THE

SAMSTAT pipeline

MANUAL

Kim Carter

□

TODAY

Contents

Contents ii

1 Introduction 1

2 File: samstat.groovy 2

3 Example running the pipeline 5

CHAPTER 1

Introduction

This Markdown document provides a simple way to do some quality control checks after aligning your sequence data as it goes through high throughput sequencing pipelines. SAMStat is an efficient C program to quickly display statistics of large sequence files from next generation sequencing projects - when applied to SAM/BAM files all statistics are reported for unmapped, poorly and accurately mapped reads separately. This allows for identification of a variety of problems, such as remaining linker and adaptor sequences, causing poor mapping. For more info on SAMStat - 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: samstat.groovy

When the MEdical Sequence Analysis Package (MESAP) is built, this Markdown file (`samstat.md`) is used to create a file called `samstat.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: "SAMStat pipeline."
def BASEROOTDIR="BASEDIRNAME"
```

SAMstat runs as a command-line program. This program is distributed with the MESAP and declared as a variable here.

```
def SAMSTAT = "$BASEROOTDIR" + "/programs/samstat-1.5.1/src/
   samstat"
```

The following routines extracts the filename part out of the full part of any input file (no path, but with file extension); and extract just the file path (with no filename). These are used in the two steps of the pipelines - the routines expects files to be in the format: XXX_XXX_BARCODE_LANE_R.bam (or .sam).

```
def get_sample_filename_nopath_withextension(filename)
{

  def returned_value = ""

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

  return m
}

def get_filepath(filename)
{

  def returned_value = ""
  def i = filename.lastIndexOf("/")
  def m = filename.substring(0,i)

  //return filepath
  return m
}
```

The main (first) step of the pipeline is to run the SAMStat program over each of the input .bam(or .sam) file.

```
run_samstat = {
        output.dir = "$OUTPUTDIR"

    doc "Run Samstat over .bam/.sam alignments"

    def filename = get_sample_filename_nopath_withextension("
       $input1")

    produce ("$input1"+".samstat.html")
    {
        exec "$SAMSTAT $input1"
    }
}
```

The second step of the pipeline creates an overview summary file, across a subset of mapping quality measures in SAMStat. You are presented with numbers (and %s) of reads that maps across various mapping qualities. If you have a large number of alignment files to QC, this output file can be a good first step to narrowing down mapping areas that may be of importance for further examination.

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

    def path = get_filepath("$input1")

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

The code below defines the pipeline and how it should be run - each input .bam (or .sam) is treated separately (there is no paired checking mode). The pipeline generates a HTML file (each named inputfile_samstat.html) containing graphical summaries of the SAMStat output for each sample respectively (note: these are written to directory where the .bam files are located). The pipeline also produces a summary of all of the SAMStat output files contained in the output directory (written to the qc/ subdirectory off of whereever the pipeline is run) named samstat_summary.txt .

Note: this summary file will capture all .bam/.sam 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_samstat] + run_samstat_parser }
```

CHAPTER 3

Example running the pipeline

To run the pipeline:

```
bpipe run -n 20 -r pipelines/samstat.groovy *.bam
```

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

The output HTML files are saved into the directory where the input files are located. The summary file is saved to the 'qc' subdirectory whereever you run the pipeline.