THE

FASTQC pipeline

MANUAL

Kim Carter

TODAY

Contents

Contents ii

1 Introduction 1

2 File: fastqc.groovy 2

3 Example running the pipeline 4

CHAPTER 1

Introduction

This Markdown document provides a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines. FastQC provides a modular set of analyses which you can use to give a quick impression of whether your data has any problems of which you should be aware before doing any further analysis. For more info on FastQC - see the project website. The pipeline has been implemented as a Bpipe pipeline, written in the Groovy programming language. The code blocks in this Markdown document are parsed and used to create the Groovy file that is used by Bpipe to run the pipeline.

CHAPTER 2

File: fastqc.groovy

When the MEdical Sequence Analysis Package (MESAP) is built, this Markdown file (`fastqc.md`) is used to create a file called `fastqc.groovy`, which is processed by Bpipe. The file begins with a title definition and the base directory is defined; `BASEDIRNAME` will be interpolated into the top directory when creating the Groovy file.

```
about title: "FastQC pipeline."
def BASEROOTDIR="BASEDIRNAME"
```

FastQC can be run as a GUI program and as a command-line program (as we use it here). This program is distributed with the MESAP and declared as a variable here.

```
def FASTQC = "$BASEROOTDIR" + "/programs/FastQC/FastQC"
```

The following routine extracts the filename part out of the full part of any input file - the routines expects files to be in the format: XXX_XXX_BARCODE_LANE_R.fastq.gz.

```
def get_sample_filename_nopath_noextension(filename)
{

  def returned_value = ""

  // strip path
  def m = filename.split("/")[-1]

  //return first part of file name irrespective of extensions (s)
  return m.split("\\.")[0]
}
```

The main (first) step of the pipeline is to run the FastQC program over each of the individual input .fastq / .fastq.gz files.

```
run_fastqc = {
        output.dir = "qc"
```

```
    doc "Run fastqc to QC fastQ file"

    def filename = get_sample_filename_nopath_noextension("
       $input1")

    produce (filename+"_fastqc.zip",filename+"_fastqc.html")
    {
        exec "$FASTQC -o $output.dir $input1"
    }
}
```

The second step of the pipeline creates an overview summary file, across the main set of quality measures in FastQC. You are presented with numbers (and %s) of files that PASS, flag a WARNING, or FAIL each test. If you have a large number of files to QC, this output file can be a good first step to narrowing down QC areas that may be of importance for further examination.

```
run_fastqc_parser = {
        output.dir = "qc"

        doc "Run fastqc_parser to summarise ouput across all
           files"
        produce ("fastqc_summary.txt")
        {
                exec "perl ../scripts/fastqc_parser.pl qc/ > qc/
                   fastqc_summary.txt"
        }
}
```

The code below defines the pipeline and how it should be run - each input .fastq/.fastq.gz file is treated separately (there is no paired checking mode). The pipeline generates a HTML file (each named inputfile_fastqc.html) and a .zip file (named inputfile_fastqc.zip) containing text and graphical summaries of the FastQC output for each sample respectively. The pipeline also produces a summary of all of the FastQC output files contained in the output directory (the qc/ subdirectory off of whereever the pipeline is run) named fastqc_summary.txt .

Note: this summary file will capture all _fastqc.zip files in the input directory (even if you run the pipeline on a subset of files).

For more information take a look at parallelising tasks in the Bpipe documentation.

```
Bpipe.run { "%.fastq%" * [run_fastqc] + run_fastqc_parser }
```

CHAPTER 3

Example running the pipeline

To run the pipeline:

```
bpipe run -n 20 -r pipelines/fastqc.groovy *.fastq.gz
```

The `-r` parameter generates a basic report and the `-n` parameter defines the number of threads to use throughout the pipeline.

The output files are saved into the ‘qc’ subdirectory.