

MultiQC: Summarise analysis results in a single report

Philip Ewels^{1,*}, Måns Magnusson¹, Max Käller¹.

1 Department of Biochemistry and Biophysics, Science for Life Laboratory, Stockholm University, Stockholm 106 91, Sweden

—* To whom correspondence should be addressed.—

Abstract

As large scale genomics studies become more common, fast and accurate quality control is increasingly important. Here, we describe MultiQC: a tool to summarize key metrics across all samples in a project into a single report. A modular design allows MultiQC to work with multiple analysis types. Templating and process triggers provide a framework for simple extension and customization. Reports enable a fast overview of the entire analysis and highlight anomalous samples and problematic steps.

Introduction

Advances in next-generation sequencing are leading to an avalanche of data. Whilst opening doors to new analysis types and experimental designs, this provides a challenge for bioinformaticians. The sheer volume of data associated with a typical project requires high throughput processing and analysis, often yielding vast numbers of log files and quality metrics. Additionally, large numbers of samples increase the chance of errors and sample swaps - the earlier such mistakes can be detected the better. This leads to the complex and time consuming task of checking multiple statistics through numerous reports and log files for every sample.

MultiQC addresses this problem by searching an analysis directory for any recognized content and creating a single summary report. It is a general use tool, designed to parse log files and create plots with overlaid data. This provides the researcher with a fast method to scan all key statistics for a project quickly and easily. Additionally, shared plots allow accurate comparison between samples which is not possible when switching between different reports. It is suitable for integration with existing workflows with little to no configuration and its routine use can save a great deal of analysis time.

Installation and Usage

MultiQC is a Python package available on the Python Package Index (PyPI) and GitHub. It can be installed on the command line using the common pip package, with the command `pip install multiqc`. MultiQC is compatible with any system running Python 2.7+ or Python 3.2+, including all Linux distributions, Mac OS X and Windows users who have installed Python. It has a small number of Python package dependencies, automatically installed by pip. A graphical interface is available for Mac OS X users, which can be downloaded from GitHub. Users of Galaxy and BaseSpace can find MultiQC available as an app.

Once installed, MultiQC can be run on the command line with the command `multiqc <analysis directory>`. This searches the specified path recursively for any recognized log files, and compiles a report. Reports are interactive web pages, viewable with any modern web browser. They are also saved as zip files for easy sharing. Parsed data is saved as tab delimited text files within the report directory for downstream use.

Due to the modular design of MultiQC, it is run in the same manner for any analysis type. Each module will run only if it's log files are present, meaning that no configuration is required. At the time

of writing, MultiQC comes with modules to parse logs from a range of tools including FastQC, FastQ Screen, Cutadapt, Bowtie 1/2, STAR, Tophat, Bismark, Picard and others. This library is poised to expand quickly following general release.

Report Tools

Within the MultiQC report there are a number of features to aid interpretation and usage. Modules add columns to a shared ‘General Statistics’ table at the top of the report, allowing the analysis progression to be easily quantified (Fig 1). Automated visualisation of these numbers and tools for sorting and sample ordering allows fast identification of patterns and sample groupings.

Sample Name	% Mapped	M Mapped	Trimmed	% Dups	% GC	Length	M Seqs ▾
7522	61.8%	22.0	1.5%	13.3%	40%	36	35.7
7510	50.6%	17.4	2.0%	11.4%	40%	35	34.3
7514	70.2%	23.6	3.1%	6.6%	44%	35	33.6
7503	63.2%	19.3	2.1%	12.9%	44%	35	30.5
7519	81.1%	19.9	2.3%	5.8%	42%	35	24.6
7505	79.1%	14.2	3.5%	7.8%	47%	35	18.0

Figure 1: General Statistics table, showing progression of samples through analysis pipeline.

Plots showing data from each module are generated using the HighCharts JavaScript library (Highsoft, 2015). They are resizable and interactive, some with click and drag zooming. Plots can be exported in a range of formats, including PNG, SVG and PDF.

MultiQC reports also have a ‘Toolbox’. Samples can be highlighted using search strings, optionally using regular expressions, to easily visualize groups within plots (Fig 2). Samples can be renamed, with a bulk option allowing data to be pasted from a spreadsheet. Matched samples can also be temporarily hidden from the report. Settings within the toolbox can be saved for the report or set as defaults for all reports. They can also be saved as a file for easy sharing with others.

A step by step tour is available in all reports to familiarize new users with the available features.

Customization and Extension

To generate the reports, MultiQC uses the Jinja2 template framework (as used by Django and others). This method allows simple customization of report output, such as branding or additional features.

MultiQC requires one module for every type of output it recognizes. Inevitably, some users will want MultiQC to work with programs which are not yet supported. To facilitate this, MultiQC has been designed as a framework with an object-orientated class based structure, making the process of writing extensions simple. Typically, a new module will use a base class function to find relevant log files, it will parse these with a custom set of regular expressions, and then pass these data to set of base class plotting functions. Such plots will be compatible with the report tools described above. If greater customization is required, a library of JavaScript functions is provided, or completely bespoke code can be used. All helper code is thoroughly described in the documentation, with examples.

It is hoped that users will contribute new modules back to the core distribution using a GitHub pull request. However, if modules or templates are too niche to be of use to others, code can be developed in an add-on Python package. This style of extension allows separate code tracking, keeping the

