# SmallRNAPipeline Notes

SmallRNAPipeline Notes 1

Overview 1

Installation 2

Configuration File: General Entries 2

Secondary Structure / miRNA prediction 2

Software 2

Data 4

Configuration File: Step specific parameters 5

pigz 5

Adapter Trimming 5

Mapping 6

miRNA counting 7

JSON Pipeline File 8

Header information 8

Step Information 9

## Overview

I have tried to create the pipeline so that it is straightforward to reconfigure and run an analysis with different settings simply by editing a small number of files. It also makes it easier to keep track of which results belong to which analysis

There are three files needed to run an analysis

* A configuration file in YAML format
* A pipeline file in JSON format
* A data file in TAB delimited format

The **configuration** file contains general information that tells the pipeline where to find data and software, and step specific settings that allow the user to customize each analysis

The **pipeline** file allows the user to specify which steps to perform in an analysis. In this way a complete data analysis can be performed, or a subset of steps rerun with different analysis settings

The **data** file contains grouping information about the data; this is needed, for example, when performing differential expression analysis

The current version of the pipeline also requires

* **Trimmomatic** for adapter trimming
* **Bowtie** for mapping reads
* The **Fastx\_toolkit** for fastq to fasta conversion and collapsing reads

Currently, the pipeline will only run on **Linux**/**MacOS** from the command line. It might be possible to run on **Windows**, but I haven’t tried (and I don’t plan to)

## Installation

The program can be run from the command line using something like

java -jar -Dlog4j.configurationFile=file:/data/projects/simonray/software/target/log4j2.xml /data/projects/simonray/software/target/NGSsmallRNA-1.0-SNAPSHOT.jar -r /data/ngsdata/sweden/sweden.small\_ngs.config.yaml -p /data/ngsdata/sweden/sweden.pipe.json -d /data/ngsdata/sweden/sweden.data.tsv

where log4j2.xml provides information about which analysis information is written out and

-r specifies the YAML configuration file

-p specifies the JSON pipeline file

-d specifies the TAB delimited datafile

Note:

The pipeline generates a lot of intermediate files, e.g. trimmed **FASTQ** files, mapped and unmapped **FASTQ**, **SAM** files, all of which take up a lot of space. There is a cleanup step that compresses all of these files. I don’t generally do this unless I have finished analyzing a dataset, which means it is important to keep track of how much disk space a project is using, it is quite easy to fill up your disk, particularly if you are working on your local machine

## Configuration File: General Entries

Some of these entries are not relevant to the standard analysis

### Secondary Structure / miRNA prediction

paths:

installation\_folder: /home/sr/NetBeansProjects/MirPara4j

model:

folder: models

rnafold:

folder: lib

dll: RNAFold.dll

These settings are used for miRNA prediction and secondary structure prediction, you don’t need them for the standard smallRNA pipeline

### Software

specifies the location of third party software

software:

root\_folder: /Users/simonray/software/

unzip: /usr/local/bin/pigz

adapter\_trimming: trimmomatic/Trimmomatic-0.33/trimmomatic-0.33.jar

fastq\_to\_fasta: /usr/local/bin/fastq\_to\_fasta

fastx\_collapser: /usr/local/bin/fastx\_collapser

mapping\_command: /usr/local/bin/bowtie

*root\_folder:*

The location of the parent folder for third party software. However, when writing the software using **NetBeans** under **MacOS**, the **PATH** information is not transferred into the environment, so I have to specify the full path for each program. This may not be a bad thing as it allows a mixture of programs that are installed locally and system wide.

*unzip:*

I use **pigz** since you can specify multiple threads to speed up the compression/decompression. Handy for large data files.

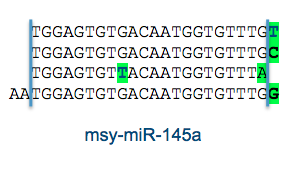
*adapter\_trimming:*

for trimming adapter sequence from reads. I tried different tools and liked this one best, however we can write additional steps using different programs if needed

*fastq\_to\_fasta:*

*fastx\_collapser:*

The pipeline performs its own counting, rather than relying on third party software. This is because it gives us greater control over which reads are included and which ones are rejected. For example,



With the standard counting files, it is unclear which of the reads would be included as mapping to this feature. By performing our own counting, we can specify exactly how many mismatches or which types of 5´/3´ extensions and deletions we keep. **fastq\_to\_fasta** converts *fastq* files to *fasta* format and **fasta\_collapser** counts up identical reads. The output file is a list of reads, with the most frequent at the top of the file, and the least common at the bottom, so

>1-152593

GCATTGGTGGTTCAGTGGTAGAATT

Tells us that the most frequently occurring read was GCATTGGTGGTTCAGTGGTAGAATT and it occurred 152593 times in the original FASTQ file.

The downside of counting reads based on FASTA is that the QC information has been lost. However, when we perform the QC step, we are removing low quality reads from the data (see the Trimmomatic parameters below), so this shouldn’t have a major impact on the counting. When we tried comparing results between FASTA and original FASTQ data, it was very consistent. In any event, this is probably better than blinding using third party tools and being uncertain what is being kept and discarded.

*mapping\_command:*

This is the mapping software used for mapping reads. Currently I use bowtie. It would be good to compare results using other tools, but I haven’t yet written steps to do this.

### Data

Specifies the location of reference datasets. Currently this comprises a reference genome and miRBase GFF files.

data:

folder: data

data\_file: mirbase.dat

genome\_root\_folder: /data/genomes

mirbase\_folder: /data/mirbase

*folder:*

*data\_file:*

These are no longer used. (I think) they can be deleted from the configuration file.

*genome\_root\_folder:*

path to the parent folder containing genome reference data. (because we might want to map to other reference genomes other than human, or may want to use different releases, we need to know this)

*mirbase\_folder:*

Path to parent folder containing miRBase reference data. In the same way, we may want to use other host reference data, or consider other releases. It is important to check the miRBase release corresponds to the genome reference release, otherwise the counting will be messed up. (I don’t check for this in the pipeline)

I prefer to put the reference and NGS data into a folder called /data, but this isn’t possible when running on the TSD or NSC secure server.

For the NSC server, we are assigned a folder with the format /data/projects/[UiOID]/

e.g.

/data/projects/simonray/

so, perhaps we should go with the format

/data/projects/simonray/

|--/genomes

|--/mirbase

|--/ngsdata

From talking to the NSC boys, they recommended keeping our own copies of reference data (i.e., genome and miRBase), but we might want to share the data for a specific project (e.g. one reference genome for RA project, another for LBP, even if they are identical as these represent distinct projects)

I have been using the Ensembl release GrCh37. Because we are only interested in smallRNA, I haven’t expanded the entire release as this takes up more than 30GB. Instead, I have only unzipped the Sequence arm, i.e.

/data/genomes/Homo\_sapiens/Ensembl/GRCh37/Sequence/

This is because we only need the reference sequence for mapping the reads, the annotation information (i.e., the miRNA location) comes from the miRBase GFF file. This is why it is important to ensure the **miRBase** and **Ensembl** release are consistent

You can find the release information at the top of the miRBase GFF file

:fastq\_files simonray$ **head /data/mirbase/20/hsa.gff3**

##gff-version 3

##date 2013-10-1

#

# Chromosomal coordinates of Homo sapiens microRNAs

# microRNAs: miRBase v20

# genome-build-id: GRCh37.p5

# genome-build-accession: NCBI\_Assembly:GCA\_000001405.6

#

# Hairpin precursor sequences have type "miRNA\_primary\_transcript".

# Note, these sequences do not represent the full primary transcript,

chr1 . miRNA\_primary\_transcript 17369 17436 …

it is important to check this, because if we look at the corresponding file for miRBase release 21 we find

:fastq\_files simonray$ **head /data/mirbase/21/hsa.gff3**

# microRNAs: miRBase v21

# genome-build-id: **GRCh38**

this would produce incorrect count information

For now, I am using **GrCh37** and **miRBase release 20**

## Configuration File: Step specific parameters

### pigz

unzip:

no\_of\_threads: 4

*no\_of\_threads:*

How many threads to use for compression/decompression

### Adapter Trimming

adapter\_trimming:

adapter\_file: /data/TruSeqE-SE.fa

no\_of\_mismatches: 2

min\_align\_score: 7

no\_of\_threads: 4

min\_avg\_read\_qual: 30

*adapter\_file:*

full path to the adapter file containing adapter sequence.

I use the adapter file specified by Illumina which they use for their own adapter trimming.

*no\_of\_mismatches:*

How many mismatches can exist when matching the adapter sequence to the read

*min\_align\_score:*

minimum alignment score for a read to be included. This corresponds to the <simple\_clip\_threshold> value specified in the ILLUMINACLIP step within Trimmomatic. I set this to 7 since, from second paragraph on p 7 of the [Trimmomatic V0.32 user manual](http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/TrimmomaticManual_V0.32.pdf)

*The full alignment score is calculated as follows. Each matching base increases the alignment score by 0.6, while each mismatch reduces the alignment score by Q/10. By considering the quality of the base calls, mismatches caused by read errors have less impact. A perfect match of a 12 base sequence will score just over 7, while 25 bases are needed to score 15. As such we recommend values of between 7 - 15 as the threshold value for simple alignment mode.*

*no\_of\_threads:*

How many threads to use when trimming.

*min\_avg\_read\_qual:*

a read must have this average minimum value to be retained (this is why I think it is okay to count reads based on FASTA files)

## Mapping

Parameters used for mapping

bowtie\_mapping:

alignment\_mode: v

no\_of\_mismatches: 2

no\_of\_threads: 4

host: msy

I use bowtie because it is well suited for mapping short reads. It would be interesting to compare other mapping tools to see if it significantly affects the mapping results

alignment\_mode:

This specifies which type of alignment mode to use (see [Bowtie manual](http://bowtie-bio.sourceforge.net/manual.shtml) for details). I use the –v mode as it accepts mismatches, which is what we need if we are investigating isomiRs

no\_of\_mismatches:

specifies how many mismatches we will accept

no\_of\_threads:

how many threads to use for mapping. Better to go as high as possible without locking up the computer

host:

three character code for the reference genome. I use the miRBase abbreviations since it makes it easier to match the two reference datasets. Generally, it seems to be based on the binomial nomenclature with first letter from genus + first two letters from the species. E.g. *Homo sapiens* 🡪 *hsa.* I use this because different reference sources have different paths to the reference sequence

e.g.

/data/genomes/Homo\_sapiens/Ensembl/GRCh37/Sequence/WholeGenomeFasta /genome.fa

points to the genome sequence for *Homo sapiens*, whereas for *Mallasezia sympodalis* it is located in

/data/genomes/Malassezia\_sympodialis\_lab/Sequence/WholeGenomeFasta/genome.fa

we can resolve this by using a link (like shortcuts in Windows) that points to the Sequence folder

So, in the genome root folder (/data/genomes in my case) I have the following

Simons-MacBook-Air:~ simonray$ **ls -all /data/genomes/**

.

..

hsa -> Homo\_sapiens/Ensembl/GRCh37/

msy -> /data/genomes/Malassezia\_sympodialis\_lab/

sce -> /data/genomes/saccharomyces\_cerevisiae

by specifying the three character code, a step can build the path to the reference sequence for that genome.

### miRNA counting

These parameters tells us what to keep and what to report when counting miRNAs

*sam\_mirna\_processing:*

bleed: 2

mirbase\_release: 20

baseline\_percent: 5

analyze\_isomirs: false

*bleed:*

Allow these many extra bases on either side of the defined feature. So, a value of two would allow a read that was either 2 nt longer or shorter than the start and finish position.

*mirbase\_release:*

which version of **miRBase** to use for feature information

*analyze\_isomirs:*

perform an isomiR analysis. i.e, rather than just counting reads, we examine and report the composition.

This will generate a report in the following format

hsa-miR-122-5p|MIMAT0000421 : chr18 56118320 --> 56118341 (+) : UGGAGUGUGACAAUGGUGUUUG

Total Counts = 102243

20-65059 TGGAGTGTGACAATGGTGTTTGT MD:Z:23 65059

89-12922 TGGAGTGTGACAATGGTGTTTGC MD:Z:22T0 12922

229-5771 TGGAGTGTGACAATGGTGTTTA MD:Z:21G0 5771

232-5662 TGGAGTGTGACAATGGTGTTTGG MD:Z:22T0 5662

*baseline\_percent:*

This is only used for isomiR reporting. Anything below this value is assumed to be random variation, so we don’t report these reads. I select 0.05 as the baseline, i.e. any read that is less than 5% of the total number of reads for that feature is not reported. In the example above this would be 102243 \*0.05 = 5112 counts

## JSON Pipeline File

The pipeline file specifies which steps should be run, where the input data is located, and where to write the output.

A typical analysis would be

{"pipelineName":"pipeline1",

"projectID":"sweden",

"projectRoot":"/data/ngsdata/",

"stepsData":

[

{"stepType":"TrimAdapters", "inputFileList":"raw\_data", "outputFileList":"trimmed\_data"},

{"stepType":"CollapseReads", "inputFileList":"trimmed\_data", "outputFileList":"collapsed\_reads"},

{"stepType":"BowtieMapReads", "inputFileList":"collapsed\_reads", "outputFileList":"mappedReads"},

{"stepType":"analyzeStartPositions", "inputFileList":"bowtie\_genome\_mapped", "outputFileList":"analyzeStartPositions"},

{"stepType":"parseSAMForMiRNAs", "inputFileList":"bowtie\_genome\_mapped", "outputFileList":"mappedReads"},

{"stepType":"differentialExpression", "inputFileList":"mirna\_isomir\_analysis", "outputFileList":"differentialExpression"},

{"stepType":"cleanup", "inputFileList":"", "outputFileList":""},

{"stepType":"exit"}

]

}

### Header information

The pipeline file must contain the following three lines.

{"pipelineName":"pipeline1",

"projectID":"sweden",

"projectRoot":"/data/ngsdata/",

pipelineName tells us which pipeline was used to generate the results, so we can go back and see what parameters we used

projectID and projectRoot tells the pipeline where to find the data. In this case, the project data root will be located in /data/ngsdata/sweden

### Step Information

The remainder of the file tells the pipeline which analysis steps to perform

In this case:

TrimAdapters:

CollapseReads:

BowtieMapReads:

AnalyzeStartPositions:

ParseSAMForMiRNAs

DifferentialExpression:

CleanUp

For each step, the following information must be supplied

{

"stepType":"<step\_type>",

"inputFileList":"<input\_folder",

"outputFileList":"<output\_folder>"

},

where <step\_type> is the name of the step, <input\_folder> is the location of the input files, <output\_folder> is where the results will be written. If <output\_folder> doesn’t exist it will be created.

For example, the TrimAdapters step has the following entry

{"stepType":"TrimAdapters",

"inputFileList":"raw\_data", "outputFileList":"trimmed\_data"},

raw\_data is the location of the raw fastq files, trimmed\_data is where the trimmed fastq files will be written.

For the next step, CollapseReads, we have

{"stepType":"CollapseReads", "inputFileList":"trimmed\_data", "outputFileList":"collapsed\_reads"},

Now the trimmed\_data is the input and the collapsed reads will be written to the collapsed\_reads folder. These collapsed reads serve as input to the next step BowtieMapReads.

## Data File

The data file contains grouping information about the sample files. This information is needed for statistical comparisons such as Differential Expression analysis

The format of the file is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File | Source | Condition | Time | Note |
| 2726\_S1.fastq | Msy | ph5.5 | t0 | Msy |
| 2726\_S3.fastq | Msy | ph5.5 | t0 | Msy |
| 2726\_S4.fastq | Msy | ph6.1 | t0 | Msy |
| 2726\_S5.fastq | Msy | ph6.1 | t0 | Msy |

Where

|  |  |
| --- | --- |
| File: | The name of the raw unprocessed fastq file |
| Source: | The host name. This isn’t currently used, but would allow us to perform comparative analysis between two different hosts under similar conditions. |
| Condition: | This is what is currently used to group the data when performing differential expression |
| Time: | Time point of the data collection. This is for time series analysis, but is not currently used |
| Note: | Additional information that can make the file more readable and easy to understand |