PIMGAVir User Manual

The current document drives the user to a quick usage of PIMGAVir pipeline. The document is organized as a “use-of-case” to be easier to follow.

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 1 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 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)