NARODOWY INSTYTUT ZDROWIA PUBLICZNEGO
PAŃSTWOWY INSTYTUT BADAWCZY

PleEpiSeq Phylogenetic Pipeline Documentation

Michał Kadlof, Michał Łaźniewski

The document provides technical documentation for the bioinformatics processing protocols (also known as pipelines) developed in the PlEpiSeq project. These protocols are designed for the automatic analysis of multiple samples to provide an input for microreact program. It covers the architecture, installation and execution procedures, the content of the Nextflow modules, and details of the validation tests.

Spis treści

1. Quick start

a. Prerequisites

1. GNU/Linux computing server with x86_64 architecture. The pipeline was tested on servers with following operational systems Ubuntu 20.04.06 LTS, Ubuntu 24.04.2 LTS, and Debian 12.5. Pipeline was never tested on Windows operating system and will likely not work, unless executed via WSL2 (windows subsystem linux).
2. User must have sudo privileges, and must be added to the docker group.
3. Docker (https://docs.docker.com/desktop/install/linux-install). The pipeline was tested on docker version 24.0.7 and 27.3.1.
4. Nexflow (https://www.nextflow.io/docs/latest/install.html) with the nextflow binary file located in a directory that is in the PATH variable. One can copy nextflow binary to e.g. `/usr/local/bin`

   ```
   sudo mv nextflow /usr/local/bin
   ```

   The pipeline was tested on nextflow version 24.10.3.5933, 24.04.4.5917, and 24.10.4.5934.
5. Nextflow config (https://www.nextflow.io/docs/latest/config.html). The config defines which nextflow executors are available, and adjust their default parameters to work best on a given machine. Below is the sample config modifying two in-build nexflow executors *slurm* and *local*. If put in `~/.nextflow/config` this config will be visible for nexflow regardless where *nextflow* command is executed.

   ```
   # definition of profiles accessible to nextflow
   profiles {
       local {
           # creates a profile called “local”.
           process.executor = 'local' # name of the executor used by this
   profile
           process.errorStrategy = 'retry'  # all modules in the pipeline
   executed using this profile are resubmitted if they fail
           process.maxRetries = 2 # number of times nexflow retries to execute
   a module
       }
      slurm {
           # create a profile called “slurm”. It uses nextflow’s “slurm”
   executor.
           process.executor = 'slurm'
   ```

```
        process.clusterOptions = '--nodelist=HOSTNAME1,HOSTNAME2' # sbatch
option list of node(s) available to slurm executor
        process.errorStrategy = 'retry'
        process.maxRetries = 2
    }
}
executor {
    # Here we modify default behavior of the executors (as named in a
previous section)
    $local {
        cpus = X # number of CPUs allocated to all processes e.g. 100
        memory = 'Y GB' # amount of RAM allocated to all processes e.g. ‘5
Gb’
    }
    $slurm {
       queueSize = Z  # amount of processes allowed in slurm queue
    }
}
# “Global” modifiers
docker.enabled = true # allow execution of processes inside docker
containers. MUST BE true
```

6. [optional] Slurm - a system workload manager (https://slurm.schedmd.com/). A program used by nextflow to distribute jobs between multiple nodes in a multiple server setup. Installation of this program is beyond the scope of this documentation. Slurm is not required to run our pipeline but it allows for independent execution of multiple nextflow jobs .

b. Installation

1. Clone the Repository with the source code

```
git clone https://github.com/mkadlof/plepiseq-phylogenetic-pipeline.git
cd plepiseq-phylogenetic-pipeline
```

2. Build the image (works for both viral and bacterial workflows)

```
docker build -t pzh_pipeline-phylo:latest -f Dockerfiles/Dockerfile.
```

3. Pull the Prokka helper image

```
docker pull staphb/prokka:latest
```

4. (optional) Copy & edit the template config (see 1.5 for detailed explanation)

```
cp nextflow.config.template nextflow.config
```

c. Running the viral pipeline

i) Create a working directory where you want to store the results and copy nf_pipeline_viral_phylo.sh from the repository's root directory (section 1.b)

ii) Prepare the input for the pipeline. The pipeline requires:

(1) The metadata file - a tab-separated text file with the following required columns:

(a) strain – Sample identifier (must match the filename, excluding extension). Sample identifier must be a part of a header in FASTA file
(b) virus - Virus name (sars-cov-2, influenza, or rsv)
(c) date – Collection date in YYYY-MM-DD format
(d) country – Country name (e.g., France)
(e) city – City name (e.g., Paris)
(f) type – Additional classification column representing suptype for Influenza (e.g H1N1)or type for for RSV (e.g. A). For SARS-CoV-2 can be identical with virus column

Examples of metadata file can be found in data/example_data/ directory.

(2) A directory containing full genomic sequences of analyzed samples in FASTA format. Each genome (sample) must be stored in a separate gzipped FASTA file (.fasta.gz). For multi-segment viruses (e.g. Influenza), all genomic segments of that sample must be included in a single FASTA file. Example directory layout of input directory:

```
/some/path/
├── Sample1.fasta.gz
├── Sample2.fasta.gz
├── Sample3.fasta.gz
```

Examples of input directories can be found in data/example_data/.

iii) Use the wrapper script with obligatory parameters. Following information must be provided

```
bash nf_pipeline_viral_phylo.sh --inputDir  PATH_TO_DIRECTORY_WITH_FASTAS \
                                --metadata PATH_TO_METADATA_FILE \
                                --organism  ALLOWED_SPECIES \
                                --results_prefix MY_AWESOME_PROJECT \
                                --projectDir PATH_TO_REPOSITORY\
```

d. Running the bacterial pipeline

i) Create a working directory where you want to store the results and copy nf_pipeline_viral_phylo.sh from the repository's root directory (section 1.b)

ii) Prepare the input for the pipeline. The pipeline requires:

(1) The metadata file -a plain tab-separated text file with the following **required** columns:

(a) strain – Sample identifier (must match the filename, excluding extension)
(b) date – Collection date in YYYY-MM-DD format
(c) region – Continent (e.g., Europe)
(d) country – Country name (e.g., France)
(e) division – Sub-national region (e.g., Ohio)
(f) city – City name (e.g., Paris)
(g) Serovar – Serovar designation (e.g., Enteritidis)
(h) MLST – MLST ID (e.g., 11)
(i) cgMLST – cgMLST ID (e.g., 110234)
(j) HC5 – HierCC cluster ID at threshold 5 (e.g., 13)
(k) HC10 – HierCC cluster ID at threshold 10 (e.g., 13)

Example of metadata file for bacterial pipeline can be found in the data/example_data/salmonella directory.

(2) A directory containing full genomic sequences of analyzed samples in FASTA format. Each isolate must be represented by a single gzipped FASTA file (.fasta.gz) containing all assembled contigs for that sample. Example directory layout:

```
/some/path/
├── Sample1.fasta.gz
├── Sample2.fasta.gz
├── Sample3.fasta.gz
```

Examples of input directories can be found in the data/example_data/salmonella directory.

(3) A path to external databases. Consult MST Tree section below

iii) Use the wrapper script with obligatory parameters. Assuming (1) your working directory contains `metadata_salmonella.txt` and (2) a `fastas/` directory, and (3) project's repository is in `/home/my_user/plepiseq-phylogenetic-pipeline`,

and (4) you've built/pulled the required Docker images as described in Quick Start section, (5) external databases are located in `/mnt/external_databases':

```
bash nf_pipeline_bacterial_phylo.sh --metadata metadata_salmonella.txt \
                                    --inputDir fastas/ \
                                    --inputType fasta \
                                    --genus Salmonella \
                                    --projectDir /home/my_user/plepiseq-
phylogenetic-pipeline \
                                    --results_prefix Salmonella_test \
                                    --db /mnt/external/databases
```

e. Running example scripts

We created a set of examples script that should work, without any alterations from the User, if guides from section 1a and 1b were carried out. The ONLY required change is in run_example_salmonella.sh script where the variable PATH_TO_EXTERNAL_DATABASES must be set. **Be advised** that example scripts for rsv and influenza use the slurm executor. If one choose a local executor simple change "-x slurm" to "-x local". To run example scripts simply execute one of the following command while inside the directory with the cloned repository.

```
./run_example_sars-cov-2.sh
./run_example_influenza.sh
./run_example_salmonella.sh
```

f. Validation

To validate the proper installation of the program you can run pytests. After successful execution of example scripts from 1e in the execution a **result** directory should appear with subdirectories like test_sars, test_influenxa, etc. To check if these directories contain a valid output simply type (e.g for Sars-CoV-2):

```
pytest tests/end-2-end/test_end-2-end.py --data-dir result/test_sars
```

Note, pytest requires following dependencies installedon your host machine: pytest, ete3, fastjsonschema, requests

2. Hardware requirements

The pipeline does not require GPU acceleration, however, some steps like proposing core genome with roary can be computationally heavy. Below are the minimal requirement to run the pipeline, however some modules (see chapter 7 and 8) might benefit from increasing number of available CPUs.

Pipeline	Suggested CPU	RAM	GPU
Bacterial	12	60 GB	0
Viral	12	30 GB	0

3. Repository layout

The structure of the main repository with its crucial components is outlined below.

```
├── Dockerfiles/
├── modules/
├── bin/
├── dag_png/
├── data/
├── tests/
├── config/
├── doc/
├── config/
├── VERSION
├── CHANGELOG.md
├── run_example_*.sh
├── nf_bacterial_phylogenetic_pipeline.nf
├── nf_viral_phylogenetic_pipeline.nf
├── nf_pipeline_viral_phylo.sh
└── nf_pipeline_bacterial_phylo.sh
```

1. Dockerfiles - a directory with Dockerfile

2. Modules – a directory with all the modules used by the viral pipeline. For bacterial pipeline modules are part of the corresponding .nf file

3. bin – a directory with a number of helper scripts used by the pieplines

4. dag_png – a directory with figures showing the pipelines topology shown as a Sirect Acyclic Graph

5. nf_bacterial_phylogenetic_pipeline.nf and nf_viral_phylogenetic_pipeline.nf – main scripts containing a nextflow-based pipeline

6. nf_pipeline_viral_phylo.sh and nf_pipeline_bacterial_phylo.sh - shell wrappers used to execute pipelines from the corresponding *.nf files. Wrappers include all the defaults used in PZH.

7. test – all the data used to perform unit tests of crucial scrpts from the bin directory

8. data - directory with all the data used to execute sample scripts and test pipeline installation, and configs used to visualize pipeline's result with microreact

9. config –/**deprecated**/ directory with JSON files understood by auspice and microreact. Used by auspice-related modules that are no longer in use. Microreact modules currently used config located in data/

10. VERSION – file containing current version of the pipeline

11. CHANGELOG.md – simplified information about history of changes for each version of the pipeline (from version 1.0.0)

4. Bacterial SNP pipeline – step-by-step

The workflow encoded in `nf_bacterial_phylogenetic_pipeline.nf` proceeds as follows:

1. **Gene prediction** — annotates each genome FASTA with Prokka. This step is skipped if user already provided gff files (that is part of the Plepiseq WGS pipeline)
2. **Pangenome & core alignment** – gff files are used to predict a pangenome for analyzed bactria using Roary program. Only genes present in more than 95% of samples are kept and used to propose full genome for each sample. Proposed genomes are already aligned.
3. **Sequence indexing** – part of the augur rpipeline - prepares an index for each sample with number of ambiguous and non-standard aminoacids in the genome
4. **Quality filtering** – part of the augur pipeline removes low-quality strains based on Ns/ambiguities thresholds.
5. **SNP alignment & partitions** -- a custom python script is used to extract only SNPs from the samples alignment and to calculate STAM correction to GTR+G model.
6. **Duplicate handling** – identification of samples with identical genome. The redundant samples are removed prior to phylogentic tree calculations (but are later reattach so the user can see all the samples he requested)
7. **RAxML-NG** – Maximum likelihood tree calculations with true bootstraps. User can select custom model supported by RAxML (GTR+G+ASAM is the default).
8. **Tree refinement** – midpoint rooting tree , collapsing weak branches (branches with support below 80 are collapsed), time-scaling with TreeTime using either user-provided evolutionary rates or values predicted by Timetree based on the provided phylogentic tree and metadata
9. **Geographic enrichment** – enrichment of user provided metadata with longitude and latitude values for better visualization.
10. **Visual exports** a Microreact project file is created it includes both user provided metadata as well ac classical phylogentic tree and timetree..

5. Viral pipeline – step-by-step

TBD

6. Output files

The JSON schema for phylogentic pipeline is located in https://github.com/michallaz/plepiseq_json repository in the main_output_schema_phylogenetics.json file. Below is a table with all the files returned by the pipeline

File	Example	Description
{results_prefix}.json	test_influenza.json	JSON file aggregating all the results of phylogentic pipeline. The name of the files depends on the value of results_prefix. Variable set during execution of the shell wrapper
{results_prefix}_{segment_id}_chronogram.nwk	test_influenza_chr1_PB2_chronogram.nwk	The file in Newick format with a chronogram calculated with TimeTree. Each segment of the viral DNA/RNA has a separate file. Name of the segments for an analyzed virus are extracted from the headers of input fasta files.
{results_prefix}_{segment_id}_filogram.nwk	test_influenza_chr1_PB2_filogram.nwk	The file in Newick format with a filogram calculated with RaXML or IQ-Tree. Each segment of the viral DNA/RNA has a separate file. Name of the segments for an analyzed virus are extracted from the headers of input fasta files.
{results_prefix}_{segment_id}_ microreactproject.microreact	test_influenza_chr1_PB2_microreactproject.microreact	The file with microreact project. Each segment of the viral DNA/RNA has a separate file. Name of the segments for an analyzed virus are extracted from the headers of input fasta files. The microreact project include following information: 1. Table with metadata for valid samples (part of the input for phylogenetic pipeline) 2. Chrnogram 3. Filogram

		4. Geographical data

7. Viral pipeline modules

TBD

8. Bacterial pipeline modules

Below is the detailed list of all modules used by the bacterial pipeline, its name, key program, role of the module, output and resources requested by this module

Module / process	Tool & key options	Function	Input	outputs	Resources (CPUs \ RAM \ time limit)
run_prokka	`prokka`	Annotate each assembly to produce GFF file	`FASTA file (can be gziped)`	`GFF file for a sample`	≤25 CPU / 10 GB / 20 m

run_roary	`roary -i 95 -cd 95 …`
	`-i - minimum percentage identity for blastp`
	`-cd percentage of isolates a gene must be in to be core`
		Create core-gene alignment across samples. Only genes present in >95% of isolates are considered as “core” genes”. The output is the FASTA with aligned genomes of provided samples and embl file with information regarding particular gene location with the alignment	`A directory with any number of GFF files`	`core_genes_alignment.fasta`
				`core_genes_alignment.embl`
					≤25 CPU / 30 GB / 1 h

augur_index_sequences	A simple module executing “augur index”	Index for downstream augur tasks. The csv file simply provides an infor which residues (standard, non-	`FASTA file with aligned genomes of analyzed samples`	`index.csv`	1 CPU / 30 GB / 1 h

		standard, Ns deletions) are present for each sample in the alignment			
augur_filter_sequences	Modules executes two independent programs “identify_low_quality_sequences.py” and “identify_low_quality_sequences.py”	identify_low_quality_sequences.py script is used to parse index csv and select based on provided threshold (for number of Ns and ambiguous residues) samples that do not pass QC criteria. This samples are next excluded from a FASTA file usind “augur filter” command. The FASTA file with aligned sequences of valid samples is called “valid_sequences.fasta”	`Index csv, FASTA with aligned genomes, embl files with gene location`	`valid_sequences.fasta`	1 CPU / 30 GB / 1 h
prepare_SNPs_alignment	`prep_SNPs_alignment_and_partition.py`	Module is a wrapper around prep_SNPs_alignment_and_partition.py. The script extract from the provided alignment only variable sites creating an alignment build solely of SNPs. To get a valid branches length it	`Fasta file with alignes valid samples, embl file with genes location in the alignment file`	`alignment_SNPs.fasta, partition.txt`	As specified by user (no upper limit) / 10 GB / 2 h

		calculates ASAM correction for GTR+G model. ASAM correction is simply a number of constant sites in the alignment split across 4 standard nuclotides			
identify_identical_sequences	`find_identical_sequences.py`	A wrapper around find_identical_sequences.py which identifies samples with identical sequence. Create a simple text file alignment_SNPs_ident_seq.csv where each row represent unique sequence. Row is a tab separated and in case multiple samples share the same sequence samples id are listed in tab separated format. Next from input fasta files is filtered so only samples with id at the firs position are kept	`Fasta file with the alignment of samples passing QC criteria Partition file`	`alignment_SNPs_unique.fasta`, mapping CSV	As specified by user (no upper limit) / 10 GB / 2 h
run_raxml	`raxml-ng with following parameters` --precision 15 (branches length ar rounded to this decimal numbers)	Wrapper around raxml-ng main program to predict phylogenetic tree based on the	`SNPs alignment of samples Partition file with model used`	`tree.raxml.support`	As specified by user (no upper limit) / 10 GB / 8 h

	-- all (perform ML search + bootstrapping) --site-repeats on (use site repeats optimization default is “on”) --model GRT+G+ASAM --tree 10 - number randomly generated ostarting trees --bs-trees 200 – number of bootstrap replicates --brlen scaled (branch length linkage between partitions scaled is the default)	provided fasta file with SNPs alignment. We dropped the idea of using per-gene partiotion file to speed up calculation several times (at the cost of reduced output quality)			
restore_identical_sequences	`root_collapse_and_add_identical_seq_to_tree.py`	Post processing of raxm-ng generated trees. This part includes several steps: 1. Tree rooting (midpoint method) 2. Collapse branches if their support is below 70 3. Reintroduce identical sequences (leaf representing identical sequence is turned into node from	`Nwk file with raxm-ng proposed tree` `A text file generated with` **identify_identical_sequences**	`${input_prefix}_classical_tree.nwk`	1 CPU / 10 GB / 10 m

		which all identical sequences arise, all of appended sequences have branch length 0)			
add_temporal_data	`augur refine. Most parameters are degfualts, however we add --keep-polytomies (to adjust for reintroduction of identical sequences that creates polytomies)`	A wrapper around augur refine that itself is a wrapper around the treetime program. This program take raxml-ng propose topology and alters branches lengths to reflect temporal signal for leafs (taken from metadata) and predicted occurrence of a node (predicted with treetime). The evolutionary rate for analyzed samples can be (I) user provided (II) either estimated from the data by treetime (III) hardcoded based on available data for a given genus. Priority is assigned to user provided value , than treetime predicted value, and if the data do not allow to predict reliable temporal signal,	`Nwk file with phylogenetic tree Metadata (for auspice) Alignment (for auspice)`	`tree3_timetree.nwk, traits.json, tree3_rescaled.nwk`	1 CPU (treetime s not parallelized) / 20 GB / 5 h

		to value obtained from publication			
metadata_for_microreact	same script as viral branch	Prepare Microreact metadata (country vs city granularity)		`${input_prefix}_metadata_microreact.tsv`	1 CPU / 20 GB / 1 h
prepare_microreact_json	prepare_json_for_microreact.py	Combine rescaled tree & metadata into `.microreact` bundle		`${input_prefix}_project.microreact`	1 CPU / 20 GB / 1 h

9. Bacterial pipeline wrapper

Below is the list of all oprions that can be specified with shell wrapper:

TBD

10. Viral pipeline wrapper

Below is the list of all oprions that can be specified with shell wrapper:

TBD