PIMGAVir Pipeline V1.1

Discovery and Molecular Characterization of Pathogens

February 2022

Table of Contents

PIMGAVir Pipeline

The main goal of the PIMGAVir pipeline is to provide the user with a preliminary taxonomic classification of the data to be analyzed. In literature, three are the more used methods to this scope: reads-based, assembly-based, and clustering-based. PIMGAVir pipeline gives the user the opportunity to analyze the data using one, more, or all the strategies in parallel. Figure 1 shows the logical flow of PIMGAVir at a high level. As a preliminary step, the pre-processing task is executed to trim the raw data and remove contaminants. Then, according to the user option, the reads_filtering (filtering out reads "probably" not belonging to desired taxa) task is executed or not.

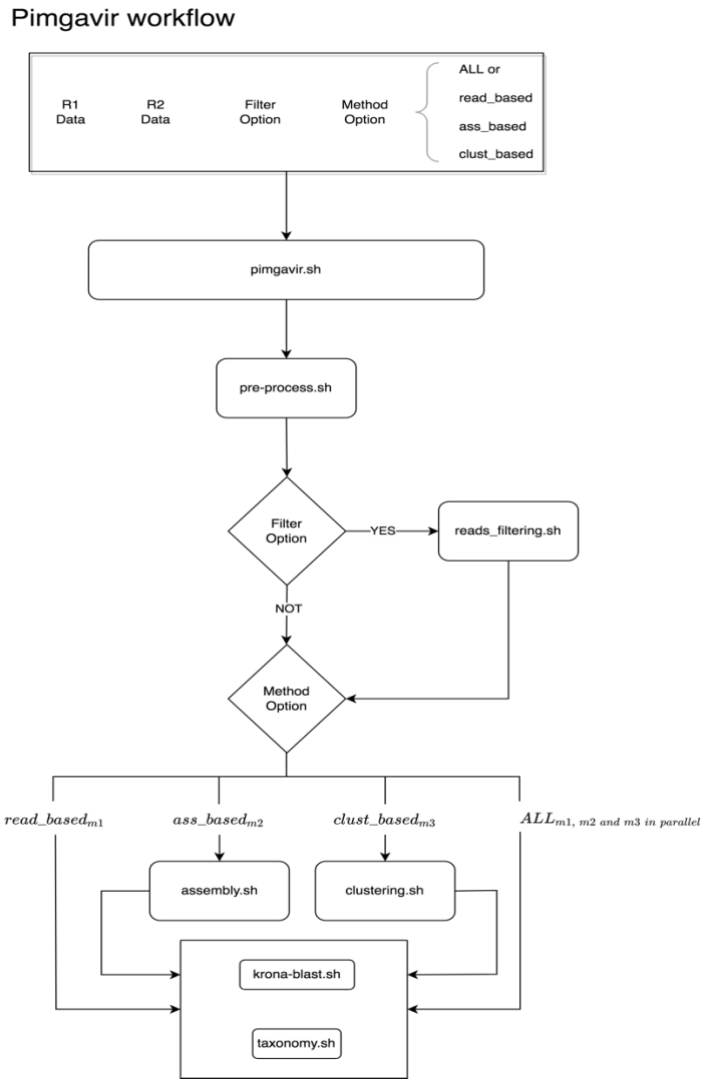

Figure 1 PIMGAVir pipeline workflow

Subsequently, PIMGAVir runs the method of investigation chosen, which will perform the next steps:

- Read_based will make the taxonomic classification starting from the file obtained by the pre-process/reads_filtering task

- Ass_based, moving from the file obtained by the pre-process/reads_filtering task, will make the taxonomic classification
- Clust_based will perform the clustering of the reads gained from the pre-process/reads_filtering task, create the phylogenetic tree and make the taxonomic classification

Note that the user can run the pimgavir.sh script with more than one "strategy" option at the same time. For example, the command

```
pimgavir.sh R1.fq R2.fq SampleName 24 —read_based —ass_based —filter
```

will run the pipeline to execute both the strategies, —read_based and —ass_based. Coming sections describe in detail every module of PIMGAVir.

Pre-process task

The pre-process.sh shell script takes care of the pre-processing task. As Figure 2 depicts, the current task accomplishes two goals: the trimming of the raw data and the removing of ribosomal contaminants. In detail:

1. Trimming is performed using trim_galore in multithread mode. Sensible parameters passed to trim_galore are:
   - –length 80, to discard reads became shorter than 80 bp
   - –paired R1 R2, to perform length trimming of quality/adapter/RRBS trimmed reads for the paired-end R1, R2
   - -q, to trim reads whose quality is less than 30, in addition to adapter removal
2. Ribosomal removing, performed through sortmeRNA in multithread mode and using the SSR138 and SLR138 ribosomal databases as references. Sensible parameters passed to sortmeRNA are:
   - –num_alignments, to report the best (n. 1) alignment
   - –paired_out, output reads into Non-Aligned file

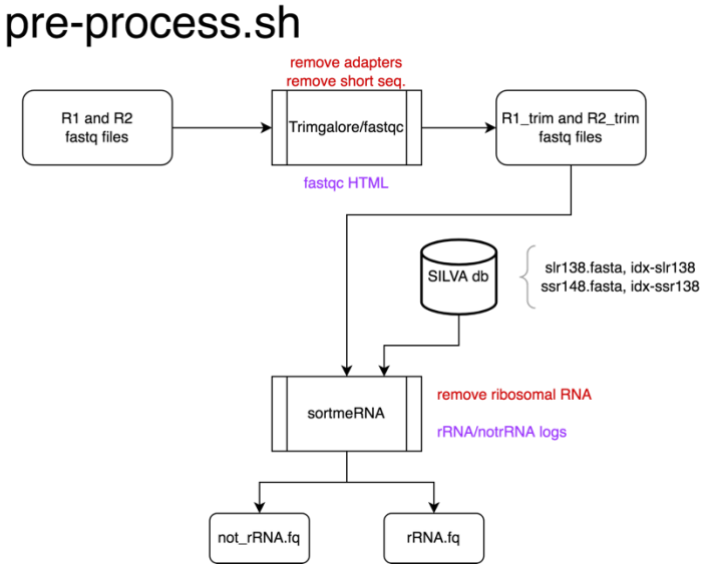

Figure 2 Pre-process task

The pre-processing step will be skipped if the SAMPLE_NAME_not-rRNA.fq file already exists in the working directory. This feature is useful to don't repeat the pre-processing task in case it has already done with the same samples.

Filtering option

The reads-filtering.sh shell script takes care to filter out the reads belonging to undesired kingdoms. To activate the filtering feature, the user has to use the "--filter" option and create a text file named "unwanted.txt" containing the list of unwanted kingdoms or species or organisms, as reported in the example in Figure 3:

```
Eubacteria
Achaeabacteria
Plantae
```

Figure 3 unwanted.txt file

As shown in Figure 4, the reads-filtering.sh shell script will take as input the trimmed reads and return the reads not classified in the unwanted list after comparing them to the RefSeq non-redundant protein database using diamond.

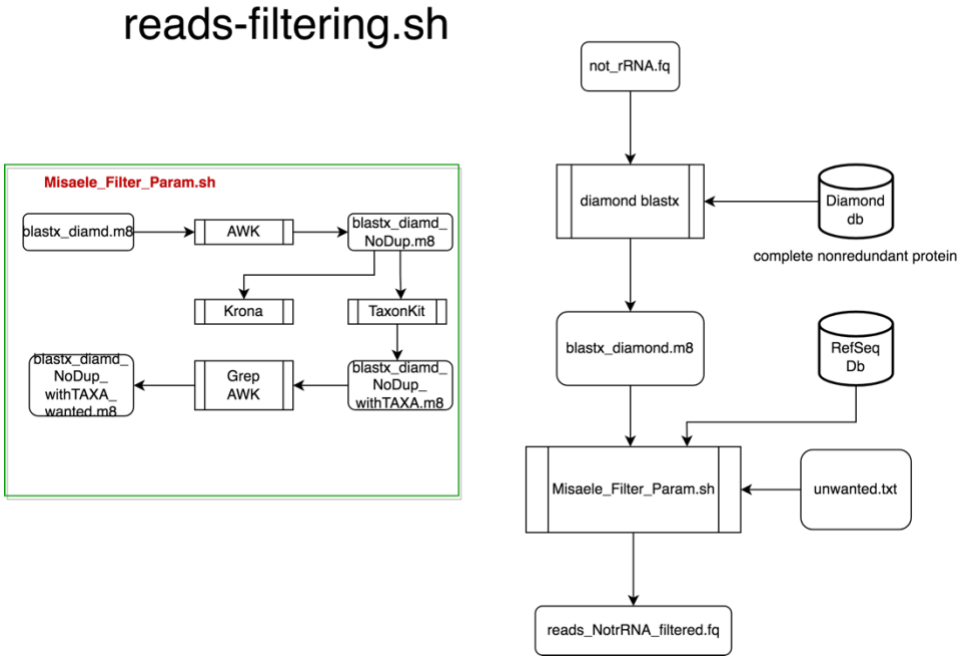

Figure 4 read-filtering.sh bash script and Misaele_Filter_Param.sh

In detail, the Misaele_Filter_Param.sh will also give back a preliminary taxonomy classification from the non-duplicated reads. The read-filtering step will be skipped if the readsNotrRNA_filtered.fq file already exists in the working directory. This feature is useful to don't repeat the read-filtering task in case it has already done with the same samples and using the same unwanted.txt filter file.

read_based function

Once invoked as an option of pimgavir.sh, it will directly execute the taxonomy.sh task using the filtered/not_filtered fastq file as input, depending on whether the filter option value.

Taxonomic classification task

The taxonomy.sh shell script will execute the taxonomic classification of the reads in the fasta file used as input. In detail, the task will accomplish the classification using both the kraken2/KrakenViral DB and kaiju/KaijuViral DB, as a first step. Then the Krona application produces the HTML files to visualize the obtained results. Figure 5 shows the procedure.

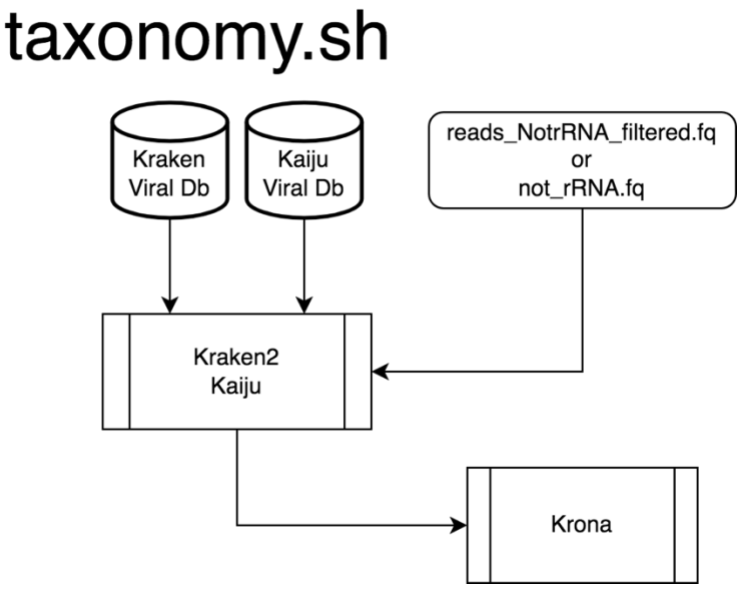

Figure 5 Taxonomy.sh shell script

Blast classification and Krona visualization task

The krona-blast.sh shell script will execute the taxonomic classification of the reads in the fasta file used as input. In detail, the task will accomplish the classification using NGS viral references repository after the blastn operation and visualize the results using Krona. Figure 6 shows the steps followed by the script.

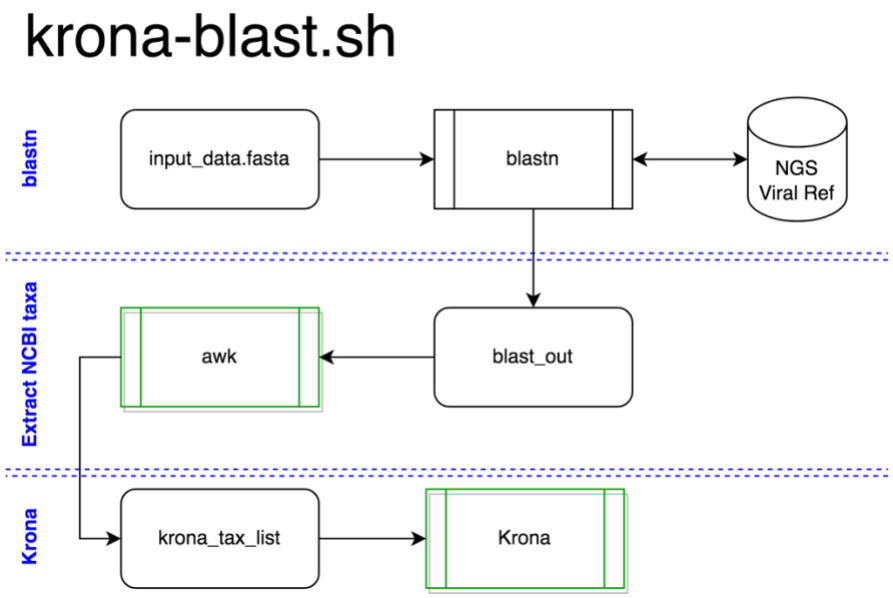

Figure 6 krona-blast.sh bash script

ass_based function

Once invoked as an option of pimgavir.sh, it will execute the assembly.sh, shell script using the filtered/not_filtered fastq file as input, depending on whether the filter option value. The bash script will perform the following sub-tasks:

1. Produce the de-novo assembly using both the MegaHit and Spades applications
2. Execute the contigs analysis of both assemblies using Quast
3. Fix possible misassembles from both assemblies (using bowtie, samtools, and pilon)
4. Create the gene annotation for both assemblies using Prokka

It is possible to visualize the gbk files (annotation files) using art application, while the use of a common browser is sufficient to view the reports produced by Quast. Figure 7 reports in detail the mentioned procedure.

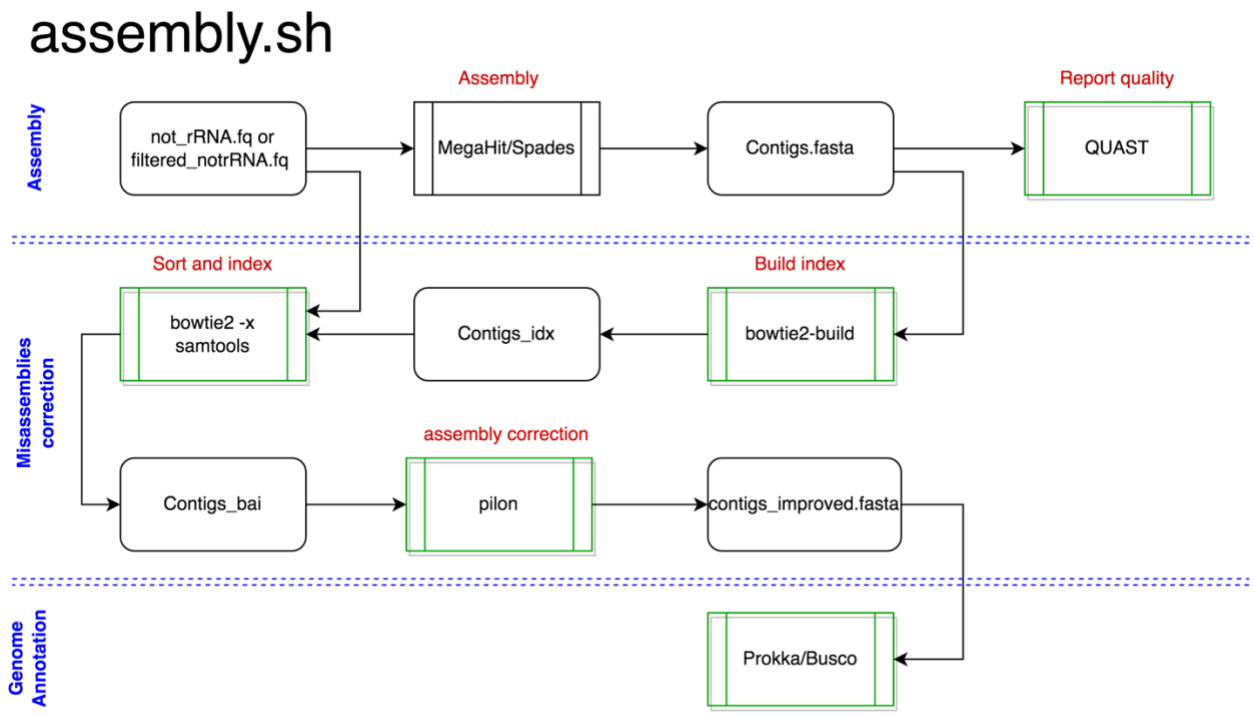

Figure 7 Assembly.sh bash script

clust_based function

Once invoked as an option of pimgavir.sh, it will execute the clustering.sh, shell script using the filtered/not_filtered fastq file as input, depending on whether the filter option value. The bash script will perform the following main steps:

1. Data preparation, arrange the data file the be used as input file from vsearch
2. De-replication, perform the dereplication step both on the merged files and on the dataset
3. Pre-clustering, identify the centroids useful for the clustering step with 95% of identity threshold
4. Remove chimeras
5. OUT-clustering, perform the OUT clustering from the non-chimera data file with an identity threshold equal to 95%.

Figure 8 shows a detailed perspective of the steps executed by clustering.sh.

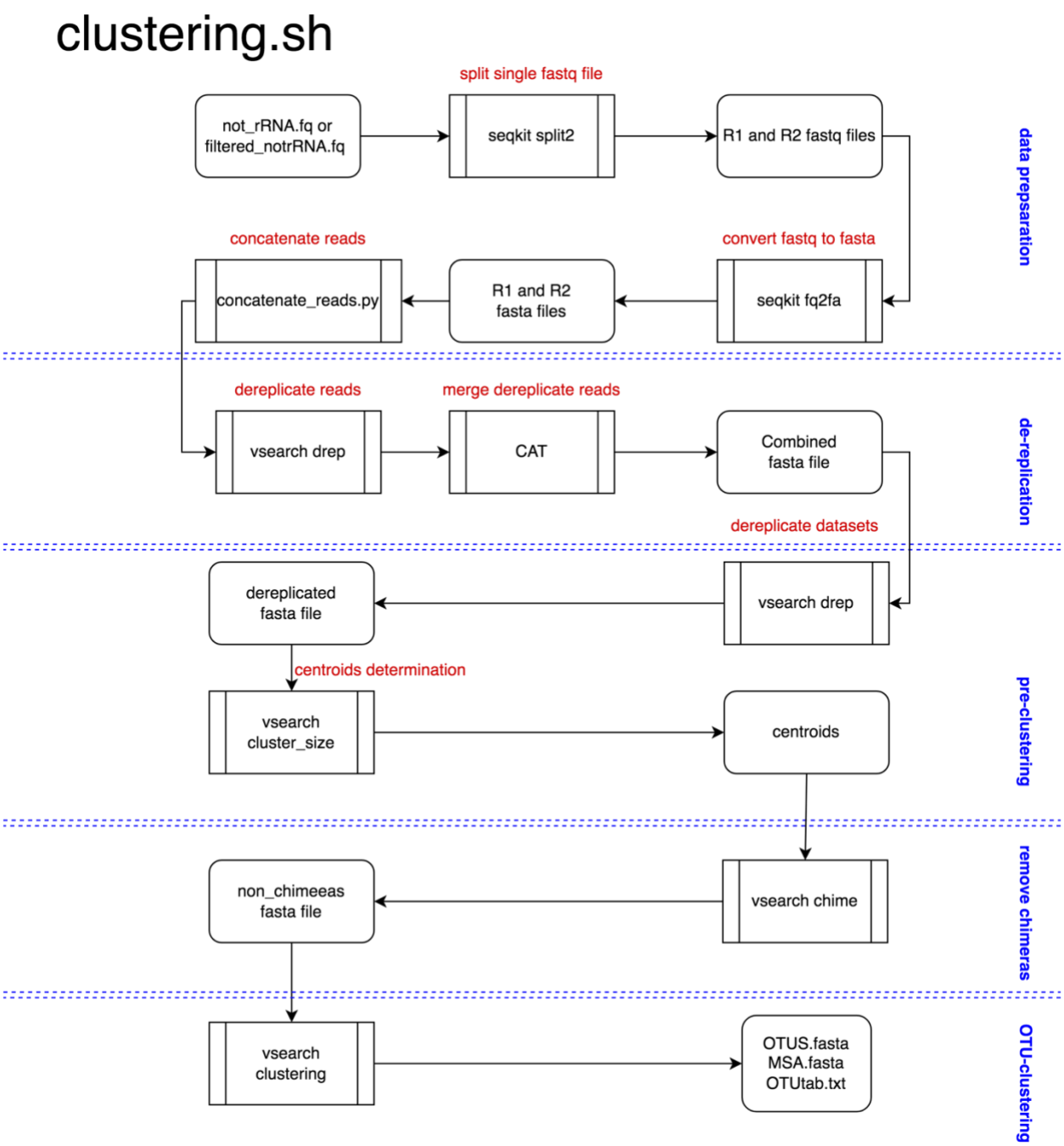

Figure 8 clustering.sh bash script

grouping_reads.sh

Sometimes the user could need to group together the results coming from the Kraken-Krona output files. The grouping_reads.sh shell script allows the user to group together the reads/contigs according to their family or genus. The script uses the genus/family value as a key and creates one

file for every set of reads/contigs belonging to the same genus or family. The user will choose the key to use for grouping at running time. Figure 9 shows its main steps.

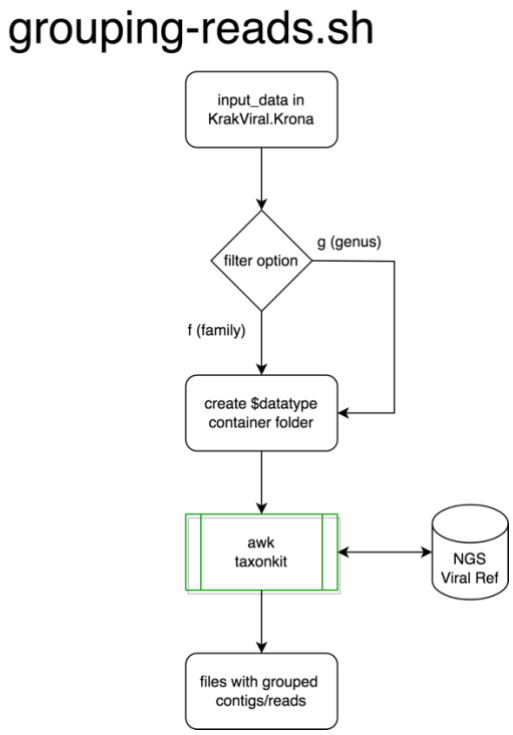

Figure 9 grouping-reads.sh shell script

Running the methods independently

The user has the freedom to run every one of the mentioned scripts as an autonomous process, as long as the input format is respected. The table reported in Figure 10 shows the expected input for every script, once apart:

Script Name	Usage	Parameters	Meaning
pre-process.sh	pre-process.sh R1.fq R2.fq Sample Threads	R1.fq, R2.fq	input fastq data file
		Sample	name used to create the output files
		Threads	number of threads to use
read-filtering.sh	read-filtering.sh DiamondDB Threads InputDB OutDiamondDB PathToRefSeq UnWanted	DiamondDB	Path to diamond Db
		Threads	number of threads to use
		OutDiamondDB	fastq file output from blasting diamond
		PathToRefSeq	Path to Reference Sequences DB
		UnWanted	Name of text file containg UNWANTED kingdonm
assembly.sh	assembly.sh Data.fasta FolderName Threads	Data.fasta	Name of the input file in fasta format
		FolderName	Name of the folder where to save results
		Threads	number of threads to use
clustering.sh	clustering.sh Data.fasta FolderName Threads	Data.fasta	Name of the input file in fasta format
		FolderName	Name of the folder where to save results
		Threads	number of threads to use
taxonomy.sh	taxonomy.sh Data.fasta FolderName Threads Prefix	Data.fasta	Name of the input file in fasta format
		FolderName	Name of the folder where to save results
		Threads	number of threads to use
		Prefix	Prefix to use for saving the output file names
krona-blast.sh	krona-blast.sh Data.fasta FolderName Threads	Data.fasta	Name of the input file in fasta format
		FolderName	Name of the folder where to save results
		Threads	number of threads to use
grouping-reads.sh	grouping-reads.sh KrakViral.Krona [f|g] TAG	KrakViral.Krona	Output file from taxonomy task
		[f|g]	This is the key value for grouping the reads/contigs. f indicates family, g stays for genus
		TAG	The TAG value to distinguish where the data come from (OUT/read/contigs/etc)

Figure 10 Expected arguments for every bash script

Running pimgavir

Suppose to run the pimgavir pipeline using the following files as input:

```
-rw-rw-r-- 1 emilio emilio 383M 9 月 28 12:03 Pool-3-1_FKDL210225623-1a-AK25938-AK25939_1.fq.gz
-rw-rw-r-- 1 emilio emilio 391M 9 月 28 12:04 Pool-3-1_FKDL210225623-1a-AK25938-AK25939_2.fq.gz
```

If we call the pimgavir.sh without indicating any parameters, the following message will be shown, indicating which parameters the pipeline is expecting:

```
emilio@Alienware:~/Downloads/veryfasttree-master$ pimgavir.sh
Error. Not enough arguments.
Usage pimgavir.sh R1.fastq.gz R2.fastq.gz SampleName NumbOfCores ALL|[--read_based --ass_based --clust_based] [--filter]
```

The user can instruct the pipeline to execute one of the following strategies using the appropriate option:

- --read_based, will run the pipeline under the “read based” strategy
- --ass_based, will run the pipeline under the “assembly based” strategy
- --clust_based, will run the pipeline under the “clustering-based” strategy

As an example, the user could run the pipeline with the following command. Note the "time" command is used to get the time used by the command to end.

```
time pimgavir.sh Pool-3-1_FKDL210225623-1a-AK25938-AK25939_1.fq.gz Pool-3-
1_FKDL210225623-1a-AK25938-AK25939_2.fq.gz FKDL210225623 24  -read_based --filter
```

The next sections will report some technical information that could be helpful to the user, such as the list of files created, the running time, or specific requirements according to the involved shell script.
Independently of which strategy the user will choose, the pre-processing task is executed running the pre-process.sh shell script.

pre-proprocess.sh

The following files will be created:

1. Log files: pimgavir.log, pre-process.log, trim-galore.log, and FKDL210225623_rRNA.fq (sortmeRNA log file)
2. Trimgalore/FastQC report files: Pool-3-1_FKDL210225623-1a-AK25938-AK25939_1.fq.gz_trimming_report.txt, Pool-3-1_FKDL210225623-1a-AK25938-AK25939_2.fq.gz_trimming_report.txt, Pool-3-1_FKDL210225623-1a-AK25938-AK25939_1_val_1_fastqc.html, Pool-3-1_FKDL210225623-1a-AK25938-AK25939_2_val_2_fastqc.html
3. sortmeRNA_wd: folder containing kvdb and readb sub-folders, symbolic link idx -> /mnt/NTFS/NGS-DBs/SILVA/idx/ (save time avoiding to re-create the idx of SILVA db)
4. Out data files: FKDL210225623_not_rRNA.fq, FKDL210225623_rRNA.fq

The time required is reported:

```
real    27m51.073s
user    322m21.479s
sys     2m1.279s
```

The size of the created files is:

```
3.4G    FKDL210225623_not_rRNA.fq
1.8G    FKDL210225623_R1_trimmed.fq
1.8G    FKDL210225623_R2_trimmed.fq
98M     FKDL210225623_rRNA.fq
```

In case the `--filter` has been expressed, the pipeline will execute the reads-filtering.sh and Misaele_Filter_Param.sh scripts.

reads-filtering.sh

The read-filtering.sh bash script needs the list of undesired species or organisms reported in a text file named unwanted.txt (as the next example) and placed in the same location as the input files:

```
Eubacteria
Achaeabacteria
Plantae
```

Figure 11 unwanted.txt text file

The following files will be created:

1. Log files: diamond.log, reads-filtering.log
2. Out data files: blastx_diamond.m8

Misaele_Filter_Param.sh

The following files will be created:

1. Log files: Misaele_Filter_Param.log
2. Out data files in m8 format: blastx_diamond_NoDup.m8, blastx_diamond_NoDup_wanted.m8, blastx_diamond_NoDup_withTaxa.m8, blastx_diamond_NoDup_withTaxa_wanted.m8
3. Out data files in html format: NoDup.taxonomy.krona.html (taxonomic classification before filtering), WantedReads.taxonomy.krona.html (taxonomic classification after filtering)

The required time for executing both scripts is equal to:

```
real    121m52.584s
user    1432m2.949s
sys     48m18.972s
```

In case the user expressed the `--ass_based` option, the pipeline will execute the assembly.sh shell script.

assembly.sh

The following files will be created:

1. Log files: assembly.log
2. Out data files: assembly_based folder (results container)

The assembly-based folder is a container of the results after the assembly operation. The figure below shows its main structure.

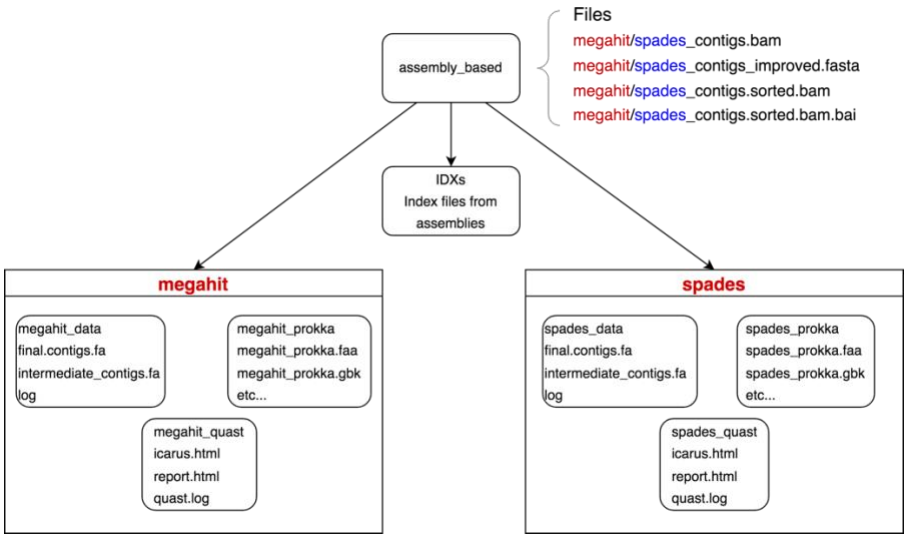

Figure 12 assembly_based data structure

The time required is:

```
real    1m45.927s
user    21m58.417s
sys     2m20.214s
```

The *assembly-based-taxonomy* folder is a container of the results after the assembly taxonomy operation.
It will contain the following files:

krakViral.krona.html_MEGAHIT, krakViral.krona.html_SPADES,
reads_kaiju.kron.html_MEGAHIT, reads_kaiju.kron.html_SPADES

The *krona-blast.sh based on the assembly* will create two folders, one for each assembly: assembly-based-MEGAHIT-KRONA-BLAST, assembly-based-SPADES-KRONA-BLAST
Every folder will contain the following files, obtained from the relative assembly: blastn.out, krona-blast.log, krona_out.html, krona_stderr, krona_stdout krona_tax.lst

In the case of the user expressed the --clust_based option, the pipeline will execute the clustering.sh shell script.

clustering.sh

The following files will be created:

1. Log files: clustering-based.log
2. Out data files: clustering-based folder (results container)

The clustering-based folder will contain the otus.fasta file and the sub-folder named readsNotrRNA_filtered.fq.split within the files coming from the clustering task:

1. Fasta files: Combined.fasta, derep_Concatenated_Unmerged.fasta, derep_Forward.fasta, Forward.fasta, preclustered.fasta, Concatenated_Unmerged.fasta, derep.fasta, derep_Reverse.fasta, nonchimeras.fasta, Reverse.fasta, MSA.fa

2. UC files: clustered.uc, combined.uc, Concatenated_Unmerged.uc, Forward.uc, Reverse.uc
3. Other files: otutab.txt, otu.biom

The time required is:

```
real    0m44.192s
user    2m15.007s
sys     0m6.057s
```

The clustering-based-taxonomy folder is a container of the results after the clustering taxonomy operation.
It will contain the following files:

1. HTML files: krakViral.krona.html_OTU, reads_kaiju.kron.html_OTU
2. OUT files: krakViral_class.out_OTU, krakViral.out_OTU, krakViral_report.out_OTU, krakViral_unclass.out_OTU, readskaiju.out_OTU

The *clustering-based-KRONA-BLAST* will contain the following files:

1. HTML files: krona_out.html
2. OUT files: blastn.out, krona_stdout, krona_tax.lst
3. Other files: krona-blast.log, krona_stderr

The read-based-taxonomy folder will contain the taxonomic classification obtained directly from the reads (filtered or not).
The folder will contain the following files:

1. HTML files: krakViral.krona.html_READ, reads_kaiju.kron.html_READ
2. OUT files: krakViral_class.out_READ, krakViral.out_READ, krakViral_report.out_READ, krakViral_unclass.out_READ, readskaiju.out_READ

grouping-reads.sh

Being accomplished the taxonomic classification (regardless of which strategy has been run), the user can group into the same file the organisms sharing the same genus or family. In detail, taking as input the file text from the Kraken blast (with krona taxonomy already done) and the desired "key" of grouping (by genus or by family), the grouping-reads.sh shell script will create one file for each "key" value containing all the reads/contigs belonging to the same "key" value. Once called without any option, the script will print out the following message:

```
Error. Not enough arguments.
Usage grouping-reads.sh InputFile [--f/--g]
InputFile must be in KrakViral.Krona format [ReadId  TaxId] // TaxId==0 stays for unclassified
[--f/--g] It can be --f (family) or --g (genus)
```

The script will take as input the file containing the taxonomic classification from KrakViral.Krona and as option --f (if the user wishes to group the read sharing the same family) or --g (if the user wishes to group the read sharing the same genus). Depending on the user option (--f/--g), the script

will create the folder family/genus containing one file for each family/genus identified in the KrakViral.Krona input file. Every file will store the reads/contigs sharing the same family/genus.

The following files will be created:

1. Log files: grouping-reads.log
2. Out data files: grouping-based folder (results container)

Required packages

PIMGAVir pipeline uses a collection of bioinformatics packages to perform. Table 1 reports the list of needed packages, the shell script using them, and the linked database, while Figure 13 shows the system architecture of the pipeline. Note that the working directory will be the same as where the input files (a couple of fastq files) are placed.

Table 1 PIMGAVir packages, scripts and DBs

Package name	Shell script	Database
trim_galore, sortmeRNA	pre-process.sh	silvadb
diamond	reads-filtering.sh	diamond
kraken2, kaiju2krona, ktImporttext	taxonomy.sh	krakenViral, kaiju
megahit, seqkit, metaspades.py, quast.py, bowtie2-build, bowtie2, samtools, pilon, Prokka, art	assembly.sh	
python3, concatenate_reads.py, vsearch, seqkit	clustering.sh	
blastn, awk, krona	krona-blast.sh	blastdb
awk, taxonkit	grouping-reads.sh	RefSeq
awk, ktImportTaxonomy, taxonkit, seqtk,	Misaele_Filter_Param.sh	RefSeq

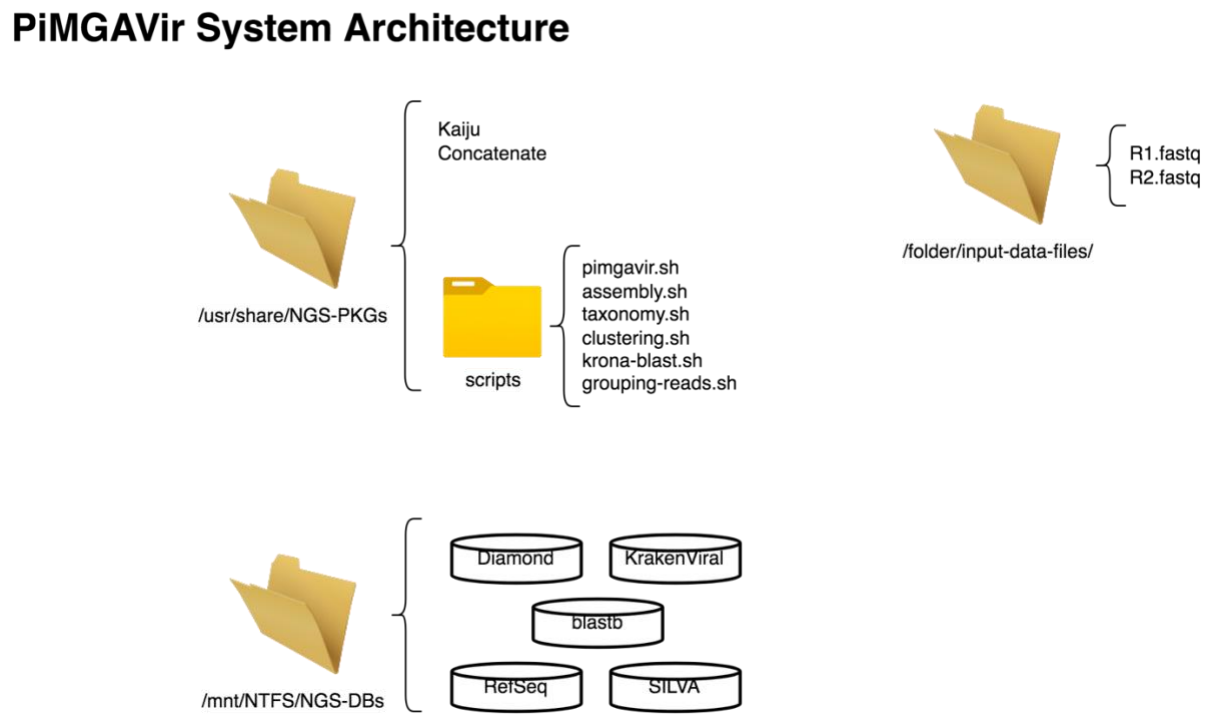

Figure 13 PIMGAVir System Architecture