# Wgs\_identification\_manual

Ramon van der Stoop

Contents

[Wgs\_identification\_manual 1](#_Toc70007156)

[Current state of the pipeline 3](#_Toc70007157)

[How to use 3](#_Toc70007158)

[Directory / file structure 3](#_Toc70007159)

[Code Discription 4](#_Toc70007160)

[<scripts/runall.sh> - Executed manually 4](#_Toc70007161)

[<scripts/fastqc.sh> - Executed manually 4](#_Toc70007162)

[<scripts/trimmomatic.sh> - Executed manually 5](#_Toc70007163)

[<scripts/kraken2\_runall\_snakemake.sh> - Executed by snakemake 5](#_Toc70007164)

[<scripts/kraken2\_isolateGS\_snakemake.sh> - Executed by snakemake 5](#_Toc70007165)

[<scripts/clark\_runall\_snakemake.sh> - Executed by snakemake 6](#_Toc70007166)

[<scripts/clark\_classifyall\_snakemake.sh> - Executed by snakemake 6](#_Toc70007167)

[<scripts/kaiju\_runall\_snakemake.sh> - Executed by snakemake 6](#_Toc70007168)

[<scripts/kaiju\_classifyall\_snakemake.sh> - Executed by snakemake 6](#_Toc70007169)

[<scripts/kaiju\_combine\_snakemake.py> - Executed by snakemake 6](#_Toc70007170)

[<scripts/combine2table.sh> - Executed by snakemake 6](#_Toc70007171)

[<scripts/append.py> - Executed manually 7](#_Toc70007172)

[<scripts/run\_append.sh> - Executed by manually 7](#_Toc70007173)

## Current state of the pipeline

* FastQC and Trimmomatic are automized, but not intergrated in the snakemake logic.
* One trimmed .fastq file can be processed to a table with the top 5 results of all 3 tools in a single .csv file, automized with Snakemake.
* By executing the runall.sh script all files are ran using the snakemake pipeline.
* With the run\_append.sh script alle results can be combined in a single table in .csv format.
* Currently all intermediate files are stored in the main folder.
* CLARK only works with unzipped files. So unzipped files are needed for the pipeline to run. (extract\_all.sh unzips all in the folder)

## How to use

Here are examples found on how to use the code executed from the main directory:

Execute FastQC on all files, directories are defined in the code:

* Bsub -q bio -n 4 -o log/fastqc\_log.txt ./scripts/fastqc.sh
  + Be sure to have the conda env activated.

Execute Trimmomatic on all files in, directories are defined in the code:

* Bsub -q bio -n 8 -o log/trimmomatic\_log.txt ./scripts/trimmomatic.sh
  + Be sure to have the conda env activated.

Identify all .fastq files in the </input> folder with Kraken, CLARK, and Kaiju and summarize their results by using snakemake:

* ./scripts/runall.sh
  + The individual settings can be configured in the code under ‘#settings’

For identifying a single file:

* Bsub -q bio -n 6 -o log/combined/combined\_<samplenumber>.txt snakemake results/combined\_<samplenumber>.csv -j
  + For testing a dry-run to use the -n option in snakemake command.

By placing run\_appended.sh, append.py and appended\_df.csv in the </results> folder, all results can be combined into one single table. This code is executed form the </results> folder.

* Bsub -q bio -o append\_log.txt ./run\_append.sh

## Directory / file structure

The main folder is called <wgs\_identification> This is where all files and directories are found.Most code is also activated from this folder. (there are exceptions to this rule: )

Below all folders are described:

* </envs>: All yaml files are supposed to be stored here. In the current state there is only one .yml file used called <WGSpipeline\_ramon\_env.yml>
* </input>: Files that are supposed to go through the pipeline (tools 🡪 top5table) are to be placed here after trimming with trimmomatic.
* </input/unpaired>: The unpaired results of trimmomatic are stored here.
* </log>: All logs are stored here
* </qc\_clean>: All fastQC results of the trimmed .fastq files are stored here.
* </qc\_raw>: All fastQC results of the raw .fastq files are stored here.
* </raw>: The raw .fastq files can be stored here.
* </results>: The results of the pipeline are stored
* </scripts>: all scripts are stored here
* <NexteraPE-PE.fa>: Contains adapter for trimmomatic
* <Snakefile>: Contains all snakemake logic

## Code Discription

All code is run from the main folder </wgs\_identification> with the exception of a few scripts

### <scripts/runall.sh> - Executed manually

Used to identify all samples in the input folder

* Prestring: required to strip leading string to end up with a string containing the monsternr only.
* Poststring: requied to strip the trailing string to end up with a string containing the monsternr only.
* Monsternr: contains just the string for the sample number.
* To use snakemake the command ‘snakemake’ is used, followed by the desired output file, with the correct naming scheme.

### <scripts/fastqc.sh> - Executed manually

Executes fastQC on all .fastq files in the designated folder and puts the results in another designated folder

* There is an example present in the code how it could be executed
* qc\_clean: the directory that contains trimmed .fastq files
* qc\_raw: the directory that contains raw .fastq files
* qc\_folder: under #settings qc\_folder can be found. This variable accepts either ‘qc\_clean’ or ‘qc\_raw’. This decides what folder becomes the output. \*Note, this could be upgraded by converting it to a passable argument.
* The loop loops through all files in the qc\_folder defined folder.

### <Snakefile> - Executed manually or by <scripts/runall.sh>

Contains all the logic to generate a combined table of the top 5 results of kraken, clark and kaiju.

Each rule has the following:

* Input: The designated filenaming structure how it receives inputfiles
* Output: The designated filenaming structure how it outputs a file
* Shell: The command that the output with the input
* Params: monsterID=: contains the sample ID to be used as argument in the shell
* Conda: contains the yaml file with the programs required to excecute the code.

### <scripts/trimmomatic.sh> - Executed manually

Executes Trimmomatic on all .fastq files in the designated directory and delivers output in the </input> folder to be processed downstream. Trimmomatic prefers 4 threads, so a mupltiple of 4 is recommended.

* There is an example presen in the code how it could be executed
* In the settings:
  + Inputfile: the directory of the inputfiles is defined
  + Prestring: required to strip leading string to end up with a string containing the monsternr only.
  + Poststring: requied to strip the trailing string to end up with a string containing the monsternr only.
* Loop:
  + Monsternr: contains just the string for the sample number
  + j: exchanges the R1 (forward) filename with R2 (reversed) filename and stores the string in variable j
  + output settings are stored here
  + adapters: adapterfile for trimmomatic
  + trimmomatic command, for more information consult trimmomatic manual.
  + Trimmomatic settings are set to default and can be customized in this file

### <scripts/kraken2\_runall\_snakemake.sh> - Executed by snakemake

Executes kraken2 through the snakemake pipeline

* Database: the directory for the database can be configured here
* Inputfile: the first argument in the shell command, contains the name of the inputfile
* Outputfile: the second argument in the shell command, contains the name of the outputfile
* Monsternr: the third argument in the shell command, required to name to name files correctly
* Threads: the maximum amount of threads can be defined here. More than 6 > diminishing returns
* I: inputfile forward
* J: inputfile reverse
* kraken2: kraken2 command which uses the settings mentioned above.

### <scripts/kraken2\_isolateGS\_snakemake.sh> - Executed by snakemake

Due to the output of kraken containing all taxonomic ranks and only genus, species, and possibly sub-species are desired this script was made. By simply using a grep all lines containing a genus or species result is stored in a new file.

Kraken also has its output in tab delimited form instead of comma delimited, this script also converts .tsv to .csv.

### <scripts/clark\_runall\_snakemake.sh> - Executed by snakemake

Executes CLARK through the snakemake pipeline

The settings are similar in concept to Kraken2. For more details refer to CLARK manual.

### <scripts/clark\_classifyall\_snakemake.sh> - Executed by snakemake

CLARK tool has no classification build-in, this is a separate tool. It simply requires an input, output, and database directory.

### <scripts/kaiju\_runall\_snakemake.sh> - Executed by snakemake

Executes Kaiju through the snakemake pipeline

The settings are similar in concept to Kraken2. For more details refer to Kaiju manual.

There is a 5 minute sleep build-in prevent the pipeline to continue with unfinished files. 5 minutes is an arbitrary amount.

### <scripts/kaiju\_classifyall\_snakemake.sh> - Executed by snakemake

Kaiju tool has no classification build-in, this is a separate tool. It simply requires an input, output, and database directory. The limitation in this tool is the fact that it can only classify one defined rank. Because both genus and species are desired the kaiju2table script is run twice and later combined in the kaiju\_combine\_snakemake.py script

A 30 second sleep is build-in to prevent the pipeline to continue with unfinished files. 30 seconds seems to be enough to prevent further issues.

### <scripts/kaiju\_combine\_snakemake.py> - Executed by snakemake

Python script which uses pandas to combine the previously mentioned 2 files and combines them into a single table.

### <scripts/combine2table.sh> - Executed by snakemake

Python script which uses pandas to combine all 3 results (from kraken, CLARK and kaiju) into one single report csv file.

* Cla\_file: All 3 files are imported and stored in their cla\_file, kra\_file and kai\_file variables respectively.
* Cla\_df: By using pd\_read.csv, pandas recognizes the csv format.
* Cla\_df.columns: The desired columns are renamed for uniformity
* Combined\_format: the combined format is defined here and stored in ‘cdf’ variable
* Cla\_sort: all dataframes are sorted by their designated column in descending order
* Kra\_sort: the kraken data contains ‘white spaces’ in front of the ‘result name’ for uniformity these are removed here.
* Cla\_head5: in this variable the top 5 results of the name and result are stored
* Cdf = pd.concat: all 3 x\_head5 files are concatenated here
* monsterID: a samplenumber ID is inserted in the dataframe
* cdf.to\_csv: the dataframe is exported to .csv format.

### <scripts/append.py> - Executed manually

Appends two tables with python with pandas. Appends the the second argument to the first argument. An empty appened\_df.csv is required. (one saved as appened\_df\_template.csv)

### <scripts/run\_append.sh> - Executed by manually

For easier execution of append.py. place in result folder with appended\_df.csv. adds all combined results to the appended\_df.csv for a single result file.