can get confused when the same (or similar) data is found in different files, or in different directories.

To address these issues, it is a good practice to copy everything you want MultiQC to process into a single directory, then either specify just that directory on the **multiqc** command line (e.g. **multiqc for_multiqc**), or exclude other directories in the **multiqc_config.yaml** file.

For example, here we can stage all the reports we want MultiQC to process in our **for_multiqc** directory:

```
cd $SCRATCH/byteclub/multiqc/for_fastqc
ln -s -f ../fastqc
cp -p ../bowtie2/*.flagstat.txt  .
cp -p ../bowtie2/*.idxstats.txt  .
```

Your **for_multiqc** directory should now everything we want MultiQC to use:

```
brain_50k_nuclei.bowtie2_isizes.tsv
brain_50k_nuclei.dupmetrics.txt
brain_50k_nuclei.flagstat.txt
brain_50k_nuclei.idxstats.txt
brain_50k_nuclei.mapq_histogram.tsv
brain_5k_nuclei.bowtie2_isizes.tsv
brain_5k_nuclei.dupmetrics.txt
brain_5k_nuclei.flagstat.txt
brain_5k_nuclei.idxstats.txt
brain_5k_nuclei.mapq_histogram.tsv
combined_genomecov.tsv
fastqc
```

To catch up, just use Anna's pre-made files:

```
mkdir -p $SCRATCH/byteclub/multiqc
cd $SCRATCH/byteclub/multiqc
rsync -avrP --delete /work/projects/BioITeam/projects/byteclub/multiqc/08_final/ .
```

Run MultiQC again, but this time just point it

```
cd $SCRATCH/byteclub/multiqc
rm -rf mqc_report*
multiqc for_multiqc
```

Alternatively, you could exclude the **bowtie2** directory entirely via a **fn_ignore_dirs** section list item in **multiqc_config.yaml**, like this:

References

Main MultiQC links

- Website: http://multiqc.info/
- Documentation: http://multiqc.info/docs
- MultiQC Github repo: https://github.com/ewels/MultiQC
- MultiQC test data repo: https://github.com/ewels/MultiQC_TestData

MultiQC configuration files

- an example **multiqc_config.yaml** file: https://github.com/ewels/MultiQC/blob/master/multiqc_config_example.yaml
- all **multiqc_config.yaml** defaults: https://github.com/ewels/MultiQC/blob/master/multiqc/utils/config_defaults.yaml

MultiQC custom data support

- structure of the custom data area of **multiqc_config.yaml**:
  - http://multiqc.info/docs/#configuration
- available plot types:
  - http://multiqc.info/docs/#plotting-functions

    - while this section is written for Python programming, the options listed in each plot type's "**config**" block can be specified declaratively in any plot's **pconfig** section in the **multiqc_config.yaml**.
- example custom data files from their test data repo:
    - https://github.com/ewels/MultiQC_TestData/tree/master/data/custom_content/no_config
    - https://github.com/ewels/MultiQC_TestData/tree/master/data/custom_content/embedded_config

Example Reports from Anna

Below are descriptions of two projects I've assisted with lately using MultiQC to help pull together visualizations assessing experiment quality.

- These example MultiQC reports below were generated by running the **multiqc** binary on a command line.
- After inspecting them locally (by just opening them as files in a web browser), they were copied to a web-accessible location to share with others. Here, that location is Iyer Lab's web-accessible directory on **corral**.

Igor Ponomarev ATAC-seq data

ATAC-seq is a transposon-insertion sequencing method where an engineered, activate transposon inserts in accessible ("open") chromatin. It is considered to be a much simpler protocol to standard DNase-seq, and requires less starting material as well.

Igor Ponomarev's lab (in WCAAR) performed the ATAC-seq protocol on 5k and 50k cell nuclei from mouse brain, producing 2 paired-end datasets.

- http://web.corral.tacc.utexas.edu/iyer/igor/mqc_report.html
- has both standard and custom data reports

Marcotte lab amplicon sequencing

The Marcotte lab is working on a deep mutational screening project of a human gene transformed into yeast as an amplicon on a plasmid. Here, the gene is **MVK**, a gene in the yeast cholesterol biosynthesis pathway. The **hsMVK** gene is amplified with an error-prone polymerase to produce point mutations. Both the native yeast gene and the human ortholog (with which it shares no sequence similarity) are under on/off promoter control. The idea is to compare the mutations that accumulate in the active **hsMVK** gene, after many growth cycles, with a background in which the **hsMVK** gene is present but not active (the yeast **MVK**is doing the work) to see which mutations are favored or disfavored. As part of this project, Riddhiman Garge produced 19 datasets.

- basic FastQC report
    - http://web.corral.tacc.utexas.edu/iyer/mvk/mvk_mqc_report.fastqc.html
- report on **BWA mem** alignments of the datasets to **hsMVK** amplicon and plasmid backbone contigs
    - http://web.corral.tacc.utexas.edu/iyer/mvk/mvk_mqc_report.bwa.html
    - standard reports from **samtools flagstat, samtools idxstats, Picard MarkDuplicates**
    - custom data reports from **bedtools genomecov** and from insert size distribution data Anna computed
- report using custom data from a specialized deep mutational screening tool from the Jesse Bloom lab
    - http://web.corral.tacc.utexas.edu/iyer/mvk/mvk_mqc_report.jbloom.html
    - this tool looks only at the overlapping portions of paired-end R1 and R2 reads