YAP - A General Purpose Pipeline for the analysis of Next Generation Sequencing Data

Vibhas Aravamuthan, Ph.D.
Tripti Kulkarni
Varun Shivashankar
Robin Ge
Stephen Litster, Ph.D.
Michael Derby
Michael Steeves

Novartis Institute for Biomedical Research
Cambridge, MA, USA

Version 2.3

September 26, 2013

Contents

List of Figures

List of Tables

What's New in YAP 2.3

1. New YAP check script that checks for syntactical errors and checks I/O dependencies.
2. YAP error checking mechanism indicates which command failed at which stage along with the standard error message.
3. Filters all the failed samples from further processing. (After alignment only)
4. Provides the user with a quick stage-wise, sample-wise PASS/FAIL matrix.
5. GATK compatibility.
6. NIBR genome index files compatibility.
7. Ability to get raw counts (HTSeq) and normalized counts (Cufflinks) in the same YAP workflow.
8. Summary across all the samples for Targeted Pcr Metrics and Calculate HS Metrics.
9. Email job update in SGE if mail id is provided.
10. New shortened configuration files for easier and lighter usage.

Release Summary for YAP 2.3

1. Package updates

    - Pre-processing tools :
        - FastQC (v0.10.1)
        - Fastq screen (v0.4.1) - handles compressed files
        - YAP base count metrics
        - FASTX-Tool kit (0.0.13)
            - FASTQ/A Barcode Splitter
            - FASTQ/A Clipper
            - FASTQ Quality Filter
            - FASTQ Quality Trimmer
    - Aligner algorithms :
        - Burrows-Wheeler Aligner (BWA) (0.7.4)
        - Bowtie aligner (1.0.0)
        - Tophat (2.0.8b)
        - Bowtie2 aligner (2.1.0)
    - Post-processing toolkit :
        - Picard (1.89)
            - Multiple Metrics
            - Alignment Metrics
            - Quality Score Distribution
            - Mean Quality by Cycle
            - GC Bias Metrics
            - Insert Size Metrics
            - RNA Seq Metrics
            - Mark Duplicates
            - Calculate Hs Metrics
            - Targeted PCR Metrics
        - HTSeq (0.5.3p9) - bowtie2 compatible
        - Cufflinks (2.1.1)
            - Cuffdiff
            - Cuffmerge

        * Cuffcompare
    - Exon counts
    - Junction counts
    - Genome Analysis Toolkit

2. New YAP check script that checks to see if all the packages are executable, accesible. Also checks to see if the paths exist or not. Checks for compatibility between filenames

3. New Error logging mechanism informs captures the standard output messages and error messages of every command and logs them.

4. Provides a sample-wise, stage-wise PASS/FAIL matrix for a yap run based on the error log files.

5. Automatically filters out failed samples from further processing.

6. Includes an option to regroup sample replicates and merge them afte the alignment stage.

7. GATK Compatibility.

8. Ability to perform dual sorting by queryname and/or coordinates on alignment files.

9. Support for reference files for all aligners based on the UCSC reference genome (02 May, 2013).

10. Compatibility with NIBR genome index files.

11. Ability to obtain normalized and raw counts in a single workflow.

12. Ability to run multiple aligners in a single workflow.

13. Support for accepting input from multiple directories.

14. Inclusion of tee mechanism in postprocess to specify a single type of input for multiple packages.

15. Generates consolidated output file facilitating an effecient subsequent YAP run with the alignment files as input.

16. Ability to summarize output results in a single pdf file.

17. Rudimentry configuration file syntax checker.

18. Provides a consolidated summary for:

- FastQC
- FastqScreen
- HtSeq
- Picard tools
  - RnaSeqMetrics
  - MarkDuplicates
  - InsertSizeMetrics
  - AlignmentSummaryMetrics
  - QSdistribution
  - QScycle
- Cufflinks
- Exon counts
- Junction counts
- Unknown junctions

19. Ability to capture the details of workflow process.

Chapter 1

Introduction

The Next Generation Sequencing (NGS) data is exponentially growing, with different groups using different software for analyzing NGS reads. A detailed analysis revealed that they share a number of common features. Almost all of these pipelines follow a standard analysis workflow. These are:

- Read the input data in fasta, fastq, qseq or tab limited formats.
- Do some pre-processing like, removing bad reads, base counting, remove adaptor and barcodes etc.
- Align the reads to a reference genome.
- Depending on the analysis, do some post-processing on the aligned reads, to get results meaningful for a particular analysis.

The facts mentioned above were leveraged in developing a single, scalable and flexible pipeline which can be optimized to perform in a parallel environment, which we call YAP $YetAnotherPipeline$.

1.1 What is YAP ?

YAP is a framework which allows researchers to put together an analysis pipeline quickly and efficiently. YAP by itself does not do any analysis of NGS data. It provides a mechanism for developing work flows where each stage of the work flow is handled by a different external package. YAP then executes these work flows in parallel by splitting the input data into chunks and assigning them to a different processor or core. All the necessary work to accomplish tasks like reading data, splitting the data, distributing the data to processors, synchronization between different processors, etc. is taken care of by YAP. The user then interacts with YAP by providing configuration files for the different stages in the pipeline.

Setting up a configuration files involves the user following a few simple rules and adhering to a defined syntax.

1.2 Features of YAP

Notable features of YAP include:

- General purpose pipeline for NGS data processing.
- Architecture and algorithm independent.
- Maximal amount of work is done when a sequence already resides in memory.
- Minimal amount of temporary files generated.
- Modular design facilitates incorporation of different analysis packages or codes as they become available.
- Minimize file reads and writes, optimize disk I/O.
- Maximum flexibility and customization based upon configuration files.
- Support for common file formats- qseq, FASTA, FASTQ, SAM, BAM.
- Support for direct manipulation of gzip and bzip2 compressed files.
- Centralized reference database.
- Parallelized using Message Passing Interface(MPI).
- Efficient utilization of available cluster nodes to minimize run times.

1.3 Organization of this Report

The report is organized into a number of chapters and sections. Chapter 1 introduces YAP and its general features. Chapter 2 details the design principle and algorithms used in YAP. Chapter 3 gives the implementation details including details regarding YAP configuration files. Setting up and using YAP is detailed in Chapter 4. The Appendices give details on file formats, configuration files, examples of a microRNA and RNAseq analysis and some graphical examples of YAP results.

Chapter 2

Design Principle

Next Generation Sequencing (NGS) is an emerging technology and new software packages are being developed by a large group of researchers. The technology and the analysis methods are changing rapidly and it is becoming difficult to develop a general pipeline to analyze a large class of NGS problems. This is compounded by the fact that individual researchers want to use different software packages to do their analyses. This has resulted in ”pipeline proliferation” with individual researchers developing their own pipelines. Most of these pipelines share the following characteristics:

- Narrow in focus and tend to address a particular problem the researcher happened to be interested in.
- Heavily I/O intensive. These pipelines write and read from files at every stage thus making them very inefficient.
- Single threaded. These typically do not use the multi core systems and clusters that are available.
- No standardized output handling.
- Difficult to integrate packages written in multiple programming languages. The philosophy of YAP is to leverage existing NGS software packages

2.1 Design Overview

A broad overview of the underlying philosophy behind YAP is given below.

- Design a general purpose pipeline which is scalable and flexible and can be adapted easily for varied workflows.

- Handle compressed input files directly without decompressing them and in a parallel fashion.

- Design it from the bottom up with parallelism and efficiency in mind.

- As far as possible minimize I/O by doing much of the work once the sequences are in memory.

- Algorithm and programming language agnostic. Ability to incorporate any external package or software written in any programming language into the pipeline with minimal effort. Researchers are free to use any pre-processing, aligner or post-processing package.

- Standardized output directory structure.

YAP is a general purpose framework. It can be adapted for a particular work flow by choosing underlying packages used at the different stages.

YAP has been designed to be scalable and flexible. The flexibility comes from the fact that users are free to choose any software package they want as long as the package allows reading data from standard input and can write to standard output. The various stages of the pipeline are chained together using 'pipes'. All aspects of data transfer are handled in memory without resorting to disk writes/reads.

The package also handles compressed data transparently and efficiently. The input files are typically ASCII files in qseq, fasta, fastq, tab-separated formats. These files can be stored in a compressed fashion. Although YAP supports both gzip and bzip2 compression, it is recommended that files be compressed using gzip, as it is faster to read from a gzip file than a bzip2 file. This is due to the fact that YAP uses file seeks to get the byte position in compressed files. This works in an emulation mode in bzip2 compressed files whereas in gzip this is implemented natively. The reasons for this lie in the type of compression algorithm used by bzip2. For more details the user is referred to the gzip and bzip2 reference manual.

The scalability in YAP is achieved using data parallel paradigms. The input data is split into a number of chunks, where the chunk size in megabytes is user defined. The chunks are then distributed evenly to a number of processors which then process them in parallel. Synchronization points are identified and is taken care of automatically by the package. This has a two fold purpose. By splitting the data into smaller chunks it is possible to run it on hardware with small amounts of RAM. This is especially important as the design philosophy of YAP is to do as much work as possible once the sequences are read into memory. This is not a problem for small amounts of data (few hundred megabytes). These data sets can be run on small desktop machines with modest amounts of RAM

(4GB). When the problem size is increased the operating system resorts to swapping to disk resulting in massive performance degradation. This situation can be alleviated by resorting to processing the data in small chunks. The chunk size can be chosen in such a way that no swapping takes place. This thus results in the optimum use of the available hardware.

The other advantage of chunking up the data is that data parallel techniques can be used to process these chunks in parallel. This is especially important on modern hardware which mostly consist of multi-core processors. It is very common to find desktop systems with four and six cores. YAP also has the ability to perform file based data distribution where each file is processed, as a whole, on a node. The user hence has the flexibility to choose between the two types of data distribution methods. The choice depends on the size of input data, resource availability and the nature of the package being used inside YAP. A data parallel model using either Symmetric Multi Processing (SMP), or Distributed Computing is necessary in order to exploit the available hardware.

In order to exploit the distributed computing clusters that are available, it was decided to use a distributed computing model to develop the application. This was implemented using the Message Passing Interface (MPI).

YAP also has the ability to perform file based parallelism

2.2 Algorithm

The algorithm behind YAP is pretty straight forward. The entire work flow is divided into four broad stages as shown in Figure (2.1). These include:

1. Read input data. Data can be in any one of fasta, fastq, qseq, and tab separated formats. The data sets are typically compressed using gzip.

2. Carry out some pre-processing operations on the input data. These pre-processing steps include:

   - Remove adaptors if present.
   - Segregate data based on barcodes for small RNA.

3. Alignment of the reads generated in the previous step. The reads are aligned to a reference genome using any one of the many alignment algorithms available. The user is free to use any algorithm or package he/she wishes or if needed can develop their own and use it.

4. The aligned read data in SAM format are then post-processed using any user defined post processing package. In this step summary statistics like gene counts, alignment summary metrics, GC Bias metrics, insert size metrics, rna seq metrics, cuffdiff etc. can be calculated.

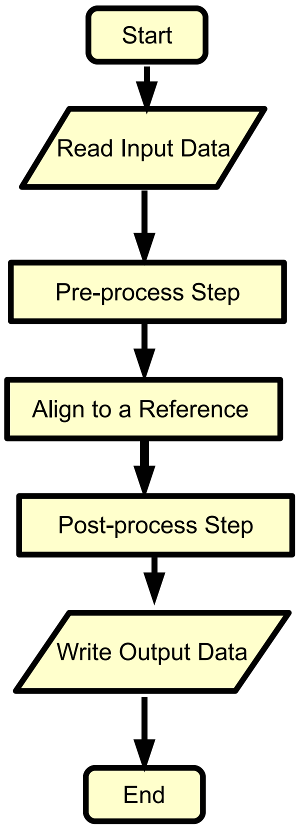

Figure 2.1: Main Stages in YAP work flow

Data flows from one stage to the next after undergoing certain transformations. The algorithm is described in detail in the following sections. A detailed flowchart of the algorithm is shown in Figure(2.2).

2.2.1 Read Input Data

The raw data produced by the instruments suitable for analysis are typically in qseq, fasta, fastq or tab separated formats [Appendix A] There are typically hundreds of these files in an ASCII format. The file sizes vary from a few hundred Megabytes to several Gigabytes.

In order to save disk space these files are compressed using standard utilities like gzip or bzip2. This can result in file sizes being reduced by a factor of three to ten depending on the contents of the file. A naive way of using these files would be decompress them into a temporary location and read the uncompressed files. The problem with this approach is when analyzing hundreds of Gigabytes of data one needs lots of temporary disk space. Another drawback is the amount of time spent decompressing these data sets.This can be considerable if the original uncompressed size of these data sets are large. Typically a 2.1 GB compressed file of fastq sequences takes about 10 mins to decompress to about 9 GB. YAP has the ability to read files in gzip or bzip2 compressed formats. The module that accomplishes this is written in Python. Python through the use of the gzip module has the ability to seek a compressed file. The algorithm to do this is done in two stages. In the first stage the size of the original uncompressed file is determined. This is necessary because gzip does not store the original file size. In the second stage the beginning and ending byte positions corresponding to the given chunk size is calculated (Since YAP 2.1.1 this section has been parallelized). For load balancing purposes the user defined chunk size is modified based on the number of processors given at run time. The chunk sizes are modified so that each processor gets the same number of chunks. In the second stage of the algorithm the chunk size is readjusted so that they fall at the boundaries of a complete fasta or fastq read. The processes that occur at this stage are summarized below.

1. Do the following on the master or root processor (Processor with rank$\bar{0}$ in MPI parlance).

2. Read in the user defined chunk size in Megabytes .

3. Find out the compressed file size in a parallel fashion.

4. Set the number of chunks to be an integral multiple of the number of processors.

5. Number of lines in a chunk is determined and adjusted to preserve the integrity of the data.Then the begining and ending byte position of the chunk is saved.This step is repeated till the file is read into specified number of chunks.

6. Each processor is assigned a chunk for computation.

2.2.2 Data Pre-processing

The data read in previous stage is now sent through a pre-processing step. The preprocessing packages already included in YAP are listed below.

1. Quality control checks on the raw sequence data is performed using FastQC. Following are the important functions of FastQC.
    - A quick overview of the quality of data using summary graphs and tables.It helps in taking appropriate decisions for further analysis.
    - Export of results in HTML based format.
2. FastQ Screen allows you to screen the raw input data against a set of sequence databases which allows you to see your input sequence composition.
    - The input data is screened against sequence libraries of organisms such as Human, Mouse, E.coli etc and Vectors, PhiX and other contaminants.
3. Preprocessing of read sequences using FASTX-Toolkit.Following pre-processing tools from toolkit are included in yap.
    - FastQC -
    - FastqScreen -
    - FASTQ/A Clipper - to remove sequencing adapters/linkers. (A default file with adapters is used unless user provides a separate file check **??**file_formatbarcode_file_formatreferred style of output can be generated by using various command line options/arguments. Like, non-clipped or clipped sequences can be discarded,Adaptor-only reads can be saved and etc
    - FASTQ/A Read based Barcode splitter - to split a small RNA(FASTQ/FASTA) file containing multiple samples.
    - FASTX Filter - Sequences are filtered based on the quality.
    - FASTX Trimmer -Sequences are trimmed based on quality.

The preprocess packages are executed in parallel with each node working on its own set of chunk or input data. Within each node multiple processes are spwan allowing the user to fully leverage multiple processors of a node or machine.Preprocessing (FastX Clipper/small RNA Barcode splitter, FastQ Filter and Trimmer) of paired-end data may generate different number of sequences in each pair.Hence,YAP algorithm retains only those reads which are common in both pairs(files) and discards the rest to avoid inaccurate results in further analysis. In each node multiple processed chunks are merged to a single chunk and sent to alignment step.

2.2.3 Alignment to a Reference

The data from the previous step is aligned to a reference genome. Any aligner can be used if the user has prepared a valid configuration file that YAP understands. The only requirement is that the aligner can read fasta, fastq, qseq or tab separated input from the standard input and write the output in SAM format to standard output. The transfer of data from the pre-processing step to the alignment step takes place in memory through pipes implemented using Python's sub-process module. Alignment speed is enhanced by enabling inbuild multithreading option of the aligner. Every processor outputs a file in SAM/BAM format. If the user opts for sorting of the alignment output, the SAM file is split into multiple chunks and sorted based on "coordinates" or "query name" or "both" in parallel. In YAP(after version 2.2), the coordinate and queryname sorting can be done simultaneously in a single YAP run. Using the python implementation of the UNIX 'tee' mechanism the alignment files are read once and a dual sort is performed in parallel hence gaining a higher level of efficiency. This feature allows for running postprocess packages that require alignment files sorted in different orders.
All the processes across nodes are synchronised at this point and sorted chunks are merged to a single file. The aligner packages currently available in YAP are Bowtie, Bowtie2, BWA and Tophat.

2.2.4 Post-processing

All files from the previous step are evenly distributed across all available nodes for post processing of the data in parallel.The input for this step consists of aligned reads in SAM/BAM format. A python implentation of the UNIX 'tee' command is used to read the files once and pipe them to a number of post-processing steps. This thus avoids the expense of reading the data multiple times. If there are multiple classes of alignment files, such as bed files or coordinate and/or queryname sorted SAM/BAM files, then this can be specified under each package using simple wildcard expressions.
The operating system then spawns multiple processes to process the input. This effectively utilizes the multiple cores that are available on a cluster node. The post-process commands included in YAP are Htseq and Picard PostalignmentQC commands (Insertsize metrics, RNA Metrics, GC Bias Metrics, RNA seq metrics etc.), Cufflinks, Cuffdiff, Cuffmerge, Cuffcompare and YAP's Exon and Junction count methodologies which count the exons and junctions with reference to a UCSC coordinate file for gene model. The user can also insert their own coordinate and coordinate number file please refer to Appendix From YAP 2.2 onwards the user can run independent postprocess packages that require different sort orders symultaneously. For eg. HtSeq, which requires queryname sorted files, can be run

in the same workflow with any of the Picard tools, which requires a coordinate sorted file, in the same YAP workflow.

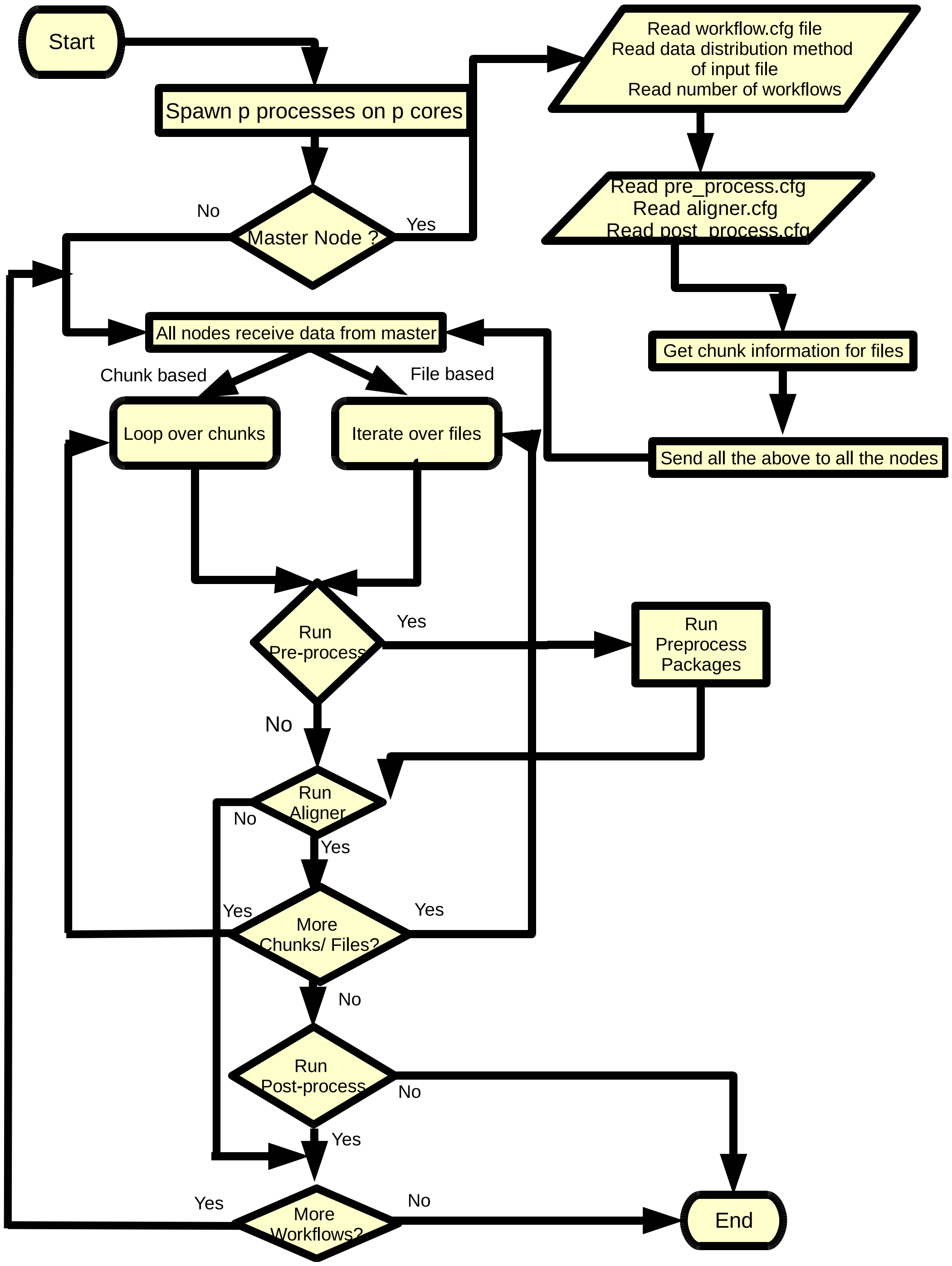

Figure 2.2: Flowchart of the Algorithm

Chapter 3

Implementation Details

YAP is implemented in the Python. The advantages of using Python are many fold. Some of these are listed below:

- Python is a general purpose mature scripting language that is easy to use and understand.
- Large user base, and hence community support is excellent.
- Excellent integration to the underlying system libraries and tools.
- Vast array of libraries and modules available for string processing and manipulation.
- Excels as a ”glue” language. It is very easy to write wrappers for functions and subroutines written in other languages.
- A solid and mature wrapper to the Message Passing Interface (MPI) libraries. Almost all of MPI 2 is available within Python.

As mentioned earlier the philosophy of YAP is to leverage existing NGS software packages. This is accomplished by making extensive use of the ”subprocess” module in Python. The subprocess module enables one to implement Unix like ”pipe” in Python. This in turn enables us to pass data from one stage to another without resorting file based I/O. It also enables us to spawn external processes and control them within Python. A detail implementation of YAP algorithm is shown in flowchart Figure (3.1)

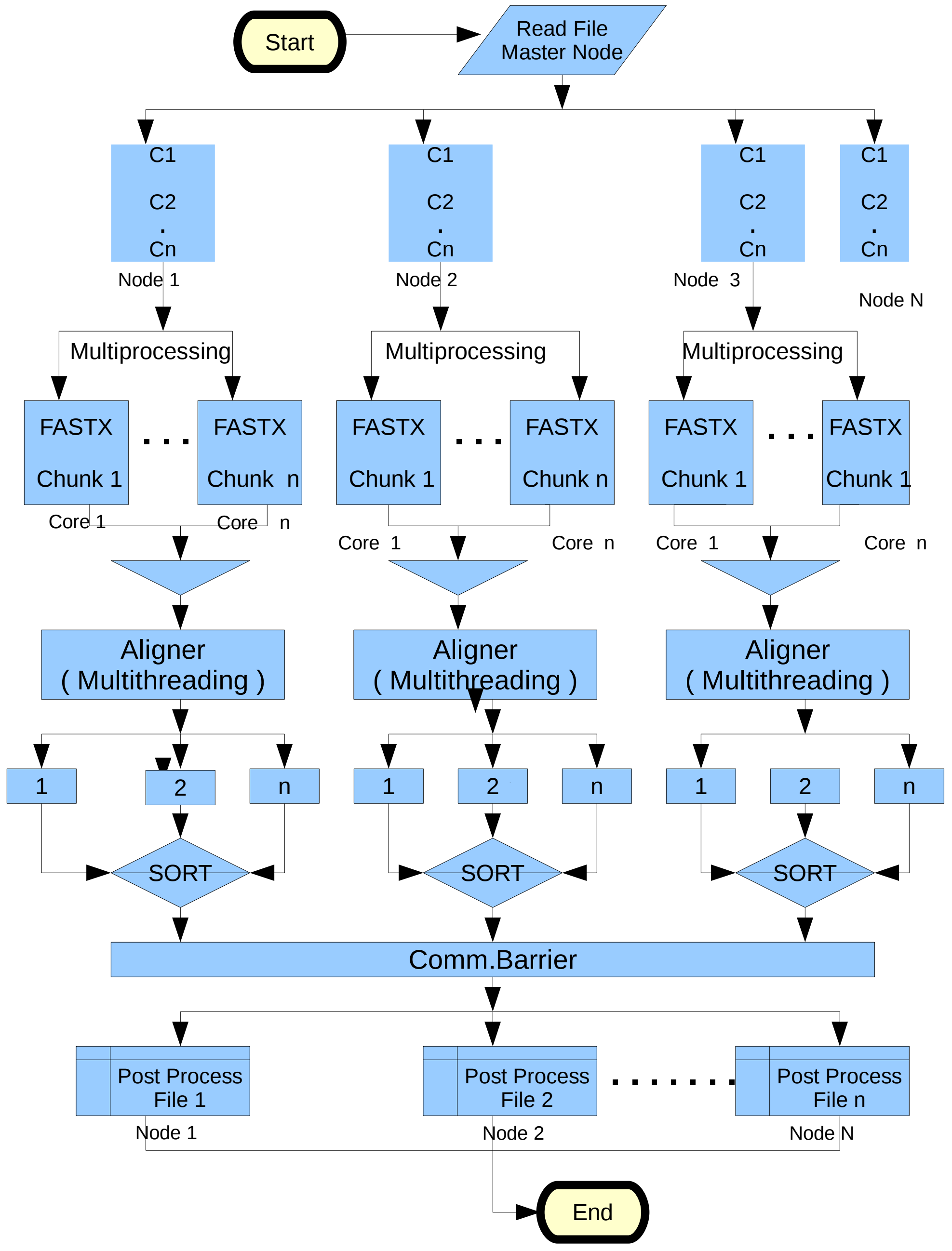

Figure 3.1: Flowchart of YAP algorithm implementaion - Chunk Based

The details about the options used at the different stages of the pipeline is achieved using configuration files. The following sections will explain what a YAP configuration file is and tells the user how to create one for a custom command. The YAP configuration files can be downloaded from the github repo. In YAP 2.3 we have 4 sets of configuration files depending on the type of workflow the user wants to run.

- demo - to run a simple YAP run to understand how YAP works. It is preset to run
- standard - modifiable YAP configuration.
- GATK - To run Broad's Genome Analysis Toolkit

3.1 Configuration Files Used by YAP

The YAP configuration files are used to steer the workflow at different stages of the pipeline. The user interacts with the pipeline by giving the desired options in these files. The pipeline consists of four main configuration files. The purpose and description of these files are shown in Table (3.1).

Table 3.1: Configuration files used in YAP.

Configuration File Name	**Purpose**	**Description**
workflow_configuration.cfg	Set up analysis workflows	Contains workflow definitions
preprocess_configuration.cfg	Set up data preprocessing steps	Contains data preprocessing step parameters
aligner_configuration.cfg	Set up the options for the aligner	Contains aligner file paths and command options
postprocess_configuration.cfg	Define the data post-processing steps	Contains Postprocess executables and command parameters

3.2 Overview of a YAP Configuration file

All YAP configuration files adhere to the following standards:

- All lines starting with the Hash # are treated as comments.
- Configuration files take the variables/options and their corresponding values in a key-value format.
- All variables and values should be enclosed within double quotes(eg : "variable").

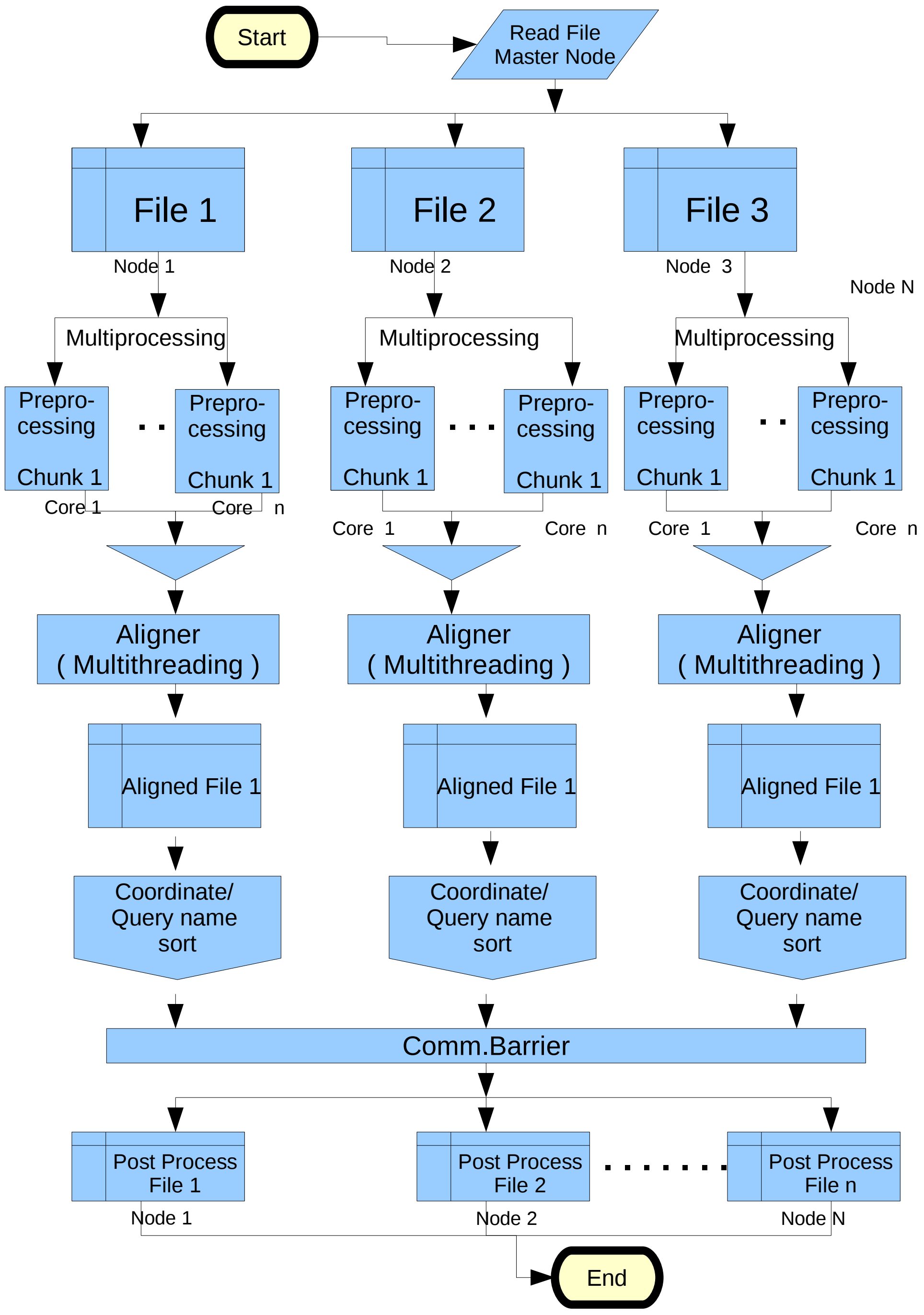

Figure 3.2: Flowchart of YAP algorithm implementaion - File Based

- The variable and its value should be separated by a colon equal-to sign(:=)
  Eg. ***"variable" := "value"***

- Configuration files may also contain file paths. eg. "path" := "/mypath/directory/-file1"

- Multiple workflows and multiple commands can be represented using a combination of :begin and :end.
  For example:

```
:begin    # beginning of first command
          "execute_command" := "yes"
          "command_name" := "command1"
          "variable1" := "5"
:end      #end of first command

:begin    # beginning of second command
          "execute_command" := "yes"
          "command_name" := "command2"
          "variable2" := "5"
:end      #end of second command
```

- For workflow configuration and preprocess configurations - the variable and corresponding value pairs are stored in a data structure and used in the analysis.

- For aligner configuration file and postprocess configuration file, the variable and its corresponding value is used to generate the commands. The values hold special meaning as described below:

- The value can be either "yes", "no" or some value appropriate for the command.

- If the value is "yes" only the variable (left portion of the := symbol) is included as part of the command.

- If the value is "no" the variable is not included as part of the command (in this case the default value is picked up by the executable).

- If the value is other than "yes" or "no", both the variable and the value are used in generating the commands.

- In case a user wants to specify a path/file corresponding to that particular YAP run as input or output, he/she can use a set of metacharacter variables:

Table 3.2: YAP Metacharacter variables

Parameter	Description
"output_directory"	Points to the YAP output directory. Useful for commands that require an output directory to write their own files. Eg. Tophat
"output_file"	Points to the YAP output directory and adds a prefix of sample name and barcode. Useful for commands that require an output file or prefix to be predetermined. Eg. Cuffmerge
"aligner_output"	Points to the aligner output directory of the respective sample. Useful for running postprocess commands that require sam, bam or bed files as input. Eg. Cufflinks,cuffdiff,picard
"postprocess_output"	Points to the postprocess output directory of a respective sample. Useful for running postprocess commands that require previously run postprocess output as input. Eg. Cuffmerge, Cuffcompare require the transcripts.gtf from cufflinks

- From YAP 2.2 onwards, the postprocess configuration file has a ":begin_tee..:end_tee" section in which a set of commands specified within this section will be provided with the same input type by the user.

3.3 Writing YAP Configuration Files

As mentioned in the previous section YAP uses configuration files to steer the workflow. In this section we illustrate this through some examples.

Let us assume that we need to incorporate a package in YAP. Let us say the command to invoke the package is called ***foo*** and it has a number of command line options. The options for the command are:

foo	-options	index_file_name
	-h	*Prints help*
	-n	An Integer Value
	-j	Parameter to invoke an optional functionality
	-k	Another parameter to invoke an optional functionality
	-i	Input file name
	-o	Output file name

Now assume we need the command line to look like:

foo -n 2 -j -i my_input.dat -o my_output.dat Index_File.idx

The corresponding entry in the configuration file would look like:

```
:begin
    "command_name"      :=    "foo"
    "execute_command"   :=    "yes"
    "-h"                :=    "no"
    "-n"                :=    "2"
    "-j"                :=    "yes"
    "-k"                :=    "no"
    "-i"                :=    "my_input.dat"
    "-o"                :=    "my_output.dat"
    " "                 :=    "Index_File.idx"
:end
```

3.3.0.1 Writing to the YAP output folder

In case an option of a command produces an output file which needs to be written in the appropriate YAP output folder please specify the yap variable ***"output_file"*** as a prefix.

for example:

```
:begin
    "command_name"      :=    "foo"
    "execute_command"   :=    "yes"
    "-h"                :=    "no"
    "-n"                :=    "2"
    "-j"                :=    "yes"
    "-k"                :=    "no"
    "-i"                :=    "my_input.dat"
    "-o"                :=    "output_file_my_output.dat" writes output to appropriate
yap directory
    " "                 :=    "Index_File.idx"
:end
```

Consequently YAP can also access files written to the YAP output folder through this variable. For example,

```
:begin
    "command_name"       :=    "foo1"
    "execute_command"    :=    "yes"
    "-h"                 :=    "no"
    "-n"                 :=    "2"
    "-j"                 :=    "yes"
    "-k"                 :=    "no"
    "-i"                 :=    "output_file.dat" gets the .dat files from the appropriate
folders using yap's output hierarchy
    "-o"                 :=    "my_output.dat"
    " "                  :=    "Index_File.idx"
:end
```

3.4 Configuration Files Description

The four configuration files used in YAP enables the user to set appropriate options pertaining to a particular workflow. Using this mechanism the user can for example carry out a RNAseq, microRNA, chipSEQ etc. analysis. The user would need to modify or create configuration files relevant to that particular analysis. A detailed description of the configuration files used in the different stages of YAP is given in the following sections.

3.4.1 Workflow Configuration File

As mentioned in Section (3.1) The Workflow configuration file is the primary configuration file used to define all the workflows, and within each workflow to define data control steps. The configuration files for pre-process, aligner and post-process, for each workflow are prescribed here. This is the main configuration file used to initiate YAP. An example of a workflow configuration file is given below.

In the workflow configuration file the beginning sections before the first :begin are reserved for general metadata pertaining to the the analysis and is applicable for all the workflows eg. general comments, analyst name and organization etc. An example of a workflow configuration file with two workflows is shown in Appendix B.1

```
# YAP main Config ##
```

```
#All Variables in small Case
# All values and paths in quotes
## YAP Analysis run general metadata
"comment":="120 chars"
"analyst_name":="120 chars"
"organisation_name" :=""

:begin
"instrument_type" := "Illumina"
"specimen_info" := "[tissue type]"
"seq_type" := "[DNA or RNA]"
"workflow_type" := "rnaseq or microna"
"workflow_name" := "yap_2.2_test"
# input information
"input_files_path" := "path_to_sample"
"input_files" := "human_sample1*.fq,human_sample2*.fq;human_sample3*.fq"
"input_file_format" := "fastq"
"paired_end_data" := "yes"
"max_read_length" := "100"
"file_chunk_size" := "1024"
"data_distribution_method" := "chunk_based"
# output options
"output_files_path" := "new_yap_2.2_test"
"preprocess_output_file_format" := "fastq"
# preprocess options
"run_preprocess_analysis" := "yes"
"preprocess_configuration_file" := "preprocess_configuration.cfg"
"write_preprocessed_data" := "no"
# aligner
"run_reference_alignment" := "yes"
"aligner_configuration_file" := "tophat_2.0.8b_bowtie2_configuration.cfg"

"alignment_sort_order" := "both"
"merge_alignment_output" := "yes"
# post process
"run_postprocess_analysis" := "yes"
```

```
"postprocess_configuration_file" := "postprocess_configuration_2.2.cfg"
:end
```

A detailed description of the various options available in the workflow configuration file is given in Table(3.3).

3.4.2 Pre-process Configuration File

The pre-process configuration file is used to set up the parameters for the pre-processing stage of YAP. Parameters pertaining to filtering bad reads, adaptor removal, splitting data based on bar codes, etc. are specified. A sample command in the pre-processing configuration file is shown below.

```
:begin
"execute_command" := "yes"
"command_name" := "fastqc"
"--outdir" := "output_directory"
"--casava" := "no"

"--extract" := "yes"
"--java" := "no"
"--noextract" := "no"
"--nogroup" := "no"
"--format" := "no"
"--threads" := "12"
"--contaminants" := "no"
"--kmers " := "no"
"--quiet" := "no"
:end

:begin
"execute_command" := "yes"
"command_name" := "fastq_screen"
"--subset" := "500000"
"--paired" := "yes"
"--outdir" := "output_directory"
"--illumina1_3" :="no"
"--quiet"    := "no"
```

```
"--version"  := "no"
"--threads"  := "no"
"--conf"     := "fastq_screen_v0.3.1.conf"
"--color"    := "no"
"--bowtie"   := "no"
"--nohits  " := "no"
"--nohits  " := "no"
:end
```

3.4.2.1 Read based Barcode File Format (for small RNA)

The barcode file enables YAP to assign some meaningful names to the barcodes present in a sequence. Please refer to A.2

3.4.3 Aligner Configuration File

The aligner configuration file is responsible for the aligner commands used for aligning the reads to a reference genome. The general rules outlined in Sections (3.2) and (3.3) are applicable. Multiple commands can be accommodated by specifying the command parameters in between ":begin" and ":end" sections.

A condensed version of the configuration file for a particular aligner "bowtie" is shown below. For the full version the user is referred to Appendix B.3.

```
:begin
  "command_name" :="bowtie"
  " " :="bowtieIndex"
  "-f" := "yes"
# query input files are (multi-)FASTA .fa/.mfa
  "-v" := "0"
# report end-to-end hits with v less than or equal to the
        # prescribed number of mismatches;ignore qualities
  " -n"  := "0"
        # Integer max mismatches in seed (can be 0-3, default:  2)
  "-a" := "yes"
        # report all alignments per read (much slower than low -k)
  "-m" := "no"
        # suppress all alignments if > <int> exist (def: no limit)
  "--best" := "yes"
        # hits guaranteed best stratum; ties broken by quality"
  "--strata" := "yes"
        # hits in sub-optimal strata aren't reported
        # (requires--best)
  "-p" := "4"
        # Integer number of alignment threads to launch (default: 1)"
  "-s" := "no"
        # prints the summery file
  "-1" := "pipe1"
       #  YAP takes input as pipe . pipe1 and pipe2 specify the paired
       #end reads. dont change the value(pipe1, pipe2)
  "-2" := "pipe2" # if input data is paired end.
  #" " := "pipe1"
  # provide pipe1 for single end read input .dont change the value(pipe1)
:end
```

The command YAP executes internally would then look like

bowtie bowtieIndex -f -v 0 n 0 -a –best –strata p 4 -

Note: If the input data is paired-end then "pipe1" and "pipe2" correspond to first and second input stream, respectively. This allows YAP to pipe data from the previous pre-processing step to the aligner step without resorting to disk I/O.[The python subprocess module is used in YAP to accomplish this].For single end data "pipe1" represents the input pipe.For bowtie single end-alignment input stream can be represented using "-" or "pipe1"

Table 3.3: Parameters in workflow configuration file .

Parameter	**Description**
"comment"	Any comments related to experiment or analysis should be mentioned here.
"analyst_name"	Enter Analyst name , 521 recommended
"organisation_name"	Name of the organization you belong to.
"instrument_type"	Instrument type used for experiments
"specimen_info"	Information about the specimen from which the samples are obtained.
"seq_type"	The type of sequences input files contain. eg. DNA, RNA, MicroRNA etc.
"workflow_type"	Type of analysis being done. eg. "rnaseq or microrna"
"workflow_name"	Workflow name can be anything eg. "rnaseq_demo". This is used to create output directory structure name for your analysis run. All the output directories will be created under this directory by workflow_name. If the workflow_name directory exists then workflow_name would be appended with the date and time and a new output directory structure is created.
"input_files_path"	Enter the full path of the directory containing input data file(s). The user can enter multiple input file paths separated by a ";" For eg.. /home/user/inputdir1;/home/user/inputdir1
"input_files"	Name of the input files . Multiple files can be given as input using wild cards or comma separated file names. Wildcards should be used carefully to assure the correct input. From YAP 2.2, the user input files from multiple input paths separated by a ";". In such a case every input file set should be separated by a ";" and correspond to its specific input path. For eg. input1_1.fq, input1_2.fq needs to be read from path1 and input2_1.fq and input2_2.fq from path2. "input_files" := "input1*.fq;input2*.fq" would work for "input_files_path" := "path1;path2" All the input files should be of same format.
"regroup_output"	If regroup is to be set to yes or no.
"regroup_file"	Specify the name of the file containing the regroup information. See the next chapter for more details
"input_file_format"	Specify the data format used to represent the sequencing data. Supported formats : fasta , fastq, qseq , tab For tab format: The file should have two fields , sequence id and sequence separated by tab.
"paired-end_data"	Type of the raw sequence data. For "single-end" data specify "no" and "paired-end data" specify "yes". **If the user specifies yes then the input files have _1,_2 as suffixes for the respective pairs after the samples names and before the extensions. For example_1.fq,sample_2.fq.**

”data_distribution_method”	YAP offers the option of how the data should be distributed across the nodes. If set to ”file_based”, YAP distributes the files as a whole across the nodes. If set to ”chunk_based” YAP splits the input data into chunks and then distributed across the available nodes. This option is offered inorder to include packages such as tophat, which requires whole files, in the pipeline. ”file_chunk_size” will appropriately by turned on/off depending on what you set here
”output_files_path”	Enter the pathname where the output results can be stored. The User must have write permission to this directory. Under this path structure the output directory with above mentioned run_name would be created.
”preprocess_output_file_format”	Output file format for pre-processed data. Supported formats: fasta,fastq, tab format. The pre-processed data can be represented in all these formats irrespective of the input file format.
”run_preprocess_analysis”	Run the data through Preprocessing steps eg. barcode splitting , filtering adaptors and low complexity sequences,basecount analysis (these preprocessing steps can further be controlled based preprocess configuration file) Options : ”yes” or ”no”. Default is ”no”
”preprocess_configuration_file”	Path to the preprocess configuration file. Ignored if ”run_preprocess_analysis” is set to ”no”
”write_preprocessed_data”	Option on whether preprocess output should be written or not.
”alignment_sort_order”	YAP gives the user the option of sorting the aligned data. The user can perform coordinate or query name sorting on their data or leave it unsorted. The possible options are ”coordinate”, ”query name” and ”unsorted”.
”merge_alignment_output”	In case the user requires the alignment to still be in chunks for postprocess, this variable can be set to ”no” the default value is ”yes”.
”run_reference_alignment”	Perform reference alignment step for the input data. (All the alignment related parameters are taken from aligner configuration file : explained in section below) Option : ”yes” or ”no” Default ”yes”
”aligner_configuration_file”	File path to the alignment algorithm configuration file. Please specify the complete path and check configuration file format . Ignored if ”run_reference_alignment” is set to ”no”
”run_postprocess_analysis”	If ”yes”, YAP runs the postprocess section after alignment
”postprocess_configuration_file”	Location of the postprocess configuration file.

Note: Variables ”file_chunk_size” and ”max_read_length” have been set as default values

file_chunk_size = 1024

max_read_length = 150

If the user wants to change these values during the workflow run he/she can mention it in the YAP workflow_configuration file within the ":begin .. :end" block.

3.4.4 Post-process Configuration File

After the reads are aligned they are post-processed. The aligners typically produce either a SAM or a BAM file as output. The alignment file need to be post-processed to get relevant information about the genome or transcriptome in question. The steps in this stage are determined by the nature of analysis that is to be conducted. For example a RNAseq analysis may invoke a set of postprocess packages as compared to a microRNA analysis. In addition, each of these packages might vary in the alignment file format and/or sort order they take as input.
From YAP 2.2 the postprocess configuration file features the following.

- Allows the user to specify the "input_file_type" for a particular postprocess package using wildcard characters.
  For eg. "*coordinate*" or "*junctions.bed" will get all the coordinate sorted alignment files or all the junction.bed files respectively.
- The user can also specify a group of packages that take the "input_file_type" in a given "input_directory" using the ":begin_tee".."end_tee" module. The user lists a group of commands after specifying the "input_directory" and "input_file_type", with in a ":begin_tee"..":end_tee" block.

A typical post process configuration file is shown below. For a complete example of a post_procress configuration file for a RNAseq analysis see Appendix

```
:begin
"execute_command" := "yes"
"command_name" := "htseq-count"
"input_directory" := "aligner_output"
"input_file_type" := "*queryname*"
"-m" := "no"
"-s no" := "yes"
"-a" := "no"
"-t" := "no"
"-i" := "no"
```

```
"-q" := "yes"
"" := "file_based_input"
"" := "gtf/hg19_refGene.gtf"
#"" := "gtf/hg19_refFlat.gtf"
"" := ">output_file.out"
:end

:begin_tee
        "input_directory" := "aligner_output"
        "input_file_type" := "*coordinate*"

        #YAP module for exon count
        :begin
        "execute_command" := "yes"
        "command_name" := "yap_exon_count"
        "-input_type" := "sam"
        "-f" := "1.0"
        "-exon_coordinates_file" := "hg19_final_exon_coord.bed"
        "-exon_CoordToNumber_file" := "hg19_final_exon_coord_number.b
ed"
        #"-i" := "file_based_input"
        "-i" := "-"
        "-o" := "output_file"
        :end

        :begin
        "execute_command" := "yes"
        "command_name" := "java -Xmx4g -jar picard-tools/1.67/CollectMultiple
Metrics.jar"
        "VALIDATION_STRINGENCY=" := "SILENT"
        "I=" := "/dev/stdin"
"O=" := "output_file"
        "ASSUME_SORTED=" := "True"
        "REFERENCE_SEQUENCE=" := "/index_files/hg19_bwa_bowtie/hg19.fa"
        "PROGRAM=" := "no"
        "QUIET=" := "True"
        :end
:end_tee
```

Postprocess Variables in YAP	
Parameter	**Description**
"input_directory"	The directory where the input files are located for that particular post process package in YAP. YAP variables such as aligner_output and postprocess_output can be used to indicate the aligner output and the postprocess output directories of the YAP run respectively.
"input_file_type"	The file format of the input file. This could be gtf, transcripts, sam, bam etc. The user can use wildcards to demarcate a set of alignment or postprocess files as postprocess input.
"list_of_samples"	If all the samples are to be considered for analysis set to "all". If the user wants a specific set of samples, provide the name of a text file which contains the sample names, comma separated. For more refer to [Appendix A.2]
"list_of_samples_to_compare"	The user provides a text file with the list of all the file combinations to be analyzed separated by a tab or space. Each line corresponds to one analysis. For more refer to [Appendix A.2]

For a complete example of a post_procress configuration file for a RNAseq analysis see Appendix

3.5 Output Directory Structure

The output directory structure of YAP is determined by the user defined options in the configuration files. Below we have the hierarchy of the YAP output directory.

YAP now includes a consolidated output folder that contains symlinks of all the alignment files across all the samples. This feature was included to make it more convenient for the user to run a subsequent YAP job with the alignement files as input. The FastQC and Fastqscreen outputs will now be written directly to the preprocess_output directory under each sample

```
/workflow_name/
    cufflinks/
            cuffdiff/
      cuffmerge/
      cuffcompare/
    workflow_summary/
consolidated_summary/
summary.pdf/
            inpfile_1/
                 /no_barcode_specified/
                      /aligner_output
                      /preprocess_output
                      /postprocess_output
    consolidated_output/
    yap_log_files/
/error_log
/status_log


 /workflow_name/
            cufflinks/
            cuffdiff/
            cuffmerge/
            cuffcompare/
            workflow_summary/
                consolidated_summary/
                summary.pdf/
            inpfile_2/
                 /no_barcode_specified/
                      /aligner_output
                      /preprocess_output
                      /postprocess_output
    consolidated_output/
    yap_log_files/
/error_log
/status_log
```

Please note that only if fastqc, fastqscreen, cuffmerge, cuffdiff and cuffcompare are run will the respective directories be created. The yap_log_files section will be explained in the following section.

3.6 Regroup Samples in YAP

In the case of experimental replicates if the user requires the bam files to be merged after the alignment step, this can be achieved through the regrouping option in YAP. All the user has to do is to supply a file containing the regroup sample name along with the samples that need to be grouped separated by a comma.

```
regroup_sample_12 sample1,sample2
regroup_sample_abc samplea,sampleb,samplec
```

Once the regroup option has been specified the user will find the results under a separate regroup_output folder under the YAP output directory. There will be a mirror of the YAP output structure as seen above except for the yap_log_files, and will be displayed in terms of the regroup sample name as shown below. Incase a YAP run consists of a regrouped sample and a non regrouped sample entering the postprocess in tandem, both of them will be output under the "regroup_output" directory structure.

```
yap_output/
    sample1/
        barcode/
            preprocess_output/
            aligner_output/
            postprocess_output/
    sample2/
        barcode/
            preprocess_output/
            aligner_output/
            postprocess_output/
    sample3/
        barcode/
            preprocess_output/
            aligner_output/
            postprocess_output/
    regroup_output/
        regrouped_sample_12/
            barcode/
                preprocess_output/
                aligner_output/
                postprocess_output/
        sample3/
            barcode/
                preprocess_output/
                aligner_output/
                postprocess_output/
    yap_log_files/
    consolidated_output/
```

Chapter 4

Yap Check New Features

The YAP check script checks for errors in the yap configuration files and informs the user of them. In YAP 2.3 we have completely changed the yap_check script and added new features that are listed below. We encourage users to run the yap_check script before submitting a yap job simply to inform the user of syntactical errors and check to see if appropriate permissions are granted, as these can't be done in runtime.

1. Validates if the file, directory and symbolic link paths specified in any configuration files exist and have required permissions.
   A path is anything which contains "/" character. If user wants YAP to validate files which are available in current working directory and referred in the YAP configuration file also, user can prefix file names with "./" and YAP will check if file exists and all necessary permissions are given.

2. Re-formatted YAP Check summary that reports, in separate sections, the status of each type of validation that is performed, and any errors if found.

   - lists the stages by status, "passed", "passed with warnings" and "failed".

   - gives detailed error and warning information of all checks
   - lists them according to section: Syntax check, Compatibility check and File Path check.

3. Validates regroup, cuffdiff, cuffmerge, cuffcompare files a reports the grouping information corresponding to each of these files if no errors are found

   (a) Validations are performed only if the particular command is switched on.

(b) Users can use unique partial names, instead of complete file paths to indicate a sample.

(c) In case of paired end data, users the user needs to specify only one pair to indicate the sample.

(d) For cuffdiff, cuffmerge and cuffcompare, users can also specify one of the groups from regroup file.

- Regroup file syntax rules:
  (a) Regroup sample name should be separated by a space or tab from the list of files it contains.
  (b) List of files should be separated by a comma.
  (c) Each regroup sample name and its files should be in same line.
- cuffdiff file syntax rules:
  (a) Each line is considered one comparison group.
  (b) In one group sets of file that need to be compared against each other should be separated by a space or tab.
  (c) If a set needs to contain more than one file then these files should be separated by a comma.
- cuffmerge and cuffcompare syntax rules:
  (a) Each line is considered one comparison group.
  (b) In one group sets of file that need to be merged or compared against each other should be separated by a space or tab.
- Below validations are performed for each of these files:
  (a) Error if file name is missing.
  (b) Error if given file name does not exist.
  (c) Error if input file is not formatted properly.
  (d) Error if any of the given names could not be uniquely translated to an input file or a group from regroup file in case of cuffdiff, cuffmerge and cuffcompare.
  (e) Error if any of the files mentioned in any group is not part of input files for current workflow.
  (f) For cuffdiff, cuffmerge and cuffcomapre, error in case of a group contains only one file
  (g) Error if any duplicate files/names are found in a single group. Adding both the files of paired end data in the same group is also considered duplicate.

4. Compatibility between workflow and postprocess configuration files.

   (a) If both run_alignment and run_postprocess are set to yes:
   check if input_file_type matches alignment_sort_order which is set in workflow config file. Only check those turned on commands whose input_file_type includes ”queryname” or ”coordinate”. Not check if it is other value, such as ”juction.bed”.

      i. if alignment_sort_order = both, match.

      ii. if input_file_type = ”queryname” or ”coordinate”, – match if it is same with alignment_sort_order. – not match if not same.

      iii. if input_file_type is any other value, don't check.

   (b) If run_alignment is set to no and run_postprocess is set to yes:
   check if needed file of given input_file_type exists in given input_directory. Give error information if not exists.

5. Contaminant validation.

   (a) Check if all filenames in the given contaminant file exist in input files list.

   (b) Check if there is filename duplicate in filenames.

6. yap command path exists and executable validation. For all turned on commands, check if it exists and is executable.

7. Default workflow configuration dictionary and workflow configuration dictionary validation.
   After reading the workflow configuration file, the workflow configuration dictionary is updated. Validate if all keys' value is valid. If not, give corresponding error information.

8. Backward compatibility with YAP 2.2
   YAP 2.3 will work even if new workflow configuration tags are not specified in the configuration file. But all the new features will not be available with old configuraiton files.

Chapter 5

YAP Error Checking and Logging

In YAP 2.3, we have addressed the issue of logging every command that is run in YAP and supplies information on whether it has passed or failed respectively. Along with this we also have a pass-fail matrix which displays, in a tabular format, whether a particular sample has run succesfully in a given stage.

In YAP 2.3, we also have an added feature of not running the subsequent stages of a failed sample. For eg. if a sample fails at the ALIGNMENT stage the subsequent POSTPROCESS commands will not be executed.

5.1 YAP Error and Status Logs

YAP now provides a set of Error(FAIL) and Status(PASS) log files per sample. They can be found under the yap_log_files directory in the output directory.

- Each sample contains a status log and an error log.
- Each log is organized stagewise, namely
  1. PREPROCESS Steps- FASTQC, FASTQSCREEN, PREPROCESS
  2. ALIGNMENT Steps- ALIGNMENT, MERGE ALIGNMENT, REGROUP
  3. POSTPROCESS Steps- POSTPROCESS, CUFFDIFF, CUFFMERGE, CUFFCOMPARE
- For a chunk-based workflow, the log will be further organized by the chunk number with the respective commands executed on that particular chunk.
- There are 4 entries for each command.
  1. YAP_COMMAND: Provides the actual command line
  2. INPUT_FILES: The input files the command used at that particular stage

3. EXIT_CODE: If 0 it means the command ran successfully else is logged in the err.log
4. YAP_ERROR_MSG/YAP_STATUS_MSG: The standardout and standard error messages written by the command.

- Each command is logged in either the status log or the error log depending on its EXIT_CODE or in other words whether it has passed or failed.
- If the REGROUP option is set to yes, then the log files will be combined and renamed in accordance with the regrouped sample name.
- If CUFFDIFF, CUFFMERGE, CUFFCOMPARE are switched on they will have their own log file as per the grouping done by YAP, suggested by the user.
- A "yap_pass_fail_matrix.log" log file will indicate stage-wise and sample-wise PASS/-FAIL statuses

```
yap_output_directory\
yap_log_files\
error_log\
sample_err.log
status_log
sample_stat.log
yap_pass_fail_matrix.log
```

5.2 YAP error based sample deletion

After the preprocess and alignment stage, YAP checks the error log files to check if there is an error in any of the previously commands executed. If so, YAP automatically discludes the respective samples from further analysis and marks them as failed. This includes from regroup and from other comparitive analyses.

5.3 YAP PASS/FAIL Matrix

As mentioned above YAP creates a PASS/FAIL matrix at the end of each YAP run which is a tabulated, tab separated flat-file that tells the user if a sample has run successfully or failed at a particular stage.

- If a stage has been switched of by the user the log will denote that with an "N/A"

- There will only be 3 stages in the table namely, PREPROCESS, ALIGNMENT and POSTPROCESS.
- CUFFDIFF, CUFFMERGE and CUFFCOMPARE will be logged separately at the bottom of the table.
- If a file has been excluded from the analysis, expect to see a 'FAIL' in the subsequent stages.

Appendix A

YAP Input Formats and Rules

A.1 YAP Input data formats

The file formats supported by YAP are :

A.1.1 SAM Format

SAM is a TAB-delimited format that describes the alignment of query sequences or sequencing reads to a reference sequence or assembly. It has been designed with the following features:

- Flexible enough to store all the alignment information generated by various alignment programs
- Simple enough to be easily generated by alignment programs or converted from existing alignment formats.
- Compact file size
- Allows most operations on alignment to work on input streams without having to load the whole alignment into memory.
- Allows the file to be indexed by genomic position to efficiently retrieve all reads aligning to a locus.

A.1.2 Fasta Format

FASTA format is a text-based format for representing either nucleotide sequences or peptide sequences. Base pairs or amino acids are represented using single-letter codes. The format also allows for sequence names and comments to precede the sequences.

A.1.3 Fatsq Format

FASTQ format is a text-based format for storing both a biological sequence (usually nucleotide sequence) and its corresponding quality scores. Both the sequence letter and quality score are encoded with a single ASCII character for brevity.

A.1.4 Illumina qseq Format

This format is used by Illumina instruments. The decription of the fields of a qseq file format are given below:

- Machine name unique
- Run number: (hopefully) unique
- lane number: [1..8]
- Tile number: positive integer
- X: x coordinate of the spot, (can be negative)
- Y: y coordinate of the spot, can be negative)
- Index: positive integer, should be greater than 0 (the files I see have it == to 0)
- Read number, 1 for single read, 1 or 2 for paired end
- sequence
- quality, the calibrated quality string.
- filter, did the read pass qc 0 - no, 1 - yes

A.1.5 Tab Separated Format

The Tab delimiter format is used as the format for barcode file but it can also be used as an input format. It consist of sequence and id separated by a tab space.

A.2 Special Input formats and Rules

A.2.1 Paired end data

While conducting analysis on paired end data, the user has to take care to follow certain naming conventions in YAP. They are as follows:

1. Both pairs have the same sample name

2. They have appropriate suffixes of '_1','_2'

3. They have the same file extensions after the suffixes.

For example:

```
sample_1.fq,sample_2.fq
```

A.2.2 Read based Barcode File Format (for small RNA)

The barcode file enables YAP to assign some meaningful names to the barcodes present in a sequences. If multiple barcodes are present the barcode removal step in the pre-process step strips the barcode from the sequence and collects and processes all sequences belonging to a particular barcode. All output is then segregated based on the logical names associated with a barcode. The file is very simple and consists of two columns separated by spaces. The first column denotes name of the barcode and the second column has respective barcode sequence. The name can be any text. The contents of a sample barcode file is shown below.

```
BC1     TTGGAC
BC2     GCCGTA
BC3     CGGCTA
BC4     TTCCGA
```

A.2.3 FASTQ/A Clipper

FASTQ/A Clipper removes sequence adapters and linkers. It takes a contaminant file as input in order to do this. YAP has a sample contaminant file shown below.

Default contaminant file in YAP

```
ACACTCTTTCCCTACACGACGCTCTTCCGATCT
GATCGGAAGAGCACACGTCT
AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT
GAACTCCAGTCACnnnnnnATCTCGTATGCCGTCTTCTGCTTG
```

However the user can create a sample contaminants file in the format specified below.

1. Each line has to have a sample name and its corresponding contaminant sequence in a tab separated format.

2. If there are more than two contaminant sequences corresponding to a single sample then the two sequences have to be separated by a ','

3. In the case of regrouped samples, please specify the regroup_title or the merged sam title.

Example of a contaminant file

```
human_sample1_1 GATCGGAAGAGCACACGTCTGAACTCCAGTCACTCAGGTATCTCGTATGCC,GATCGGA
AGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGT
human_sample2_1 GATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGT
```

A.2.4 Cuffdiff

Cuffdiff measures relative changes in transcript expression levels, splicing and promoter use. It also tracks changes in relative abundances of transcripts sharing a common TSS (transcription start site) and the primamry transcripts of a given gene. It takes a GTF2/GFF3 reference file, a sam/bam file as input. Additionally the user can provide cuffdiff with a file of the possible combinations of the input samples needed for analysis. YAP handles this using the "list_of_samples_to_compare" option in the cuffdiff configuration file. The user has the option of specifying "all" as the value of this option, in which case all the samples are submitted to cuffdiff symultaneously. They could also provide a simple text file with the list of samples to be considered for analysis in a space/tab separated format.
Alternatively the user could supply another text file which specifically provides a defined set of combinations for the analysis. It is essentially a list of commands as opposed to a list of samples. The rules of this special input file is given below.

(a) Each analysis is written in a new line.

(b) Each sample is separated by a space or a tab.

(c) If two samples are experimental replicates, they should be separated by a ','.

(d) In the case of regrouped samples, please specify the regroup_title or the merged sam title.

Example of a Cuffdiff, Cuffmerge or Cuffcompare input file

```
sample1_1,sample2_1 sample1_2,sample2_2
sample1_1 sample1_2,sample2_2
sample1_1 sample2_1 sample3_1
```

In the example above the first line compares two pairs of replicates with eachother. It must be noted that cuffdiff, for all practicle purposes treats replicates as a single entity. The second line compares one sample to a pair of experimental replicates. In the the third line 3 independent samples are going to be symultaneously submitted to cuffdiff. However if the time series option ”-T” is set to ”yes” then cuffdiff will perform analysis on the samples as a time series or sequentially rather than all at once.

A.2.5 Cuffmerge and Cuffcomapre

A.2.6 Cuffcompare

Cuffcompare takes *.gtf* files as input. It also (optionally) takes a reference annotation GFF file.
The user provides a file containing the list of sample names, one in each line, and specify the file name with full path as value to the variable ”list_of_samples_to_compare”. In the case of paired samples, giving the name of the first pair would suffice. The file format is the same as cuffdiff.
Optionally if the user wants all the files in the comparison, ”all” is to be entered as the value.

A.2.7 Cuffmerge

Cuffmerge, another package that Cufflinks includes, merges several Cufflinks assemblies. It can also run Cuffcompare, and automatically filters experimental artifact transcription fragments. The user could provide an optional reference GTF file for more elegant merging of novel and known isoforms and enhances the over all quality of the assembly GTF it produces.
Input: Cuffmerge takes as input a text file that provides a list of the GTF files to be merged (one GTF file per line) or a sam/bam file. If you switch Cuffmerge on, it will produce a merged GTF file of the inputfiles specified.
In YAP all you have to do is provide a file containing the list of sample names, one in each line, and assign that file to the variable ”list_of_samples_to_compare”. There is no necessity to specify the full file name. In the case of paired ends, giving the name of the first paired end would suffice. The file format is the same as cuffdiff.
Optionally if the user wants all the files in the comparison, ”all” is to be entered as the value.

A.2.8 Exon and Junction Counts

The Exon and Junction counts programs require an "-exon_coordinates_file" which is a reference coordinate bed file and an "-exon_CoordToNumber_file" which is a BED file that converts the corresponding coordinates to exon number. The user has the flexibility to provide their own files. The default provided by YAP is a UCSC coordinate file for gene model.

Appendix B

Configuration Files Used in YAP

In this Appendix an example of all configuration files use in YAP are shown. The configuration files for YAP 2.3 can be found at yap/2.3/cfg/demo. The configuration files point to the default YAP ucsc index files and contain appropriate YAP metacharacter variables.

B.1 Workflow Configuration File

This shows a sample workflow_configuration.cfg file which has two workflows. Both workflows share the same preprocess_configuration.cfg but have different configuration files for aligner and post-process steps. This allows the user the flexibility to change parameters at the different stages of a workflow and among different workflows.

```
# YAP main configuration
#Rules:
#Workflow is defined by the "key" := "value" configurations
#Dont modify the "key"( left hand side)
#Please specify all values in double quotes, specify full path wherever
 required.

##Meta data related to analysis
#This section is flexible,can add more "key" := "value" pair defining y
our analysis.
"comment":="120 chars"
"analyst_name":="120 chars"
"organisation_name" :="NIBR"
```

```
:begin  #Workflow configuration starts
#Experiment metadata
"instrument_type" := "Illumina"
"specimen_info" := "[tissue type]"
"seq_type" := "[DNA or RNA]"
"workflow_type" := "rnaseq"     #Type of experiment eg: "rnaseq" or "dnaseq
" or "microrna_analysis"
"workflow_name" := "yap2.3_test"        #Output directory is created by this n
ame

#Input
"input_files_path" := "yap/2.3/examples/sample_input" #Provid
 complete input directory path here. Can specify multiple paths sepera
ted by ';'
#eg: "path1;path2"
"input_files" := "human*.fq"    #Provide the input file name, this field c
an contain wild charcters or comma separated list.
#If multiple input paths, then provide list seperated ';'       #eg: "*.fq ;
*.fq" or "file1_*.fq ; sample1_*.fq"
"input_file_format" := "fastq"  #options:[qseq,fasta,fastq,sam,bam,gtf]
"paired_end_data" := "yes"      #options:["yes" or "no"]        #If paired end then
corresponding files should have suffix "_1" and "_2" before the file e
xtension.
"data_distribution_method" := "chunk_based"     #options: ["file_based" or
chunk_based"]   #If using Tophat as aligner then keep it as "file_based"


#Output
"output_files_path" := "./"     #Provide output directory path here, the di
rectory should have write access.
"regroup_output" := "yes"       #Merge the sam/bam files from replicates/lane
s after alignment step, [Options: 'yes' or 'no'(default)]
"regroup_file" := "regroup_sample.cfg"  #Provide the files to be merged.

"preprocess_output_file_format" := "fastq"      #options:["fastq" or "fasta"
or "tab"]       #Please make sure to change the input format option in align
```

```
er options

#Preprocess stage
"run_preprocess_analysis" := "yes"      #options:["yes" or "no"]        #If "no" th
en preprocess configuration is not read.
"preprocess_configuration_file" := "preprocess_configuration.cfg"
"write_preprocessed_data" := "no"       #options:["yes" or "no"]        #default "no
"       #If "yes" then preprocessed data is written to the output directory.

#Alignment stage
"run_reference_alignment" := "yes"      #options:["yes" or "no"]        #If "no" th
en aligner configuration is not read.
"aligner_configuration_file" := "bowtie_1.0.0_configuration.cfg"
"alignment_sort_order" := "both"        #options: ["queryname" or "coordinate"
 or "both" or "unsorted"]
"merge_alignment_output" := "yes"       #options: ["yes" or "no"]       #Default: "
yes"    #If "no", chunked data for each file won't be merged together.
#"aligner_configuration_file" := "bwa_0.7.4_configuration.cfg"
#"aligner_configuration_file" := "bowtie2_2.1.0_configuration.cfg"
#"aligner_configuration_file" := "tophat_2.0.8b_bowtie_configuration.cf
"       #If you are running Tophat,please make sure "data_distribution_metho
d" is set to "file_based"
#"aligner_configuration_file" := "tophat_2.0.8b_bowtie2_configuration.c
g"      #If you are running Tophat,please make sure "data_distribution_meth
od" is set to "file_based"

#Postprocess stage
"run_postprocess_analysis" := "yes"     #options: ["yes" or "no"]       #If "no"
then postprocess configuration is not read
"postprocess_configuration_file" := "postprocess_configuration.cfg"
:end    #Workflow configuration ends
```

B.2 Pre-process Configuration File

A sample preprocess configuration file is shown below:

```
#Preprocess configuration file
#General rules->
# In order to include the command in the analysis, set "execute_command
 option to 'yes'. if set to "no" command and its parameters will not b
e taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names

#fastqc - aims to provide a simple way to do some quality control check
s on raw sequence data
:begin
"execute_command" := "yes"
"command_name" := "fastqc/0.10.1/fastqc"
"--outdir" := "output_directory"
"--casava" := "no"      #Files come from raw casava output. Files in the sam
 sample group (differing only by the group number) will be analysed as
 a set rather than individually. Sequences with the filter
#flag set in the header will be excluded from the analysis.Files must h
ve the same names given to them by casava (including being gzipped and
 ending with .gz) otherwise they
#won't be grouped together correctly.
"--extract" := "yes"    #If set then the zipped output file will be uncomr
ssed in the same directory after it has been created. By default this
option will be set if fastqc is run in non-interactive mode.
"--java" := "no"        #Provides the full path to the java binary you want to
use to launch fastqc. If not supplied then java is assumed to be in yo
ur path.
"--noextract" := "no"   #Do not uncompress the output file after creating
it. You should set this option if you do not wish to uncompress the ou
tput when running in non-interactive mode.
"--nogroup" := "no"     #Disable grouping of bases for reads >50bp. All reports
will show data for every base in the read. WARNING: Using this option will caus
```

```
e fastqc to crash and burn if you use it on really long reads, and your plots m
ay end up a ridiculous size. You have been warned!

"--format" := "no"      #Bypasses the normal sequence file format detection
nd forces the program to use the specified format. Valid formats are b
am,sam,bam_mapped,sam_mapped and fastq
"--threads" := "12"     #--threads Specifies the number of files which can
e processed simultaneously. Each thread will be allocated 250MB of mem
ry so you shouldn't run more threads than your available memory will c
ope with, and not more than 6 threads on a 32 bit machine
"--contaminants" := "no"        # Specifies a non-default file which contains
he list of --contaminants contaminants to screen overrepresented seque
ces against. The file must contain sets of named contaminants in the f
orm name[tab]sequence. Lines prefixed with a hash will be ignored.
"--kmers " := "no"      #Specifies the length of Kmer to look for in the Kme
 content module. Specified Kmer length must be between 2 and 10. Defau
lt length is 5 if not specified.
"--quiet" := "no"       #--quiet Supress all progress messages on stdout and
only report errors.
:end

#fastq screen - allows you to screen a library of sequences in FastQ fo
mat against a set of sequence databases so you can see if the composit
ion of the library matches
:begin
"execute_command" := "yes"
"command_name" := " FastQScreen/v0.4.1/fastq_screen"
"--subset" := "500000"  ##Don't use the whole sequence file to search
"--paired" := "yes"     #Files are paired end.
"--outdir" := "output_directory"        # Please specify "output_directory",YA
P takes care of naming convention
"--illumina1_3" :="no"
"--quiet" := "no"       #Supress all progress reports on stderr and only repo
rt errors
"--version" := "no"     #Print the program version and exit
"--threads" := "no"     #Specify across how many threads bowtie will be all
```

```
owed to run. Overrides the default value set in the conf file
"--conf" := "fastq_screen_v0.4.1.conf"  #Manually specify a location for
the configuration file to be used for this run. If not specified then
he file will be taken from the same directory as the fastq_screen prog
ram
"--color" := "no"       #FastQ files are in colorspace. This requires that th
e libraries configures in the config file are colorspace indices.
"--bowtie" := "no"      #Specify extra parameters to be passed to bowtie. Th
se parameters should be quoted to clearly delimit bowtie parameters fr
om \    ## fastq_screen parameters.
"--bowtie2" := "no"     #Specify extra parameters to be passed to bowtie2.
hese parameters should be quoted to clearly delimit bowtie parameters
from \  ## fastq_screen parameters.
"--nohits " := "no"     #Writes to a file the sequences that did not map to
any of the specified genome libraries. If the subset option is chosen
s well, only reads from the temporary dataset that failed to align to
the reference genomes will be written to the output file.
"--aligner" := "bowtie" # --aligner Specify the aligner to use for the
mapping. Valid arguments are 'bowtie' or 'bowtie2'.
:end


#Yap module to calculate sequence content across all the bases.
:begin
"execute_command" := "no":
"command_name" := "calculate_basecount_metrics"
:end

#fastx_barcode_splitter.pl - Command for splitting of a FASTA/FASTQ fil
 based on barcode matching from FASTX Tool kit.( Note:For read based b
arcode splitting only)
:begin
"execute_command" := "no"       #Please use this command for small RNA only !
!!
"command_name" := "fastx_barcode_splitter.pl"
"--bcfile" := "sample_barcode_file"     #- Barcodes file name.
```

```
"--bol" := "yes"        #- Try to match barcodes at the BEGINNING of sequence.
What biologists would call the 5’ end, and programmers would call inde
x 0.)
"--eol" := "no" #- Try to match barcodes at the END of sequences.(What
iologists would call the 3’ end, and programmers would call the end of
 the string.)
"--mismatches" := "0"   #- Max. number of mismatches allowed. default is
1.
"--exact " := "no"      #- Same as ’--mismatches 0’. If both --exact and --m
ismatches are specified, ’--exact’ takes precedence.
"--partial" := "no"     #- Allow partial overlap of barcodes.
:end


#fastx_clipper - for clipping sequences provided in FASTA/FASTQ filefor
mat based on ADAPTER or contaminants.
:begin
"execute_command" := "no"
"command_name" := "fastx-toolkit/0.0.13/fastx_clipper"
"contaminants_file ":= "contaminants_sample_file"       # options :"no" or pr
ovide text file with contaminants (eg:contaminants_sample_file)
#if option is set to ’no’ , the default contaminants will be used. Plea
se refer to the manual for default cotaminant set
"-l" := "no"    #- discard sequences shorter than N nucleotides.[ Default
is 5 ]
"-d" := "0"     #- Keep the adapter and N bases after it. [ using ’-d 0’ is
 the same as not using ’-d’ at all. which is the default ].
"-c" := "no"    #- Discard non-clipped sequences (i.e. - keep only sequenc
es which contained the adapter).
"-C" := "yes"   #- Discard clipped sequences (i.e. - keep only sequences
which did not contained the adapter).
"-k" := "no"    #- Report Adapter-Only sequences.
"-n" := "0"     #- keep sequences with unknown (N) nucleotides. default is
to discard such sequences.
"-M" := "51"    # require minimum adapter alignment length of N. If less t
han N nucleotides aligned with the adapter - don’t clip it.
```

```
"-i" := "pipe1" # YAP takes input as pipe . pipe1 represents single inp
ut stream. dont change the value(pipe1)
:end

#fastx_quality_filter-Filters sequences based on quality
:begin
"execute_command" := "no"
"command_name" := "fastx-toolkit/0.0.13/fastq_quality_filter"

"-q" := "20"    ## N ; Minimum quality score to keep.
"-p" := "70"    ## N ; Minimum percent of bases that must have [-q] qualit
y.
"-z" := "no"    ## Compress output with GZIP.
"-i" := "pipe1" ## INFILE; FASTA/Q input file. default is STDIN.
## If [-o] is specified, report will be printed to STDOUT.
## If [-o] is not specified (and output goes to STDOUT),report will be
printed to STDERR.
"-v" := "no"    ## Verbose - report number of sequences.
:end

#fastq_quality_trimmer-Trims (cuts) sequences based on quality
:begin
"execute_command" := "no"
"command_name" := "fastx-toolkit/0.0.13/fastq_quality_trimmer
"
"-t" := "2"     # Integer value ; Quality threshold - nucleotides with lowe
r quality will be trimmed (from the end of the sequence).
"-l" := "no"    # Integer value ; Minimum length - sequences shorter than
his (after trimming) will be discarded. Default = 0 = no minimum lengt
h.
"-z" := "no"    # Compress output with GZIP.
"-i" := "pipe1" # INFILE; FASTQ input file. default is STDIN.
# If [-o] is specified, report will be printed to STDOUT.
# If [-o] is not specified (and output goes to STDOUT),report will be p
rinted to STDERR.
"-v" := "no"    # Verbose - report number of sequences.
```

```
:end
```

B.3 Aligner Configuration File

A sample aligner configuration file is shown below. This is specific to the aligner used here which is bowtie. The most important bowtie options are specified here and should serve the users in understanding how to write one on their own for a different aligner.

B.3.1 Configuration file for Bowtie Aligner

```
#Bowtie Aligner Config File     ###
#General rules->
# In order to include the command in the analysis, set "execute_command
 option to 'yes'. if set to "no" command and its parameters will not b
e taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names
#If the comamnd requires a output file name as input, then please speci
fy "output_file"
#YAP takes care of naming output file based on samples. you can aslo sp
cify any suffix. eg : "output_file.txt", "output_file_example.out" or
"output_file.pdf"
#Program recoginzes the syntax "output_file" and replaces that with pro
per output structure and output file name.
#eg : output_file.txt , output_file.out if the command generates .pdf f
iles then put output_file.pdf
#If the command requires the output directory as a input , then please
specify "output_directory".
# YAP creates proper output directory based on the analysis stage or th
e command.
#YAP takes input as a named pipe.
#if paired end then specify pipe1 and pipe2 for first pair and second p
air,respectively.
#if single end then specify 'pipe1' for read input.
```

```
:begin
"execute_command" := "yes"
"command_name" := "bowtie/1.0.0/bowtie"
## insert index here->
"" := "indexes/bowtie/hg19"    #provide path for
bowtie index

#Input:
"-q" := "yes"   # -q query input files are FASTQ .fq/.fastq (default)
"-f" := "no"    # -f query input files are (multi-)FASTA .fa/.mfa
"-r" := "no"    # -r query input files are raw one-sequence-per-line
"-c" := "no"    # -c query sequences given on cmd line (as <mates>, <singl
es>)
"-C" := "no"    # -C reads and index are in colorspace
"-Q" := "no"    # -Q/--quals <file> QV file(s) corresponding to CSFASTA in
puts; use with -f -C
"--Q1" := "no"  # --Q1/--Q2 <file> same as -Q, but for mate files 1 and
2 respectively
"-s" := "no"    # -s/--skip <int> skip the first <int> reads/pairs in the
input
"-u" := "no"    # -u/--qupto <int> stop after first <int> reads/pairs (exc
l. skipped reads)
"-5" := "no"    # -5/--trim5 <int> trim <int> bases from 5' (left) end of
reads
"-3" := "no"    # -3/--trim3 <int> trim <int> bases from 3' (right) end of
 reads
"--phred33-quals" := "no"       # --phred33-quals input quals are Phred+33 (d
efault)
"--phred64-quals" := "no"       # --phred64-quals input quals are Phred+64 (s
ame as --solexa1.3-quals)
"--solexa-quals" := "no"        # --solexa-quals input quals are from GA Pipel
ine ver. < 1.3
"--solexa1" := "no"     # --solexa1.3-quals input quals are from GA Pipelin
e ver. >= 1.3
"--integer-quals" := "no"       # --integer-quals qualities are given as spac
e-separated integers (not ASCII)
```

```
#Alignment:
"-v" := "2"     # -v <int> report end-to-end hits w/ <=v mismatches; ignore
 qualities
#or
"-n" := "no"    # -n/--seedmms <int> max mismatches in seed (can be 0-3, d
efault: -n 2)
"-e" := "no"    # -e/--maqerr <int> max sum of mismatch quals across align
ment for -n (def: 70)
"-l" := "no"    # -l/--seedlen <int> seed length for -n (default: 28)
"--nomaqround" := "no"  # --nomaqround disable Maq-like quality rounding
 for -n (nearest 10 <= 30)
"-I" := "no"    # -I/--minins <int> minimum insert size for paired-end ali
gnment (default: 0)
"-X" := "no"    # -X/--maxins <int> maximum insert size for paired-end ali
gnment (default: 250)
"--fr" := "no"  # --fr/--rf/--ff -1, -2 mates align fw/rev, rev/fw, fw/f
w (default: --fr)
"--nofw" := "no"        # --nofw/--norc do not align to forward/reverse-comple
ment reference strand
"--maxbts" := "no"      # --maxbts <int> max    # backtracks for -n 2/3 (defaul
t: 125, 800 for --best)
"--pairtries" := "no"   # --pairtries <int> max # attempts to find mate f
or anchor hit (default: 100)
"-y" := "no"    # -y/--tryhard try hard to find valid alignments, at the e
xpense of speed
"--chunkmbs" := "no"    # --chunkmbs <int> max megabytes of RAM for best-f
irst search frames (def: 64)
#Reporting:
"-k" := "10"    # -k <int> report up to <int> good alignments per read (de
fault: 1)
"-a" := "no"    # -a/--all report all alignments per read (much slower tha
n low -k)
"-m" := "10"    # -m <int> suppress all alignments if > <int> exist (def:
no limit)
"-M" := "no"    # -M <int> like -m, but reports 1 random hit (MAPQ=0); req
uires --best
```

```
"--best" := "yes"       # --best hits guaranteed best stratum; ties broken by
 quality
"--strata" := "no"      # --strata hits in sub-optimal strata aren't reporte
d (requires --best)
#Output:
"-t" := "no"    # -t/--time print wall-clock time taken by search phases
"-B" := "no"    # -B/--offbase <int> leftmost ref offset = <int> in bowtie
 output (default: 0)
"--quiet" := "no"       # --quiet print nothing but the alignments
"--refout" := "no"      # --refout write alignments to files refXXXXX.map, 1
 map per reference
"--refidx" := "no"      # --refidx refer to ref. seqs by 0-based index rathe
r than name
"--al" := "no"  # --al <fname> write aligned reads/pairs to file(s) <fna
me>
"--un" := "no"  # --un <fname> write unaligned reads/pairs to file(s) <f
name>
"--max" := "no" # --max <fname> write reads/pairs over -m limit to file
(s) <fname>
"--suppress" := "no"    # --suppress <cols> suppresses given columns (comm
a-delim'ed) in default output
"--fullref" := "no"     # --fullref write entire ref name (default: only up
 to 1st space)
#Colorspace:
"--snpphred" := "no"    # --snpphred <int> Phred penalty for SNP when deco
ding colorspace (def: 30)
#or
"--snpfrac" := "no"     # --snpfrac <dec> approx. fraction of SNP bases (e.
g. 0.001); sets --snpphred
"--col-cseq" := "no"    # --col-cseq print aligned colorspace seqs as colo
rs, not decoded bases
"--col-cqual" := "no"   # --col-cqual print original colorspace quals, no
t decoded quals
"--col-keepends" := "no"        # --col-keepends keep nucleotides at extreme e
nds of decoded alignment
#SAM:
```

```
"-S" := "yes"   # -S/--sam write hits in SAM format
"--mapq" := "no"        # --mapq <int> default mapping quality (MAPQ) to print
 for SAM alignments
"--sam-nohead" := "no"  # --sam-nohead supppress header lines (starting
with @) for SAM output
"--sam-nosq" := "no"    # --sam-nosq supppress @SQ header lines for SAM ou
tput
"--sam-RG" := "no"      # --sam-RG <text> add <text> (usually "lab=value") t
o @RG line of SAM header
#Performance:
"-o" := "no"    # -o/--offrate <int> override offrate of index; must be >=
 index's offrate
"-p" := "8"     # -p/--threads <int> number of alignment threads to launch
(default: 1)
"--mm" := "no"  # --mm use memory-mapped I/O for index; many 'bowtie's c
an share
"--shmem" := "no"       # --shmem use shared mem for index; many 'bowtie's ca
n share
#Other:
"--seed" := "no"        # --seed <int> seed for random number generator
"--verbose" := "no"     # --verbose verbose output (for debugging)
"--version" := "no"     # --version print version information and quit
"-h" := "no"    # -h/--help print this usage message
"-1" := "pipe1" #YAP takes input as pipe . pipe1 and pipe2 specify the
respective paired end reads. dont change the value(pipe1, pipe2)
"-2" := "pipe2" #YAP takes input as pipe . pipe1 and pipe2 specify the
respective paired end reads. dont change the value(pipe1, pipe2)
#" " := "pipe1" #provide pipe1 for single end read input .dont change t
he value(pipe1)
"" := ">output_file.sam"
:end
```

vfill

B.3.2 Configuration file for Bowtie2

```
#Bowtie2 Aligner Config File    ###
#General rules->
# In order to include the command in the analysis, set "execute_command
 option to 'yes'. if set to "no" command and its parameters will not b
e taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names
#If the comamnd requires a output file name as input, then please speci
fy "output_file"
#YAP takes care of naming output file based on samples. you can aslo sp
cify any suffix. eg : "output_file.txt", "output_file_example.out" or
"output_file.pdf"
#Program recoginzes the syntax "output_file" and replaces that with pro
per output structure and output file name.
#eg : output_file.txt , output_file.out if the command generates .pdf f
iles then put output_file.pdf
#If the command requires the output directory as a input , then please
specify "output_directory".
# YAP creates proper output directory based on the analysis stage or th
e command.
#YAP takes input as a named pipe.
#if paired end then specify pipe1 and pipe2 for first pair and second p
air,respectively.
#if single end then specify 'pipe1' for read input.

:begin
## Run bowtie options:
"execute_command" := "yes"
"command_name" := "bowtie2/2.1.0/bowtie2"

#Input:
"-q" := "no"    # -q query input files are FASTQ .fq/.fastq (default)
"--qseq" := "no"        # --qseq query input files are in Illumina's qseq form
at
```

```
"-f" := "no"    # -f query input files are (multi-)FASTA .fa/.mfa
"-r" := "no"    # -r query input files are raw one-sequence-per-line
"-c" := "no"    # -c <m1>, <m2>, <r> are sequences themselves, not files
"-s" := "no"    # -s/--skip <int> skip the first <int> reads/pairs in the
input (none)
"-u" := "no"    # -u/--upto <int> stop after first <int> reads/pairs (no l
imit)
"-5" := "no"    # -5/--trim5 <int> trim <int> bases from 5'/left end of re
ads (0)
"-3" := "no"    # -3/--trim3 <int> trim <int> bases from 3'/right end of r
eads (0)
"--phred33" := "no"     # --phred33 qualities are Phred+33 (default)
"--phred64" := "no"     # --phred64 qualities are Phred+64
"--int-quals" := "no"   # --int-quals qualities encoded as space-delimite
d integers
#
#Presets: Same as:
#For --end-to-end:
"--very-fast" := "no"   # --very-fast -D 5 -R 1 -N 0 -L 22 -i S,0,2.50
"--fast" := "no"        # --fast -D 10 -R 2 -N 0 -L 22 -i S,0,2.50
"--sensitive" := "no"   # --sensitive -D 15 -R 2 -N 0 -L 22 -i S,1,1.15 (
default)
"--very-sensitive" := "no"      # --very-sensitive -D 20 -R 3 -N 0 -L 20 -i
S,1,0.50
#
#For --local:
"--very-fast-local" := "no"     # --very-fast-local -D 5 -R 1 -N 0 -L 25 -i
 S,1,2.00
"--fast-local" := "no"  # --fast-local -D 10 -R 2 -N 0 -L 22 -i S,1,1.75

"--sensitive-local" := "no"     # --sensitive-local -D 15 -R 2 -N 0 -L 20 -
i S,1,0.75 (default)
"--very-sensitive-local" := "no"        # --very-sensitive-local -D 20 -R 3 -N
 0 -L 20 -i S,1,0.50
#
#Alignment:
```

```
"-N" := "no"    # -N <int> max  # mismatches in seed alignment; can be 0 or
 1 (0)
"-L" := "no"    # -L <int> length of seed substrings; must be >3, <32 (22)

"-i" := "no"    # -i <func> interval between seed substrings w/r/t read le
n (S,1,1.15)
"--n-ceil" := "no"      # --n-ceil <func> func for max  # non-A/C/G/Ts permit
ted in aln (L,0,0.15)
"--dpad" := "no"        # --dpad <int> include <int> extra ref chars on sides
of DP table (15)
"--gbar" := "no"        # --gbar <int> disallow gaps within <int> nucs of read
 extremes (4)
"--ignore-quals" := "no"        # --ignore-quals treat all quality values as 3
0 on Phred scale (off)
"--nofw" := "no"        # --nofw do not align forward (original) version of re
ad (off)
"--norc" := "no"        # --norc do not align reverse-complement version of re
ad (off)
#
"--end-to-end" := "no"  # --end-to-end entire read must align; no clippi
ng (on)
#OR
"--local" := "no"       # --local local alignment; ends might be soft clipped
 (off)
#
#Scoring:
"--ma" := "no"  # --ma <int> match bonus (0 for --end-to-end, 2 for --lo
cal)
"--mp" := "no"  # --mp <int> max penalty for mismatch; lower qual = lowe
r penalty (6)
"--np" := "no"  # --np <int> penalty for non-A/C/G/Ts in read/ref (1)
"--rdg" := "no" # --rdg <int>,<int> read gap open, extend penalties (5,
3)
"--rfg" := "no" # --rfg <int>,<int> reference gap open, extend penaltie
s (5,3)
"--score-min" := "no"   # --score-min <func> min acceptable alignment sco
```

```
re w/r/t read length
#(G,20,8 for local, L,-0.6,-0.6 for end-to-end)
#
#Reporting:
#(default) look for multiple alignments, report best, with MAPQ
#OR
"-k" := "no"    # -k <int> report up to <int> alns per read; MAPQ not mean
ingful
#OR
"-a" := "no"    # -a/--all report all alignments; very slow, MAPQ not mean
ingful
#
#Effort:
"-D" := "no"    # -D <int> give up extending after <int> failed extends in
 a row (15)
"-R" := "no"    # -R <int> for reads w/ repetitive seeds, try <int> sets o
f seeds (2)
#
#Paired-end:
"-I" := "no"    # -I/--minins <int> minimum fragment length (0)
"-X" := "no"    # -X/--maxins <int> maximum fragment length (500)
"--fr" := "no"  # --fr/--rf/--ff -1, -2 mates align fw/rev, rev/fw, fw/f
w (--fr)
"--no-mixed" := "no"    # --no-mixed suppress unpaired alignments for pair
ed reads
"--no-discordant" := "no"       # --no-discordant suppress discordant alignme
nts for paired reads
"--no-dovetail" := "no" # --no-dovetail not concordant when mates exten
d past each other
"--no-contain" := "no"  # --no-contain not concordant when one mate alig
nment contains other
"--no-overlap" := "no"  # --no-overlap not concordant when mates overlap
 at all
#
#Output:
"-t" := "no"    # -t/--time print wall-clock time taken by search phases
```

```
"--un" := "no"  # --un <path> write unpaired reads that didn't align to
<path>
"--al" := "no"  # --al <path> write unpaired reads that aligned at least
 once to <path>
"--un-conc" := "no"     # --un-conc <path> write pairs that didn't align co
ncordantly to <path>
"--al-conc" := "no"     # --al-conc <path> write pairs that aligned concord
antly at least once to <path>
#(Note: for --un, --al, --un-conc, or --al-conc, add '-gz' to the optio
n name, e.g.
"--un-gz" := "no"       # --un-gz <path>, to gzip compress output, or add '-b
z2' to bzip2 compress output.)
"--quiet" := "no"       # --quiet print nothing to stderr except serious erro
rs
"--met-file" := "no"    # --met-file <path> send metrics to file at <path>
 (off)
"--met-stderr" := "no"  # --met-stderr send metrics to stderr (off)
"--met" := "no" # --met <int> report internal counters & metrics every
<int> secs (1)
"--no-head" := "no"     # --no-head supppress header lines, i.e. lines star
ting with @
"--no-sq" := "no"       # --no-sq supppress @SQ header lines
"--rg-id" := "no"       # --rg-id <text> set read group id, reflected in @RG
line and RG:Z: opt field
"--rg" := "no"  # --rg <text> add <text> ("lab:value") to @RG line of SA
M header.
#Note: @RG line only printed when --rg-id is set.
"--omit-sec-seq" := "no"        # --omit-sec-seq put '*' in SEQ and QUAL field
s for secondary alignments.
#
#Performance:
"-o" := "no"    # -o/--offrate <int> override offrate of index; must be >=
 index's offrate
"-p" := "12"    # -p/--threads <int> number of alignment threads to launch
 (1)
"--reorder" := "no"     # --reorder force SAM output order to match order o
```

```
f input reads
"--mm" := "no"  # --mm use memory-mapped I/O for index; many 'bowtie's c
an share
#
#Other:
"--qc-filter" := "no"   # --qc-filter filter out reads that are bad accor
ding to QSEQ filter
"--seed" := "no"        # --seed <int> seed for random number generator (0)
"--non-deterministic" := "no"   # --non-deterministic seed rand. gen. arb
itrarily instead of using read attributes
"--version" := "no"     # --version print version information and quit
"-h" := "no"    # -h/--help print this usage message
## insert index here->
"-x" := "indexes/bowtie2/hg19" #provide path f
or bowtie index Index. filename prefix (minus trailing .X.bt2).
"-1" := "pipe1" # YAP takes input as pipe . pipe1 and pipe2 specify the
 respective paired end reads. dont change the value(pipe1, pipe2)
"-2" := "pipe2" # YAP takes input as pipe . pipe1 and pipe2 specify the
 respective paired end reads. dont change the value(pipe1, pipe2)
#"-U" := "pipe1"        # provide pipe1 for single end read input .dont change
 the value(pipe1)
"-S" := "output_file.sam"
:end
```

vfill

B.3.3 Configuration file for BWA

```
## bwa Config File      ###
#General rules->
# In order to include the command in the analysis, set "execute_command
 option to 'yes'. if set to "no" command and its parameters will not b
e taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names
#If the comamnd requires a output file name as input, then please speci
```

```
fy "output_file"
#YAP takes care of naming output file based on samples. you can aslo sp
cify any suffix. eg : "output_file.txt", "output_file_example.out" or
"output_file.pdf"
#Program recoginzes the syntax "output_file" and replaces that with pro
per output structure and output file name.
#eg : output_file.txt , output_file.out if the command generates .pdf f
iles then put output_file.pdf
#If the command requires the output directory as a input , then please
specify "output_directory".
# YAP creates proper output directory based on the analysis stage or th
e command.
#YAP takes input as a named pipe.
#if paired end then specify pipe1 and pipe2 for first pair and second p
air,respectively.
#if single end then specify 'pipe1' for read input.

:begin
## Run bwa options
"execute_command" := "yes"
"command_name" := "bwa/0.7.4_gcc/bin/bwa aln"
"-n" := "2"     # NUM max       #diff (int) or missing prob under 0.02 err rate (
float) [0.04]
"-o" := "no"    # INT maximum number or fraction of gap opens [1]
"-e" := "no"    # INT maximum number of gap extensions, -1 for disabling l
ong gaps [-1]
"-i" := "no"    # INT do not put an indel within INT bp towards the ends [
5]
"-d" := "no"    # INT maximum occurrences for extending a long deletion [1
0]
"-l" := "no"    # INT seed length [32]
"-k" := "no"    # INT maximum differences in the seed [2]
"-m" := "no"    # INT maximum entries in the queue [2000000]
"-t" := "12"    # INT number of threads [1]
"-M" := "no"    # INT mismatch penalty [3]
"-O" := "no"    # INT gap open penalty [11]
```

```
"-E" := "no"    # INT gap extension penalty [4]
"-R" := "no"    # INT stop searching when there are >INT equally best hits
 [30]
"-q" := "no"    # INT quality threshold for read trimming down to 35bp [0]

"-B" := "no"    # INT length of barcode
"-c" := "no"    # input sequences are in the color space
"-L" := "no"    # log-scaled gap penalty for long deletions
"-N" := "no"    # non-iterative mode: search for all n-difference hits (sl
ooow)
"-I" := "no"    # the input is in the Illumina 1.3+ FASTQ-like format
"-b" := "no"    # the input read file is in the BAM format
"-0" := "no"    # use single-end reads only (effective with -b)
"-1" := "no"    # use the 1st read in a pair (effective with -b)
"-2" := "no"    # use the 2nd read in a pair (effective with -b)
"" := "indexes/bwa/hg19"       #provide path for bwa
 index
"" := "pipe1"   # YAP takes input as pipe . pipe1 and pipe2 specify the r
espective paired end reads. dont change the value(pipe1, pipe2)
"" := ">output_file_seq1.sai"
:end

:begin
"execute_command" := "yes"      #Set this option to "no" for single end read
 alignments!!!
"command_name" := "bwa/0.7.4_gcc/bin/bwa aln"
"-n" := "2"     # NUM max       #diff (int) or missing prob under 0.02 err rate (
float) [0.04]
"-o" := "no"    # INT maximum number or fraction of gap opens [1]
"-e" := "no"    # INT maximum number of gap extensions, -1 for disabling l
ong gaps [-1]
"-i" := "no"    # INT do not put an indel within INT bp towards the ends [
5]
"-d" := "no"    # INT maximum occurrences for extending a long deletion [1
0]
"-l" := "no"    # INT seed length [32]
```

```
"-k" := "no"    # INT maximum differences in the seed [2]
"-m" := "no"    # INT maximum entries in the queue [2000000]
"-t" := "12"    # INT number of threads [1]
"-M" := "no"    # INT mismatch penalty [3]
"-O" := "no"    # INT gap open penalty [11]
"-E" := "no"    # INT gap extension penalty [4]
"-R" := "no"    # INT stop searching when there are >INT equally best hits
 [30]
"-q" := "no"    # INT quality threshold for read trimming down to 35bp [0]

"-B" := "no"    # INT length of barcode
"-c" := "no"    # input sequences are in the color space
"-L" := "no"    # log-scaled gap penalty for long deletions
"-N" := "no"    # non-iterative mode: search for all n-difference hits (sl
ooow)
"-I" := "no"    # the input is in the Illumina 1.3+ FASTQ-like format
"-b" := "no"    # the input read file is in the BAM format
"-0" := "no"    # use single-end reads only (effective with -b)
"-1" := "no"    # use the 1st read in a pair (effective with -b)
"-2" := "no"    # use the 2nd read in a pair (effective with -b)
"" := "indexes/bwa/hg19"       #provide path for bwa
 index
"" := "pipe2"   # YAP takes input as pipe . pipe1 and pipe2 specify the r
espective paired end reads. dont change the value(pipe1, pipe2)
"" := ">output_file_seq2.sai"
:end



:begin
"execute_command" := "yes"      #Set this option to "no" for single end read
 alignments!!!
"command_name" := "bwa/0.7.4_gcc/bin/bwa sampe"
"-a" := "no"    #INT maximum insert size [500]
"-o" := "no"    #INT maximum occurrences for one end [100000]
"-n" := "5"     #INT maximum hits to output for paired reads [3]
"-N" := "no"    #INT maximum hits to output for discordant pairs [10]
```

```
"-c" := "no"    #FLOAT prior of chimeric rate (lower bound) [1.0e-05]
"-r" := "no"    #STR read group header line such as ‘@RG\tID:foo\tSM:bar’
[null]
"-P" := "no"    #preload index into memory (for base-space reads only)
"-s" := "no"    #disable Smith-Waterman for the unmapped mate
"-A" := "no"    #disable insert size estimate (force -s)
"" := "indexes/bwa/hg19"       #provide path for bwa
 index
"" := "output_file_seq1.sai"
"" := "output_file_seq2.sai"
"" := "pipe1"   # YAP takes input as pipe . pipe1 and pipe2 specify the r
espective paired end reads. dont change the value(pipe1, pipe2)
"" := "pipe2"   # YAP takes input as pipe . pipe1 and pipe2 specify the r
espective paired end reads. dont change the value(pipe1, pipe2)
"" := ">output_file.sam"
:end


:begin
"execute_command" := "no"       #Set this option to "yes" for single end read
 alignments!!!
"command_name" := "bwa/0.7.4_gcc/bin/bwa samse"
"-n" := "2"     #Maximum number of alignments to output in the XA tag for r
ads paired properly. If a read has more than INT hits, the XA tag will
 not be written.[3]
"-f" := "output_file.sam"
"-r" := "no"    #option: STR    #specify the read group in a format like ’@RG
\tID:foo\tSM:bar’, [null]
"" := "indexes/bwa/hg19"       #provide path for bwa
 index
"" := "output_file_seq1.sai"
"" := "pipe1"   # YAP takes input as pipe .
"" := ">output_file.sam"
:end
```

vfill

B.3.4 Configuration file for Tophat with Bowtie2

```
#Tophat 2.0.8b configuration file
#General rules->
#In order to include the command in the analysis, set "execute_command"
option to 'yes'. if set to "no" command and its parameters will not be
 taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names
#If the comamnd requires a output file name as input, then please speci
fy "output_file"
#YAP takes care of naming output file based on samples. you can aslo sp
cify any suffix. eg : "output_file.txt", "output_file_example.out" or
"output_file.pdf"
#Program recoginzes the syntax "output_file" and replaces that with pro
per output structure and output file name.
#eg : output_file.txt , output_file.out if the command generates .pdf f
iles then put output_file.pdf
#If the command requires the output directory as a input , then please
specify "output_directory".
#YAP creates proper output directory based on the analysis stage or the
 command.
#YAP takes input as a named pipe.
#if paired end then specify pipe1 and pipe2 for first pair and second p
air,respectively.



#tophat:
#TopHat maps short sequences from spliced transcripts to whole genomes.


#
#Usage:
#tophat [options] <bowtie_index> <reads1[,reads2,...]> [reads1[,reads2,
...]] \
#[quals1,[quals2,...]] [quals1[,quals2,...]]
```

```
#

:begin
"execute_command" := "yes"
"command_name" := "tophat/2.0.8b/bin/tophat"

#Options:
"-v" := "no"    # -v/--version
"-o" := "output_directory"      # -o/--output-dir <string> [ default: ./toph
at_out ]
"--bowtie1" := "no"     # --bowtie1 [ default: bowtie2 ]
"-N" := "no"    # -N/--read-mismatches <int> [ default: 2 ]
"--read-gap-length" := "no"     # --read-gap-length <int> [ default: 2 ]
"--read-edit-dist" := "no"      # --read-edit-dist <int> [ default: 2 ]
"--read-realign-edit-dist" := "no"      # --read-realign-edit-dist <int> [ d
efault: "read-edit-dist" + 1 ]
"-a" := "no"    # -a/--min-anchor <int> [ default: 8 ]
"-m" := "no"    # -m/--splice-mismatches <0-2> [ default: 0 ]
"-i" := "no"    # -i/--min-intron-length <int> [ default: 50 ]
"-I" := "no"    # -I/--max-intron-length <int> [ default: 500000 ]
"-g" := "no"    # -g/--max-multihits <int> [ default: 20 ]
"--suppress-hits" := "no"       # --suppress-hits
"-x" := "no"    # -x/--transcriptome-max-hits <int> [ default: 60 ]
"-M" := "no"    # -M/--prefilter-multihits (for -G/--GTF option, enable
#an initial bowtie search against the genome)

"--max-insertion-length" := "no"        # --max-insertion-length <int> [ defau
lt: 3 ]
"--max-deletion-length" := "no" # --max-deletion-length <int> [ default
: 3 ]
"--solexa-quals" := "no"        # --solexa-quals
"--solexa1" := "no"     # --solexa1.3-quals (same as phred64-quals)
"--phred64-quals" := "no"       # --phred64-quals (same as solexa1.3-quals)
"-Q" := "no"    # -Q/--quals
"--integer-quals" := "no"       # --integer-quals
"-C" := "no"    # -C/--color (Solid - color space)
```

```
"--color-out" := "no"   # --color-out
"--library-type" := "no"        # --library-type <string> (fr-unstranded, fr-f
irststrand, fr-secondstrand)
"-p" := "no"    # -p/--num-threads <int> [ default: 1 ]
"-R" := "no"    # -R/--resume <out_dir> ( try to resume execution )
"-G" := "yap/human-ucsc-refGene.gtf"   # -G/--GT
 <filename> (GTF/GFF with known transcripts)    # use this for transcript
 model
#"-G" := "yap/human-ucsc-refFlat.gtf"  # -G/--G
F <filename> (GTF/GFF with known transcripts)   # use this for transcrip
t model
"--transcriptome-index" := "indexes/tophat_tra
scriptome/bowtie2/hg19_refgene.transcriptome"   # --transcriptome-index
<bwtidx>        #use this for transcript model
#"--transcriptome-index" := "indexes/tophat_tr
nscriptome/bowtie2/hg19.refFlat.transcriptome"  # --transcriptome-index
 <bwtidx>       #use this for transcript model
"-T" := "yes"   # -T/--transcriptome-only (map only to the transcriptome)

"-j" := "no"    # -j/--raw-juncs <filename>
"--insertions" := "no"  # --insertions <filename>
"--deletions" := "no"   # --deletions <filename>
"-r" := "no"    # -r/--mate-inner-dist <int> [ default: 50 ]
"--mate-std-dev" := "no"        # --mate-std-dev <int> [ default: 20 ]
"--no-novel-juncs" := "no"      # --no-novel-juncs
"--no-novel-indels" := "no"     # --no-novel-indels
"--no-gtf-juncs" := "no"        # --no-gtf-juncs
"--no-coverage-search" := "no"  # --no-coverage-search
"--coverage-search" := "no"     # --coverage-search
"--microexon-search" := "no"    # --microexon-search
"--keep-tmp" := "no"    # --keep-tmp
"--tmp-dir" := "no"     # --tmp-dir <dirname> [ default: <output_dir>/tmp ]

"-z" := "no"    # -z/--zpacker <program> [ default: gzip ]
"-X" := "no"    # -X/--unmapped-fifo [use mkfifo to compress more temporar
y files for color space reads]
```

```
#
#Advanced Options:
"--report-secondary-alignments" := "no" # --report-secondary-alignments
"--no-discordant" := "no"       # --no-discordant
"--no-mixed" := "no"    # --no-mixed
"--segment-mismatches" := "no"  # --segment-mismatches <int> [ default:
2 ]
"--segment-length" := "no"      # --segment-length <int> [ default: 25 ]
"--bowtie-n" := "no"    # --bowtie-n [ default: bowtie -v ]
"--min-coverage-intron" := "no" # --min-coverage-intron <int> [ default
: 50 ]
"--max-coverage-intron" := "no" # --max-coverage-intron <int> [ default
: 20000 ]
"--min-segment-intron" := "no"  # --min-segment-intron <int> [ default:
50 ]
"--max-segment-intron" := "no"  # --max-segment-intron <int> [ default:
500000 ]
"--no-sort-bam" := "no" # --no-sort-bam (Output BAM is not coordinate-s
orted)
"--no-convert-bam" := "no"      # --no-convert-bam (Do not output bam format
.
#Output is <output_dir>/accepted_hit.sam)
"--keep-fasta-order" := "no"    # --keep-fasta-order
"--allow-partial-mapping" := "no"       # --allow-partial-mapping

#Bowtie2 related options:
#Preset options in --end-to-end mode (local alignment is not used in To
pHat2)
"--b2-very-fast" := "no"        # --b2-very-fast
"--b2-fast" := "no"     # --b2-fast
"--b2-sensitive" := "no"        # --b2-sensitive
"--b2-very-sensitive" := "no"   # --b2-very-sensitive

#Alignment options
"--b2-N" := "no"        # --b2-N <int> [ default: 0 ]
```

```
"--b2-L" := "no"        # --b2-L <int> [ default: 20 ]
"--b2-i" := "no"        # --b2-i <func> [ default: S,1,1.25 ]
"--b2-n-ceil" := "no"   # --b2-n-ceil <func> [ default: L,0,0.15 ]
"--b2-gbar" := "no"     # --b2-gbar <int> [ default: 4 ]

#Scoring options
"--b2-mp" := "no"       # --b2-mp <int>,<int> [ default: 6,2 ]
"--b2-np" := "no"       # --b2-np <int> [ default: 1 ]
"--b2-rdg" := "no"      # --b2-rdg <int>,<int> [ default: 5,3 ]
"--b2-rfg" := "no"      # --b2-rfg <int>,<int> [ default: 5,3 ]
"--b2-score-min" := "no"        # --b2-score-min <func> [ default: L,-0.6,-0.6
 ]

#Effort options
"--b2-D" := "no"        # --b2-D <int> [ default: 15 ]
"--b2-R" := "no"        # --b2-R <int> [ default: 2 ]

#Fusion related options:
"--fusion-search" := "no"       # --fusion-search
"--fusion-anchor-length" := "no"        # --fusion-anchor-length <int> [ defau
lt: 20 ]
"--fusion-min-dist" := "no"     # --fusion-min-dist <int> [ default: 100000
00 ]
"--fusion-read-mismatches" := "no"      # --fusion-read-mismatches <int> [ d
efault: 2 ]
"--fusion-multireads" := "no"   # --fusion-multireads <int> [ default: 2
]
"--fusion-multipairs" := "no"   # --fusion-multipairs <int> [ default: 2
]
"--fusion-ignore-chromosomes" := "no"   # --fusion-ignore-chromosomes <li
st> [ e.g, <chrM,chrX> ]
"--fusion-do-not-resolve-conflicts" := "no"     # --fusion-do-not-resolve-c
onflicts [this is for test purposes ]

#SAM Header Options (for embedding sequencing run metadata in output):
"--rg-id" := "no"       # --rg-id <string> (read group ID)
```

```
"--rg-sample" := "no"   # --rg-sample <string> (sample ID)
"--rg-library" := "no"  # --rg-library <string> (library ID)
"--rg-description" := "no"      # --rg-description <string> (descriptive str
ing, no tabs allowed)
"--rg-platform-unit" := "no"    # --rg-platform-unit <string> (e.g Illumin
a lane ID)
"--rg-center" := "no"   # --rg-center <string> (sequencing center name)
"--rg-date" := "no"     # --rg-date <string> (ISO 8601 date of the sequenci
ng run)
"--rg-platform" := "no" # --rg-platform <string> (Sequencing platform d
escriptor)

#index information
"" := "indexes/bowtie2/hg19"

# file input
"" := "pipe1"   #YAP takes input as pipe . pipe1 and pipe2 specify the re
spective paired end reads. dont change the value(pipe1, pipe2)
"" := "pipe2"#YAP takes input as pipe . pipe1 and pipe2 specify the res
pective paired end reads. dont change the value(pipe1, pipe2)
:end
```

B.4 Post-process Configuration File

A sample post-process configuration file is shown below. This is specific to a particular analysis a user is interested in undertaking. The configuration file below has a number of options for QC as well as for RNAseq and microRNA analyses. Multiple commands can be placed here and the user can choose to execute or not execute a particular by using the "execute_command" option for every command. Setting this option to "yes" or "no" executes or prvents execution of that particular command. In most of the commands input file/s are provided through input stream i.e "/dev/stdin".But some command like "CollectMultipleMetrics" doesn't accepts files through input stream and requires exact file name in the argument list.This is achived by setting the option "I" to "file_based_input" and YAP finds the appropriate input file for the comand . HTSeq script in this section uses "GTF and GFF" files.The later files can be downloaded from UCSC browser, (http://genome.ucsc.edu/cgi-bin/hgTables).A help page about various Data formats of gene files can be found at (http://genome.ucsc.edu/FAQ/)

```
#Postprocess configuration file
#General rules->
# In order to include the command in the analysis, set "execute_command
 option to 'yes'. if set to "no" command and its parameters will not b
e taken into consideration.
#All paths should be in double quotes
#No spaces/blanks between file names
#If the comamnd requires a output file name as input, then please speci
fy "output_file"
#YAP takes care of naming output file based on samples. you can aslo sp
cify any suffix. eg : "output_file.txt", "output_file_example.out" or
"output_file.pdf"
#Program recoginzes the syntax "output_file" and replaces that with pro
per output structure and output file name.
#eg : output_file.txt , output_file.out if the command generates .pdf f
iles then put output_file.pdf
#If the command requires the output directory as a input , then please
specify "output_directory".
# YAP creates proper output directory based on the analysis stage or th
e command.

#If the command does not read from input pipe and requires a file based
 input, Please specify(file_based_input) as the variable.

:begin
"execute_command" := "yes"
"command_name" := "yap_junction_count"
"input_directory" := "aligner_output"
"input_file_type" := "*junctions.bed*"
"-exon_coordinates_file" := "hg19/final_ex
n_coord.bed"    #-exon_coordinates_file[ucsc coordinate bed file for gene
 model]
"-exon_CoordToNumber_file" := "hg19/final_
```

```
xon_coord_number.bed"   #-exon_CoordToNumber_file[filename] for gene mod
el, coordinates are converted to exon number
"-i" := "-"     #-i [file_based_input or for stdin '-'] keep it as "-"
"-o" := "output_file"   #[output filename prefix] keep is as "output_file
"
:end

#htseq-count for sam files
:begin
"execute_command" := "yes"
"command_name" := "python/2.6.5_gnu/bin/htseq-count"
"input_directory" := "aligner_output"
"input_file_type" := "*queryname*.sam"
"-m" := "no"    #-m(mode) = mode to handle reads overlapping more than one
feature(choices: union, intersection-strict,intersection-nonempty; DEF
AULT: union)
"-s no" := "yes"        #-s(stranded)= whether the data is from a strand-speci
fic assay (DEFAULT: yes)
"-a" := "no"    #-a (minaqual) = skip all reads with alignment quality low
er than the given minimum value (DEFAULT: 0)
"-t" := "no"    #-t(feature type)= feature type(3rd column in GFF file)to
e used, all features of other type are ignored (DEFAULT, suitable for
Ensembl GTF files: exon)
"-i" := "no"    #-i(idattr) = GFF attribute to be used as feature ID (DEFA
ULT,suitable for Ensembl GTF files: gene_id)
"-q" := "yes"   #-q(quiet) = suppress progress report and warnings ((choi
ces: yes,no) DEFAULT: yes)
"" := "file_based_input"        # yap takes input directly as STDIN .dont chan
ge '-' the value
"" := "hg19_refGene.gtf"      #path of the
gtf file (features in GFF format) (cannot be null).use for trascript m
odel.
#"" := "hg19_refFlat.gtf"     #path of th
 gtf file (features in GFF format) (cannot be null).use for gene model
.
"" := ">output_file.out"
```

```
:end

#htseq-count for bam files
:begin
"execute_command" := "no"
"command_name" := "samtools"
"input_directory" := "aligner_output"
"input_file_type" := "*queryname*.bam"
"view" := "-h"
" " := "-"
" " := "|"
" " := "python/2.6.5_gnu/bin/htseq-count"
"-m" := "no"    #-m(mode) = mode to handle reads overlapping more than one
feature(choices: union, intersection-strict,intersection-nonempty; DEF
AULT: union)
"-s no" := "yes"        #-s(stranded)= whether the data is from a strand-speci
fic assay (DEFAULT: yes)
"-a" := "no"    #-a (minaqual) = skip all reads with alignment quality low
er than the given minimum value (DEFAULT: 0)
"-t" := "no"    #-t(feature type)= feature type(3rd column in GFF file)to
e used, all features of other type are ignored (DEFAULT, suitable for
Ensembl GTF files: exon)
"-i" := "no"    #-i(idattr) = GFF attribute to be used as feature ID (DEFA
ULT,suitable for Ensembl GTF files: gene_id)
"-q" := "yes"   #-q(quiet) = suppress progress report and warnings ((choi
ces: yes,no) DEFAULT: yes)
"" := "-"       # yap takes input directly as STDIN .dont change '-' the valu
e
"" := "hg19_refGene.gtf"      #path of the
gtf file (features in GFF format) (cannot be null).use for trascript m
odel.
#"" := "hg19_refFlat.gtf"     #path of th
 gtf file (features in GFF format) (cannot be null).use for gene model
.
"" := ">output_file_htseq-count.out"
:end
```

```
:begin_tee
"input_directory" := "aligner_output"
"input_file_type" := "*coordinate*"

#YAP module for exon count
:begin
"execute_command" := "yes"
"command_name" := "yap_exon_count"
"-f" := "1.0"   #-f [PercentOverlap] cannot be null
"-exon_coordinates_file" := "hg19/final_ex
n_coord.bed"    #-exon_coordinates_file[ucsc coordinate bed file for gene
 model]
"-exon_CoordToNumber_file" := "hg19/final_
exon_coord_number.bed"  #-exon_CoordToNumber_file [filename]
"-i" := "-"     #-i [file_based_input or for stdin '-'] keep it as "-"
"-o" := "output_file"   #[output filename prefix] keep is as "output_file
"
:end

:begin
"execute_command" := "no"
"command_name" := "java -Xmx4g -jar picard-tools/1.89/Collect
MultipleMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"I=" := "/dev/stdin"
"O=" := "output_file"
"ASSUME_SORTED=" := "True"      ### set assume_sorted 'True' always. YAP by
default sorts SAM file based on coordinate.
"REFERENCE_SEQUENCE=" := "hg19.fa"
        ## Reference database sequence should have read acce
"PROGRAM=" := "no"      #takes default values
"QUIET=" := "True"
:end

:begin
```

```
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Collect
AlignmentSummaryMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"I=" := "/dev/stdin"
"O=" := "output_file.txt"
"IS_BISULFITE_SEQUENCED=" := "true"
"ASSUME_SORTED=" := "True"
"REFERENCE_SEQUENCE=" := "hg19.fa"
        ## Reference database sequence should have read acce
"ADAPTER_SEQUENCE=" := "no"
"QUIET=" := "True"
:end

#picard quality score distribution
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Quality
ScoreDistribution.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"I=" := "/dev/stdin"
"O=" := "output_file.txt"
"ASSUME_SORTED=" := "true"
"REFERENCE_SEQUENCE=" := "hg19.fa"

"CHART=" := "output_file.pdf"
"ALIGNED_READS_ONLY=" := "true"
"QUIET=" := "no"
:end

#picard mean quality by cycle
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/MeanQua
lityByCycle.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
```

```
"ASSUME_SORTED=" := "true"
"REFERENCE_SEQUENCE=" := "hg19.fa"

"I=" := "/dev/stdin"
"O=" := "output_file.txt"
"CHART=" := "output_file.pdf"
"ALIGNED_READS_ONLY=" := "true"
"QUIET=" := "no"
:end

#picard collect GC bias metrics
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Collect
GcBiasMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"I=" := "/dev/stdin"
"O=" := "output_file.txt"
"SUMMARY_OUTPUT=" := "output_file_summary.txt"
"CHART=" := "output_file.pdf"
"ASSUME_SORTED=" := "true"
"REFERENCE_SEQUENCE=" := "hg19.fa"

"QUIET=" := "no"
:end

#picard collect rnaseq metrics
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Collect
RnaSeqMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"ASSUME_SORTED=" := "true"
"REF_FLAT=" := "human_refflat_for_pic
ard.gff"
"RIBOSOMAL_INTERVALS=" := "Homo_sapie
```

```
ns_assembly19.rRNA.interval_list"
"STRAND_SPECIFICITY=" := "NONE"
"I=" := "/dev/stdin"
"O=" := "output_file.txt"
"CHART_OUTPUT=" := "output_file.pdf"
"REFERENCE_SEQUENCE=" := "hg19.fa"

:end

:begin
"execute_command" := "yes"
"command_name" := "picard-tools/1.89/CalculateHsMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"BAIT_INTERVALS=" := "hg19/TruSeq_exome_ta
geted_regions_for_picard.bed"   #An interval list file that contains the
 locations of the baits used. Required.
"BAIT_SET_NAME=" := "no"        # Bait set name. If not provided it is inferre
d from the filename of the bait intervals.
#Default value: null.
"TARGET_INTERVALS=" := "hg19/TruSeq_exome_
argeted_regions_for_picard.bed" #An interval list file that contains t
he locations of the targets. Required.
"INPUT=" := "/dev/stdin"        #An aligned SAM or BAM file. Required.
"OUTPUT=" := "output_file.txt"  #The output file to write the metrics to
. Required.
"METRIC_ACCUMULATION_LEVEL=" := "no"    #MetricAccumulationLevel The level
s) at which to accumulate metrics. Possible values: {ALL_READS, SAMPLE
,
#LIBRARY, READ_GROUP} This option may be specified 0 or more times. Thi
s option can be set
#to 'null' to clear the default list.
"REFERENCE_SEQUENCE=" := "hg19.fa"

"PER_TARGET_COVERAGE=" := "no"  #An optional file to output per target c
overage information to. Default value: null.
:end
```

```
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Collect
TargetedPcrMetrics.jar"
"VALIDATION_STRINGENCY=" := "SILENT"
"AMPLICON_INTERVALS=" := "hg19/TruSeq_exom
_targeted_regions_for_picard.bed"       # An interval list file that contain
s the locations of the baits used. Required.
"CUSTOM_AMPLICON_SET_NAME=" := "no"     #Custom amplicon set name. If not p
rovided it is inferred from the filename of the
#AMPLICON_INTERVALS intervals. Default value: null.
"TARGET_INTERVALS=" := "hg19/TruSeq_exome_
argeted_regions_for_picard.bed" #An interval list file that contains t
he locations of the targets. Required.
"INPUT=" := "/dev/stdin"        #An aligned SAM or BAM file. Required.
"OUTPUT=" := "output_file.txt"  #The output file to write the metrics to
. Required.
"METRIC_ACCUMULATION_LEVEL=" := "no"    #The level(s) at which to accumula
te metrics. Possible values: {ALL_READS, SAMPLE,
#LIBRARY, READ_GROUP} This option may be specified 0 or more times. Thi
s option can be set
#to 'null' to clear the default list.
"REFERENCE_SEQUENCE=" := "hg19.fa"

"PER_TARGET_COVERAGE=" := "no"  # An optional file to output per target
coverage information to. Default value: null.
:end

:end_tee



#picard collect insert size metrics
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/Collect
```

```
InsertSizeMetrics.jar"
"input_directory" := "aligner_output"
"input_file_type" := "*coordinate*"
"VALIDATION_STRINGENCY=" := "SILENT"
"ASSUME_SORTED=" := "true"
"I=" := "file_based_input"
"O=" := "output_file.txt"
"H=" := "output_file.pdf"
"QUIET=" := "no"
"TMP_DIR=" := "/scratch/$USER"
"STOP_AFTER=" := "no"
"HISTOGRAM_WIDTH=" := "500"
:end

#picard mark duplicates
:begin
"execute_command" := "yes"
"command_name" := "java -Xmx1g -jar picard-tools/1.89/MarkDup
licates.jar"
"input_directory" := "aligner_output"
"input_file_type" := "*coordinate*"
"VALIDATION_STRINGENCY=" := "SILENT"
"TMP_DIR=" := "/scratch/$USER"
"MAX_RECORDS_IN_RAM=" := "1000000"
"MAX_SEQUENCES_FOR_DISK_READ_ENDS_MAP=" := "2000"
"ASSUME_SORTED=" := "true"
"I=" := "file_based_input"
"O=" := "output_file.bam"
"METRICS_FILE=" := "output_file.txt"
:end

#cufflinks v2.1.1
#Usage: cufflinks [options] <hits.sam>
:begin
"execute_command" := "yes"
"command_name" := "cufflinks/2.1.1/cufflinks"
```

```
"input_directory" := "aligner_output"
"input_file_type" := "*coordinate*"
"" := "file_based_input"        #
#General Options:
"-o" := "output_directory"      # -o/--output-dir write all output files to
this directory [ default: ./ ]
"-p" := "12"    # -p/--num-threads number of threads used during analysis
[ default: 1 ]
"--seed" := "no"        # --seed value of random number generator seed [ defau
lt: 0 ]
"-G" := "hg19_refGene.gtf"    # -G/--GTF
 quantitate against reference transcript annotations
#"-G" := "hg19_refFlat.gtf"   # -G/--GT
F quantitate against reference transcript annotations
"-g" := "no"    # -g/--GTF-guide use reference transcript annotation to gu
ide assembly
"-M" := "no"    # -M/--mask-file ignore all alignment within transcripts i
n this file
"-b" := "no"    # -b/--frag-bias-correct use bias correction - reference f
asta required [ default: NULL ]
"-u" := "no"    # -u/--multi-read-correct use 'rescue method' for multi-re
ads (more accurate) [ default: FALSE ]
"--library-type" := "no"        # --library-type library prep used for input r
eads [ default: below ]
"--library-norm-method" := "no" # --library-norm-method Method used to
normalize library sizes [ default: below ]
#Advanced Abundance Estimation Options:
"-m" := "no"    # -m/--frag-len-mean average fragment length (unpaired rea
ds only) [ default: 200 ]
"-s" := "no"    # -s/--frag-len-std-dev fragment length std deviation (unp
aired reads only) [ default: 80 ]
"--max-mle-iterations" := "no"  # --max-mle-iterations maximum iteration
s allowed for MLE calculation [ default: 5000 ]
"--compatible-hits-norm" := "no"        # --compatible-hits-norm count hits co
mpatible with reference RNAs only [ default: FALSE ]
"--total-hits-norm" := "no"     # --total-hits-norm count all hits for norm
```

```
alization [ default: TRUE ]
"--num-frag-count-draws" := "no"        # --num-frag-count-draws Number of fra
gment generation samples [ default: 100 ]
"--num-frag-assign-draws" := "no"       # --num-frag-assign-draws Number of f
ragment assignment samples per generation [ default: 50 ]
"--max-frag-multihits" := "no"  # --max-frag-multihits Maximum number of
 alignments allowed per fragment [ default: unlim ]
"--no-effective-length-correction" := "no"      # --no-effective-length-corr
ection No effective length correction [ default: FALSE ]
"--no-length-correction" := "no"        # --no-length-correction No length cor
rection [ default: FALSE ]
"-N" := "no"    # -N/--upper-quartile-norm Deprecated, use --library-norm-
method [ DEPRECATED ]
"--raw-mapped-norm" := "no"     # --raw-mapped-norm Deprecated, use --libra
ry-norm-method [ DEPRECATED ]
#Advanced Assembly Options:
"-L" := "no"    # -L/--label assembled transcripts have this ID prefix [ d
efault: CUFF ]
"-F" := "no"    # -F/--min-isoform-fraction suppress transcripts below thi
s abundance level [ default: 0.10 ]
"-j" := "no"    # -j/--pre-mrna-fraction suppress intra-intronic transcrip
ts below this level [ default: 0.15 ]
"-I" := "no"    # -I/--max-intron-length ignore alignments with gaps longe
r than this [ default: 300000 ]
"-a" := "no"    # -a/--junc-alpha alpha for junction binomial test filter
[ default: 0.001 ]
"-A" := "no"    # -A/--small-anchor-fraction percent read overhang taken a
s 'suspiciously small' [ default: 0.09 ]
"--min-frags-per-transfrag" := "no"     # --min-frags-per-transfrag minimum
 number of fragments needed for new transfrags [ default: 10 ]
"--overhang-tolerance" := "no"  # --overhang-tolerance number of termina
l exon bp to tolerate in introns [ default: 8 ]
"--max-bundle-length" := "no"   # --max-bundle-length maximum genomic len
gth allowed for a given bundle [ default:3500000 ]
"--max-bundle-frags" := "no"    # --max-bundle-frags maximum fragments all
owed in a bundle before skipping [ default: 500000 ]
```

```
"--min-intron-length" := "no"   # --min-intron-length minimum intron size
 allowed in genome [ default: 50 ]
"--trim-3-avgcov-thresh" := "no"        # --trim-3-avgcov-thresh minimum avg c
overage required to attempt 3' trimming [ default: 10 ]
"--trim-3-dropoff-frac" := "no" # --trim-3-dropoff-frac fraction of avg
 coverage below which to trim 3' end [ default: 0.1 ]
"--max-multiread-fraction" := "no"      # --max-multiread-fraction maximum f
raction of allowed multireads per transcript [ default: 0.75 ]
"--overlap-radius" := "no"      # --overlap-radius maximum gap size to fill
between transfrags (in bp) [ default: 50 ]
#Advanced Reference Annotation Guided Assembly Options:
"--no-faux-reads" := "no"       # --no-faux-reads disable tiling by faux read
s [ default: FALSE ]
"--3-overhang-tolerance" := "no"        # --3-overhang-tolerance overhang allo
wed on 3' end when merging with reference[ default: 600 ]
"--intron-overhang-tolerance" := "no"   # --intron-overhang-tolerance ove
rhang allowed inside reference intron when merging [ default: 30 ]
#
#Advanced Program Behavior Options:
"-v" := "no"    # -v/--verbose log-friendly verbose processing (no progres
s bar) [ default: FALSE ]
"-q" := "no"    # -q/--quiet log-friendly quiet processing (no progress ba
r) [ default: FALSE ]
"--no-update-check" := "no"     # --no-update-check do not contact server t
o check for update availability[ default: FALSE ]
:end

#cuffdiff v2.1.1 (exported)
#sage: cuffdiff [options] <transcripts.gtf> <sample1_hits.sam> <sample2
_hits.sam> [... sampleN_hits.sam]
#Supply replicate SAMs as comma separated lists for each condition: sam
ple1_rep1.sam,sample1_rep2.sam,...sample1_repM.sam
:begin
"execute_command" := "no"
"command_name" := "cufflinks/2.1.1/cuffdiff"
"list_of_samples_to_compare" := "cuffdiff_input_sample_file"    # options
```

```
 "all" or a text file with sample names(eg: cuffdiff_input_sample_file
)
#If set to "all", all the samples are considered in cuffdiff analysis i
n all-to-all fashion.
#or provide text file to specify specific sample list and groupings for
 the anaysis.please refer to the documentation for more information.
"input_directory" := "aligner_output"
"input_file_type" := "*coordinate*"     # Specify the aligner output file t
ype here. example : sam or bam

#General Options:
"-o" := "output_directory"      # -o/--output-dir write all output files to
this directory [ default: ./ ]
"-L" := "no"    # -L/--labels comma-separated list of condition labels
"--FDR" := "no" # --FDR False discovery rate used in testing [ default:
 0.05 ]
"-M" := "no"    # -M/--mask-file ignore all alignment within transcripts i
n this file [ default: NULL ]
"-b" := "no"    # -b/--frag-bias-correct use bias correction - reference f
asta required [ default: NULL ]
"-u" := "no"    # -u/--multi-read-correct use 'rescue method' for multi-re
ads [ default: FALSE ]
"-p" := "12"    # -p/--num-threads number of threads used during quantific
ation [ default: 1 ]
"--no-diff" := "no"     # --no-diff Don't generate differential analysis fi
les [ default: FALSE ]
"--no-js-tests" := "no" # --no-js-tests Don't perform isoform switching
 tests [ default: FALSE ]
"-T" := "no"    # -T/--time-series treat samples as a time-series [ defaul
t: FALSE ]
"--library-type" := "no"        # --library-type Library prep used for input r
eads [ default: below ]
"--dispersion-method" := "no"   # --dispersion-method Method used to esti
mate dispersion models [ default: below ]
"--library-norm-method" := "no" # --library-norm-method Method used to
normalize library sizes [ default: below ]
```

```
#Advanced Options:
"-m" := "no"    # -m/--frag-len-mean average fragment length (unpaired rea
ds only) [ default: 200 ]
"-s" := "no"    # -s/--frag-len-std-dev fragment length std deviation (unp
aired reads only) [ default: 80 ]
"-c" := "no"    # -c/--min-alignment-count minimum number of alignments in
 a locus for testing [ default: 10 ]
"--max-mle-iterations" := "no"  # --max-mle-iterations maximum iteration
s allowed for MLE calculation [ default: 5000 ]
"--compatible-hits-norm" := "no"        # --compatible-hits-norm count hits co
mpatible with reference RNAs only [ default: TRUE ]
"--total-hits-norm" := "no"     # --total-hits-norm count all hits for norm
alization [ default: FALSE ]
"-v" := "no"    # -v/--verbose log-friendly verbose processing (no progres
s bar) [ default: FALSE ]
"-q" := "no"    # -q/--quiet log-friendly quiet processing (no progress ba
r) [ default: FALSE ]
"--seed" := "no"        # --seed value of random number generator seed [ defau
lt: 0 ]
"--no-update-check" := "no"     # --no-update-check do not contact server t
o check for update availability[ default: FALSE ]
"--emit-count-tables" := "no"   # --emit-count-tables print count tables
used to fit overdispersion [ DEPRECATED ]
"--max-bundle-frags" := "no"    # --max-bundle-frags maximum fragments all
owed in a bundle before skipping [ default: 500000 ]
"--num-frag-count-draws" := "no"        # --num-frag-count-draws Number of fra
gment generation samples [ default: 100 ]
"--num-frag-assign-draws" := "no"       # --num-frag-assign-draws Number of f
ragment assignment samples per generation [ default: 50 ]
"--max-frag-multihits" := "no"  # --max-frag-multihits Maximum number of
 alignments allowed per fragment [ default: unlim ]
"--min-outlier-p" := "no"       # --min-outlier-p Min replicate p value to ad
mit for testing [ DEPRECATED ]
"--min-reps-for-js-test" := "no"        # --min-reps-for-js-test Replicates ne
eded for relative isoform shift testing [ default: 3 ]
"--no-effective-length-correction" := "no"      # --no-effective-length-corr
```

```
ection No effective length correction [ default: FALSE ]
"--no-length-correction" := "no"        # --no-length-correction No length cor
rection [ default: FALSE ]
"-N" := "no"    # -N/--upper-quartile-norm Deprecated, use --library-norm-
method [ DEPRECATED ]
"--geometric-norm" := "no"      # --geometric-norm Deprecated, use --library
-norm-method [ DEPRECATED ]
"--raw-mapped-norm" := "no"     # --raw-mapped-norm Deprecated, use --libra
ry-norm-method [ DEPRECATED ]
"--poisson-dispersion" := "no"  # --poisson-dispersion Deprecated, use -
-dispersion-method [ DEPRECATED ]
#Debugging use only:
"--read-skip-fraction" := "no"  # --read-skip-fraction Skip a random sub
set of reads this size [ default: 0.0 ]
"--no-read-pairs" := "no"       # --no-read-pairs Break all read pairs [ defa
ult: FALSE ]
"--trim-read-length" := "no"    # --trim-read-length Trim reads to be this
 long (keep 5' end) [ default: none ]
"--no-scv-correction" := "no"   # --no-scv-correction Disable SCV correct
ion [ default: FALSE ]
"" := "hg19_refGene.gtf"      # use this f
or transcript model.
#"" := "hg19_refFlat.gtf"     # use this
for gene model.
:end

#cuffmerge:
#cuffmerge takes two or more Cufflinks GTF files and merges them into a


#single unified transcript catalog. Optionally, you can provide the scr
ipt
#with a reference GTF, and the script will use it to attach gene names
and other
#metadata to the merged catalog.
#
#Usage:
```

```
#cuffmerge [Options] <assembly_GTF_list.txt>
#
:begin
"execute_command" := "no"
"command_name" := "cufflinks/2.1.1/cuffmerge"
"input_directory" := "postprocess_output"
"input_file_type" := "transcripts"
"-h" := "no"    # -h/--help Prints the help message and exits
#"--ref-gtf" := "hg19_refFlat.gtf"    #
-g/--ref-gtf An optional "reference" annotation GTF.
"--ref-gtf" := "hg19_refGene.gtf"     # -
g/--ref-gtf An optional "reference" annotation GTF.
"--ref-sequence" := "hg19.fa"      # -s
--ref-sequence <seq_dir>/<seq_fasta> Genomic DNA sequences for the ref
erence.
"-p" := "12"    # -p/--num-threads <int> Use this many threads to merge as
semblies. [ default: 1 ]
"--min-isoform-fraction" := "no"        # --min-isoform-fraction <0-1.0> Disca
rd isoforms with abundance below this [ default: 0.05 ]
"--keep-tmp" := "no"    # --keep-tmp Keep all intermediate files during me
rge
"-o" := "output_directory"      # -o <output_dir> Directory where merged ass
embly will be written [ default: ./merged_asm ]
"list_of_samples" := "cuffmerge_input_sample_file"      # options : "all" or
 a text file with sample names.(eg: cuffmerge_input_sample_file)
# If set to "all", all the samples are considered in analysis
# or provide text file to specify specific sample for anaysis.(please r
efer to the documentation for more information)
:end

#cuffcompare v2.1.1 (exported)
#Usage:
#cuffcompare [-r <reference_mrna.gtf>] [-R] [-T] [-V] [-s <seq_path>]
#[-o <outprefix>] [-p <cprefix>]
#{-i <input_gtf_list> | <input1.gtf> [<input2.gtf> .. <inputN.gtf>]}
#Cuffcompare provides classification, reference annotation mapping and
```

```
various
#statistics for Cufflinks transfrags.
#Cuffcompare clusters and tracks transfrags across multiple samples, wr
iting
#matching transcripts (intron chains) into <outprefix>.tracking, and a
GTF
#file <outprefix>.combined.gtf containing a nonredundant set of transcr
ipts
#across all input files (with a single representative transfrag chosen
#for each clique of matching transfrags across samples).
:begin
"execute_command" := "no"
"command_name" := "cufflinks/2.1.1/cuffcompare"
"input_directory" := "postprocess_output"
"input_file_type" := "transcripts"
"" := "-i"
"list_of_samples" := "cuffcompare_input_sample_file"    #"all" or a text f
ile with sample names.(eg: cuffcompare_input_sample_file)
#If set to "all", all the samples are considered in analysis.
#or provide text file to specify specific sample for anaysis.(please re
fer to the documentation for more information)
"-o" := "output_file"   #set this to "output_file". YAP takes care of nam
ing output file based on samples.
"-r" := "hg19_refGene.gtf"    # -r a set
of known mRNAs to use as a reference for assessing the accuracy of mRN
As or gene models given in <input.gtf>
#"-r" := "hg19_refFlat.gtf"   # -r a se
 of known mRNAs to use as a reference for assessing the accuracy of mR
NAs or gene models given in <input.gtf>
"-R" := "no"    # -R for -r option, consider only the reference transcript
s that overlap any of the input transfrags (Sn correction)
"-Q" := "no"    # -Q for -r option, consider only the input transcripts th
at
#overlap any of the reference transcripts (Sp correction);
#(Warning: this will discard all "novel" loci!)
"-M" := "no"    # -M discard (ignore) single-exon transfrags and reference
```

```
 transcripts
"-N" := "no"    # -N discard (ignore) single-exon reference transcripts
"-s" := "hg19.fa"  # -s <seq_path>
can be a multi-fasta file with all the genomic sequences or
#a directory containing multiple single-fasta files (one file per conti
g);
#lower case bases will be used to classify input transcripts as repeats

"-e" := "no"    # -e max. distance (range) allowed from free ends of termi
nal exons of reference
#transcripts when assessing exon accuracy (100)
"-d" := "no"    # -d max. distance (range) for grouping transcript start s
ites (100)
"-p" := "12"    # -p the name prefix to use for consensus transcripts in t
he
#<outprefix>.combined.gtf file (default: 'TCONS')
"-C" := "no"    # -C include the "contained" transcripts in the .combined.
gtf file
"-G" := "no"    # -G generic GFF input file(s) (do not assume Cufflinks GT
F)
"-T" := "no"    # -T do not generate .tmap and .refmap files for each inpu
t file
"-V" := "no"    # -V verbose processing mode (showing all GFF parsing warn
ings)
:end
```

B.5 References

B.5.1 Preprocess Packages:

(a) FastQC - http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

(b) Fastq screen - http://www.bioinformatics.babraham.ac.uk/projects/fastq_screen/

(c) FASTX-Toolkit - http://hannonlab.cshl.edu/fastx_toolkit/

- FASTQ/A Barcode Splitter
- FASTQ/A Clipper
- FASTQ Quality Filter
- FASTQ Quality Trimmer

B.5.2 Aligner algorithms:

(a) Burrows-Wheeler Aligner (BWA) - http://bio-bwa.sourceforge.net/

(b) Bowtie aligner - http://bowtie-bio.sourceforge.net/

(c) Tophat - http://tophat.cbcb.umd.edu/

(d) Bowtie2 aligner - http://bowtie-bio.sourceforge.net/bowtie2/

(e) SAMtools(for sorting) - http://samtools.sourceforge.net/

B.5.3 Post-processing Packages:

(a) Picard Tools - http://picard.sourceforge.net/

- Multiple Metrics
- Alignment Metrics
- Quality Score Distribution
- Mean Quality by Cycle
- GC Bias Metrics
- Insert Size Metrics
- RNA Seq Metrics
- Mark Duplicates
- Targeted Pcr Metrics
- Calculate Hs Metrics

(b) HTSeq - http://www-huber.embl.de/users/anders/HTSeq/

(c) Cufflinks - http://cufflinks.cbcb.umd.edu/

- Cuffdiff
- Cuffmerge
- Cuffcompare

(d) Exon Counts

(e) Junction Counts

(f) BEDtools(for junction counts) - http://code.google.com/p/bedtools/

B.5.4 Index files

(a) UCSC reference genome - http://genome.ucsc.edu

(b) Broad index files