The Livnat Lab

How Evolution Happens

The MEMDS analysis pipeline

By

Assaf Malik, Evgeni Bolotin, Daniel Melamed, Yuval Nov and Adi Livnat

Department of Evolution - University of Haifa;

Corresponding author: Adi Livnat;
Email: adi.livnat@sci.haifa.ac.il

Table of contents

Introduction

The MEMDS analysis pipeline is an accompanying software package to the MEMDS protocol. The pipeline analyses deep sequencing data produced by MEMDS, and outputs summary tables of mutations found in the analyzed data relative to the reference gene(s).

Currently, the pipeline is CLI (command line interface) only, thus its scripts need to be executed from the terminal of the operational system on which it is ran.

Run requirements

Operating system

The pipeline can run on any major Linux distribution with a configured bash shell. On Windows it requires installation of Windows Subsystem for Linux, since some of the external programs the pipeline uses do not have Windows-compatible installers. For Mac, check the list of external dependencies below to see if they have native installers or ports for the system.

Distributed computing

The scripts are designed to run on distributed computing systems (clusters) managed by the SLURM Workload Manager. Running the pipeline on a different cluster system requires adjustment of job submission parameters within the wrapper scripts to its native commands.

To run the pipeline on a local machine, job submission scripts can be converted into a standalone version by substituting SLURM job submission command (**"sbatch"**) into a direct call to the job script (see examples below):

Example #1: calling to a python script with variables

In the submission script substitute "sbatch" submission code with a call to python script "python script.py <script parameters>"

*****

Cluster submission:

*sbatch --output="$f1.trimmed.out" --error="$f1.trimmed.err" -N1 -n1 --ntasks-per-node=1 *

-p hive1d,hive7d

*--wrap "**python trim7.py \"$f1\" \"$f1.trimmed\" \"$size_f1\" \"$size_r1\" \"$read_seq\" \"$read_pos\" \"$read_action\""***

Local run:

python trim7.py \"$f1\" \"$f1.trimmed\" \"$size_f1\" \"$size_r1\" \"$read_seq\" \"$read_pos\" \"$read_action\"

*****

Example #2: calling to a bash script using environmental variables

In the submission script substitute "sbatch" submission code with a call to a bash script "bash script.sh <script parameters>". If the script requires external parameters to run, they need to be exported to the environment prior to the work script invocation using "export" command.

*****

Cluster submission:

*sbatch -N1 -n20 --ntasks-per-node=20 *

*-p hive1d,hive7d *

*-o "$fout1".out -e "$fout1".err *

***--export=ref1="$ref1",f1="$f1",fout1="$fout1",threads=20,ref1_length="$ref1_length",params_1="$params_1" ***

bwa_job9.sh

Local run:

***export ref1="$ref1" f1="$f1" fout1="$fout1" threads=20 ***

ref1_length="$ref1_length" params_1="$params_1"

bash bwa_job9.sh

Note on the local run: Some of the jobs can be quite resource intensive, so it is better to avoid running the pipeline on weaker machines. Additionally, since local machines doesn't benefit from distributed computing capabilities, it is advisable to modify job submission scripts to launch only a single job at a time, by opening "for" loops

implemented in them.

External dependencies

The easiest way to install and manage all the external programs utilized by the pipeline is with Conda package manager (see below). Alternatively, the following programs and libraries need to be installed prior to running the pipeline:

- biopython
- bwa
- cutadapt
- fastqc
- pear
- perl
- picard
- pysam
- samtools
- seqtk
- trimmomatic

Note: The pipeline uses a direct call to invoke required programs. Therefore, in case of manual installation, all program installation folders need to be added to the $PATH for the pipeline to execute them correctly.

Installation of dependencies with Conda

1) Install Miniconda (**https://docs.conda.io/en/latest/miniconda.html**).

2) Install Bioconda channel (**https://bioconda.github.io**).

3) Use Conda to create an environment with required dependencies:

---

conda create -n modules3 python=2.7 bwa cutadapt fastqc pear perl picard pysam biopython samtools seqtk trimmomatic

---

Note: The "-n" option defines the name of the new environment created by Conda. The MEMDS analysis pipeline uses the name "modules3", but it can be changed to any name.

4) Check that all the dependencies were successfully installed.

a) **Run:** *conda env list*. This will display a list of existing Conda environments. Check that the newly created environment is on the list.

b) **Run:** *conda list -n modules3 > list_env.txt.* This will create a list of all linked packages in Conda environment specified by the "-n" option and output them to the **"list_env.txt"** file. Check the file to see that all programs listed in step 3 appear in the environment.

Note: If the environment was created with a custom name, use your environment name after the "-n" option and not the default name "modules3".

Preparation of the parameter files

The pipeline relies on user-defined data in the parameter files during its run. All the files are housed under the "scripts" directory. **Before starting a new run of the pipeline always remember to check that the parameters are correct and match your data!**

The pipeline utilizes the following parameter files during its run:

fastq_merging/samples_table_0.txt

This file defines parameters for merging partial ".fastq" files into a single data file. This is an optional step for same-sample data sequenced across several lanes. The file contains three columns:

1) **pair**: a serial number assigned by the user to the analyzed fastq files. For **single-end** data, **each read file** should have a unique pair number. For **paired-end** data **each pair of files** should have a unique pair number. Forward and reverse read files belonging to the same pair should share same pair number.

2) **sample:** A name of the sample to which the read files belong. All files sharing same sample name would be merged, resulting in a single ".fastq" file per sample for single-end data and in a pair of files for paired-end data (forward and reverse reads).

3) **file:** A full path to the location where the read files are stored, ending with the name of the file:

e.g.: /data/home/user/experiment/Raw_data/S1_L001_R1_001.fastq.gz.

For **paired-end** data the script is designed to parse a **first** path in the pair as

containing **forward-read** file and the second path - as containing **reverse-read** file. **Before concatenating the files always ensure that they are listed in a correct order across all analyzed pairs!**

scripts/wildcard_adapters_1.fa

This file contains sequence information of adapters and other contaminants that might be present in the analyzed data. Sequences specified in this file would be removed from the raw sequence data during the quality control step.

The file can be opened with any text editor program to check its contents and add additional sequences to it, as needed. New sequences should be added in FASTA format, as follows:

> Sequence_name

Nucleotide Sequence

config_files/samples_table.txt

This file stores information regarding the location of the analyzed data files. **It has the same structure as the "samples_table_0.txt" parameter file used for merging partial ".fastq" files (see above)**.

Note: If the data files underwent merging before the analysis, remember to provide here location of the merged files and not of the original!

config_files/params_1.sh

This file defines a number of parameters used by the pipeline to analyze the data:

1) **Conda settings:**

. /path_to_conda_install_dir/miniconda2/etc/profile.d/conda.sh

conda activate modules3

The first line invokes "conda.sh" script so Conda commands can be used from the shell. This line should contain **a full path** to the "conda.sh" script (found within Conda installation directory under "etc/profile.d/"). **Important:** The dot before the path is a part of the command, not a typo!

The second line activates Conda environment. If the pipeline-related environment was created with a custom name (see explanation on Conda above), the default

“modules3” name in the command should be replaced by the custom name.
In case that the dependencies were installed manually and installation folders were added to the $PATH, this part of the script can be removed. Alternatively, it can be used to export location of the manually installed programs to the environment, instead of the Conda commands. To export paths, use the following command:
export PATH=$PATH:/path/to/program/:/path/to/program2/

2) **params_adapters_1:** Specifies path to the file containing sequence information of adapters and other contaminants that might be present in the analyzed data. This file is used to identify unwanted sequences in the data during the quality control step. By default **params_adapters_1** points to the adapter file distributed with the pipeline - “wildcard_adapters_1.fa” (see explanation above).

3) **params_dir_out_1:** Specifies location of the output directory to store results produced by the pipeline.

4) **params_dir_reference:** Specifies location of the reference file directory. The pipeline aligns analyzed data against the reference files to identify mutations. See the “Reference file preparation” section below for further information on the reference files.

5) **Trimmomatic options:** Options supplied to the Trimmomatic program during the quality control step. This program is used to identify and remove potential contaminants and low quality bases from the analyzed reads. The following options can be defined:
a) **params_minimum_fastq_size_1:** Defines minimal length threshold for the reads to be included in the subsequent analyses. Sequence shorter than the threshold would be removed (default: 90 bp).
b) **qual_threshold:** Average quality score threshold for read bases. Bases having lower quality score than the threshold would be cut from the read by Trimmomatic (default: 30).
c) **qual_window:** Specifies number of consequent bases in the read over which average quality score is calculated (default: 3)

6) **is_SE:** Defines whether the analyzed data is paired-end or single-end. Specify '0' for **paired-end** data and '1' for **single-end** (default: 0)

7) **offset (read sorting):** In the read sorting step of the pipeline (see below) this parameter specifies the offset (in bp) relative to **sorting positions (**see explanation on **factors_table.txt: sort_pos** below) for pipeline to search for nucleotides identifying the read as belonging to gene of interest. The search is done within **sort_pos +- offset** (default: 3).

8) **BC3_min:** During mutation calling step this parameter specifies how many different secondary barcodes should be associated with a mutation for it to be included in further analyses (default: 1). It is inadvisable to use stringent parameter here, since additional steps aimed at filtering out potential artifact mutations are undertaken down the road.

9) **TSS:** This variable allows the pipeline to report mutation position relative to the Translation Start Site (TSS) of the analyzed gene, in addition to its position on the read itself. Here two numbers, separated by comma, are specified (**e.g.: -30,1):**

a) **Position of the first base in the read relative to the TSS**. If TSS occurs before the analyzed portion of the gene, this value should represent position of the first read base in the analyzed gene (e.g. 1100). If TSS falls **inside** the read, this value should be negative and calculated as **1 - TSS position in the read**. E.g., if TSS occurs at **position 31 of the read**, it relative position is: **1 - 31 = -30.**
b) **Read orientation relative to the coding strand.** 1 - same orientation; -1 - opposite orientation (complimentary strand was sequenced).

Thus, **-30,1** example above means that the read and the coding strand of the gene are in the same orientation and TSS falls in the 31st position of the read.

Note: If TSS data is not relevant, one can specify **1,1** here to report same position for mutation location in the read and its location relative to the TSS.

10) **filter_pos:** Defines position of the control insertion used to identify artifacts

occurring during the amplification stage. This parameter tells the pipeline to disregard mutations at this position (reads with the control insertion are filtered out beforehand), since they do not represent mutations of interest. If control insertion is not used, position 0 should be specified here.

11) **Consensus cut-offs:** These parameters set cutoff criteria used to identify consensus variants. Following parameters can be defined:
a) **min_freq -** Comma-separated list defining variant frequency cutoffs within read family for variant to be considered "true". Variants not passing the cutoff criteria would be considered artifacts.
E.g: **min_freq="0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0"**

b) **min_count -** Comma-separated list defining minimal size of read families to be used in the consensus variant analysis. Read families having less reads than the cutoff criteria would be removed from the analysis.
E.g.: **min_count="0,1,2,3,4,5,6,7,8,9,10,25,50"**

c) **min_bc3 -** Comma-separated list defining minimal number of unique secondary barcodes to be associated with a variant within read family for variant to be considered true. Variants not passing the cutoff criteria would be considered as artifacts, unless they pass criteria for number of secondary barcodes associated with more than one read (see below).
E.g.: **min_bc3="0,1,2,3,4,5"**

d) **bc3groupCountOK -** Variable defining minimal number of unique secondary barcodes to be associated with a variant within read family for variant to be considered true. If **min_bc3** cutoff defined above is smaller than this variable, the variant would be considered true only if it passes **bc3groupsWithReadsAbove1** criteria (see below).
By controlling this variable, the user sets a threshold above which sheer number of secondary barcodes associated with a variant is enough to confirm its authenticity. But below it, some of the secondary barcode read groups associated with a variant have to be of a specific size for variant to pass the secondary barcode group thresholds. The idea is that a variant associated with large enough number of reads is more likely to be

authentic than not, even if it is associated with smaller number of different secondary barcodes.
E.g.: **bc3groupCountOK=0**

a) **bc3groupsWithReadsAbove1 -** Variable defining minimal number of unique secondary barcode groups containing user-defined number of reads each to be associated with a variant within read family for variant to be considered true. A variant needs to pass either **bc3groupsWithReadsAbove1** or **bc3groupCountOK** criteria to be considered true. Minimal number of reads in the secondary barcode read groups analyzed here is defined by the **BC3_min** parameter (see point 8 above).
E.g.: **bc3groupsWithReadsAbove1=0**

config_files/factors_table.txt

This file defines parameters of barcode trimming, sorting by origin and variant search for analyzed sequences. It is a tab-delimited file with 17 columns:

1) **sample**: Name of the analyzed sample to which parameters appearing in the next columns would be applied. Sample names listed here should be consistent with the names listed in the 'sample' field of the 'samples_table.txt' parameter file.

2) **reference_size:** Length of the reference sequences to which the analyzed reads are aligned (in bp). **The pipeline can work with multiple references per sample, but only if they have same length, hence this field accepts only a single length value for all reference sequences used.**

3) **size_f:** Size of the primary barcode (5'), in bp, attached to the analyzed reads by the MEMDS procedure. The barcode includes unique sequence to distinguish between group of reads in the sample + identifier bases to distinguish between reads from different samples.

4) **size_r:** Size of the secondary barcode (3'), in bp, attached to the analyzed reads by the MEMDS procedure. The barcode includes unique sequence to distinguish between group of reads in the sample + identifier bases to distinguish between reads from different samples.

5) **limit_starts:** Start position of the window in which the pipeline would search for mutations in the analyzed reads. If left empty, the pipeline would search from the first position of the reference. To specify multiple search windows, use comma-separated list of values (e.g: 15,31). Positions of "planted" mutations that indicate problems with barcode attachment (see explanation below for columns 7 - 9) also should be listed here.

6) **limit_ends:** End position of the window in which the pipeline would search for mutations in the analyzed reads. If left empty, the pipeline would search until the end of the reference. To specify multiple search windows, use comma-separated list of values (e.g: 15,84). The order of the end position values in the list should match the order of the start positions in the previous column. Positions of "planted" mutations that indicate problems with barcode attachment also should be listed here.

7) **seq_pos:** During the MEMDS procedure each analyzed DNA sequence is barcoded with a set of unique barcodes at its 5' and 3' ends. To account for the rare events when oligonucleotides used to attach the primary barcode (5') undergo extension themselves, using the barcoded DNA as a template, a single base insertion is planted in the oligo sequence. 'Seq_pos' field specifies a position of this insertion, which allows the pipeline to identify these sequences and remove them from further analyses.

8) **seq_mut:** This field specifies what allele should be found at the position specified by 'seq_pos' field to mark the sequence as an extended oligo and remove it from the analysis. The allele is listed as <reference_nucleotide><query_nucleotide>, with INDELs marked by hyphen ("-") (e.g.: -G).

9) **seq_action:** This field specifies what action to take on sequences containing the allele defined by the previous fields. **Currently, the pipeline is designed to remove such alleles and accepts only 'exclude' keyword in this field.**

10) **read_seq:** The primary and the secondary barcodes attached to the analyzed DNA contain a sequence of four nucleotides that serve as a sample identifier. The

'read_seq' field lists these identifier sequences, to allow the pipeline distinguish between sample sequences and contaminants from other libraries. Comma is used to separate between primary and secondary barcode identifiers. Vertical bar ("|") is used to separate variants of the sequence in the same barcode. For the primary barcode sequence it is advisable to include also first three - four nucleotides of the analyzed gene following the identifier to ensure that the barcode was attached to the right DNA sequence (e.g. - ACGTTGT|ACGTAGT,CGTG). **Note:** The pipeline assumes that the order of listed identifiers is always <5' barcode ID>,<3' barcode ID>

11) **read_pos:** This field specifies start position of the identifier sequences listed in the 'read_seq' field, so the pipeline knows where in the read it should look for the identifiers. As in the previous field, comma separates primary and secondary barcode positions and vertical bar separates start positions of the variants in the same barcode. The secondary barcode identifier start position is determined **from the end** of the read, therefore it is expressed as a negative value (E.g. - 15|15,-6).

12) **read_action:** This field specifies what action to take if identifier sequences were found at right positions within the analyzed reads. **The pipeline accepts 'include' keyword to indicate that the reads should be included in further analyses and any other string - to remove them**. The keywords should be listed as a comma separated list, with separate values for primary and secondary barcode identifiers (e.g.: include,include).

13) **sort_pos:** In case that the analyzed DNA originates from multiple genes, this column specifies positions in the read that can be used to sort reads by their origin gene. The positions are listed as a comma separated list, with the semicolon separating identifying positions of different origin gene (e.g. - 63,64,65,66,67,68;63,64,65,66,67,68). Identifying positions of different haplotypes belonging to the same gene should be separated by ampersand ("&").

14) **sort_nucl:** This column specifies which nucleotides should be found at the positions listed in the 'sort_pos' field to consider analyzed read as originating from a specific gene. As in the 'sort_pos' field, identifying nucleotides should be comma separated with semicolon separating data for different genes and ampersand

separating haplotypes (e.g. - C,G,T,T,A,C;T,G,T,C,A,A). The order in which identifying nucleotides are listed should match the order of identifying positions listed in the previous field.

15) **sort_refs:** This column specifies names of genes from which the reads originate. These genes serve as a reference against which the reads of matching origin are aligned for mutation search. Names of different genes should be separated by semicolon (e.g. - HBB;HBD). The order in which gene names appear should match the order in which identifying nucleotide lists appear in the previous column.

16) **sort_ref:** This column specifies the name of a default reference against which all reads whose origin couldn't be determined are aligned. This column should contain only a single gene name (e.g. - HBB). It can be one of the genes listed in the previous field or another, unrelated gene.

17) **sort_match:** This column specifies what fraction of nucleotides found at the positions specified by the 'sort_pos' field should match nucleotides listed in the 'sort_nucl' field for the read to be considered as originating from a specific gene.

Preparation of the reference sequence files

To identify mutations in the analyzed sample, the pipeline compares reads against a set of reference genes defined by the user. To prepare reference sequence files for use by the pipeline:

1) Place **all reference sequence files** in a **directory** specified by the 'params_dir_reference' parameter in the 'params_1.sh' file. The pipeline is not designed to search for reference files at any other destination.

2) Match the names of the reference files to the names appearing in the "sort_refs" and "sort_ref" fields of the "factors_table.txt".

3) Make sure that all reference files have **".fa" extension**. The pipeline doesn't recognize reference files with a different extension. Remember to check that OS is not configured to hide file extensions by default and ".fa" is the actual extension of the

file!

4) Check that each reference file contains only **a single reference sequence** in FASTA format. The first line should include sequence name, starting with the ">" symbol (E.g.: >PPIA). The second line should include the sequence itself.

Example: If the "sort_refs" field contains values "HBB;HBD" and "sort_ref" field contains "HBB" value - the reference sequence directory should contain **two files**: "HBB.fa" and "HBD.fa". Each file should contain **a single reference sequence** of the appropriate gene (or part of it).

Quick start guide

This section provides basic information needed to run the pipeline. For a more detailed information regarding each analysis step and its associated output refer to the next section.

Notes before the run

1) The "**$i"** symbol after command name indicates that multiple substeps are available for a given pipeline step, with available run options listed in parentheses.
For example: *bash my_script.sh $i (1-3)* means that pipeline step run by **my_script.sh** is composed of 3 substeps and the user needs to choose which one they want to activate. To complete the pipeline step, all relevant substeps need to be run in a sequential order, save for substeps marked as "optional".

2) When running on a cluster, for each job submitted by the script the following message would appear: "Submitted batch job #job_serial_number". **Before moving to the next option in the script or to the next step in the pipeline**, a**lways check that all running jobs were completed successfully.** To check job status, use the following commands (for SLURM systems):

a) *squeue -u "username"| grep -c "username"* – this command displays number of jobs in queue for account defined by the "username". Completed or canceled jobs would be removed from the queue.

b) *sacct -u "username"* – this command lists all the jobs that ran on the account defined by the "username" at current session. In the last column it indicates for each job if it is completed, canceled or still running. Ensure that all relevant jobs have 'Completed' status before submitting new ones!

3) **Job submission commands vary between different cluster systems**. Before the run remember to update the **"sbatch"** command in the pipeline job submission bash (".sh") files with the syntax relevant to your cluster.

4) **Remember to check the ".err" and the ".out" log files** produced during job run on the cluster. Log files are generated in the same folder as the result files produced by the pipeline in each step or substep. They record warnings and error messages raised by the script or by the cluster during job run.

5) Before running the pipeline, remember to put the folder containing pipeline scripts and associated parameter files into the relevant sample directory. If multiple samples are analyzed, "scripts" folder should be created in each sample's directory.

Pipeline wrapper script

The wrapper script provides an interactive menu allowing the users to navigate through the pipeline and choose which step or substep to execute.

To run the wrapper script, navigate from the terminal to the directory containing it and use the command: *bash MEMDS_pipeline_wrapper.sh.* By default, the wrapper script is provided with the rest of the pipeline scripts, but it can be placed in any directory. If pipeline scripts are placed on a remote machine, helper script should be placed on the same machine, and not in a local directory.

At the beginning of the run, helper script prompts to provide paths to the folders containing pipeline scripts, e.g: ***My_computer/analysis/sample/scripts***.
Multiple paths, pointing to script folders of different samples can be provided, separated by semi-colon, e.g:
My_computer/analysis/sample1/scripts;My_computer/analysis/sample2/scripts.

After the paths are entered, the script offers on-screen menu with possible run options. If a step contains several substeps, additional menu would appear asking to choose relevant substep. If multiple paths are provided, step and substep choice prompts would appear for each path provided.

Running pipeline scripts directly

1) Navigate into the scripts folder from command line (***cd /path/to/sample/scripts***) to use the pipeline. All pipeline commands should be run from inside the scripts folder!

2) **(Optional) Merging raw data:**

a) *bash concatenate_partfiles.sh*

b) **Local run:** *bash fastq_merging/samples_table.concat.sh* **or**

c) **Cluster run:** *srun bash fastq_merging/samples_table.concat.sh*

3) **Formatting parameter data for use by the pipeline:**

a) **Single-end data:** *bash setting_1-SE.sh* **or**

b) **Paired-end data:** *bash setting_1-PE.sh*

4) **Quality control and clearing of raw data + paired-end data merging**:

a) **Single-end data:** *bash filter-SE4.sh $i* (1-3) **or**

b) **Paired-end data:** *bash filter-PE4.sh $i* (1-4)

5) **Selecting reads with correct barcodes and separating between barcodes and genomic data**:

a) *bash trim7.sh 1*

6) **Sorting reads by their origin gene:**

a) *bash sort2.sh 1*

7) **Mapping reads to the reference sequences:**

a) *bash bwa9.sh $i* (1-2)

8) **Making alignment files viewable in IGV:**

a) *bash create_dummy_genome5.sh 1*

9) **Creating mutation table that lists sequencing quality alongside each mutation:**

a) *bash sam_to_mutation-list-3.sh 1*

10) **Creating a table of mutations found in the analyzed data:**

a) *bash sam_to_mutation-table_5.2.sh 1*

11) **Detecting consensus mutations that pass a set of user defined thresholds:**

a) *bash consensus_15.1.sh $i* (1-2)

Detailed pipeline guide

The section below provides detailed information regarding the various steps of the MEMDS analysis pipeline. The scripts making up the pipeline are meant to be run in the sequential order listed below, unless noted otherwise. Optional steps are applicable only to certain types of data and can be skipped, if not needed.

Before the run

1) Create sample folder to store analysis results. Under the sample folder:

a) create a **"scripts"** directory to contain pipeline scripts and associated parameter files;

b) create a **"seqs"** directory containing reference sequences for alignment of studied reads (see **"Preparation of the parameter files"** section above for more details on the reference files).

2) Refer to the **"Preparation of the parameter files"** section above to organize pipeline parameter files needed for the analysis inside the 'scripts' folder.

3) Use the helper script, **"MEMDS_pipeline_wrapper.sh",** situated in the "scripts" directory, and follow on-screen instructions to run the pipeline. Alternatively, enter the 'scripts' directory, to run pipeline scripts directly. Pipeline scripts need to be invoked only from inside the "scripts" directory to run properly.

Running the pipeline

1) (Optional) Merging raw data

Step description:

This step allows concatenation of reads from multiple ".fastq" files into a single file. Commonly, it is used to collect same-sample data sequenced on multiple lanes. **Important:** When merging ".fastq" files, make sure that all merged files belong to same sample and same treatment, before merging.

Run instructions:

a) **Run script:** *bash concatenate_partfiles.sh*

b) This command will create two files in the "scripts/fastq_merging" folder:

1) samples_table.concat.sh
2) samples_table_err.log

These files would provide information needed to merge the input data.

c) **Run script:**

Cluster: *srun bash fastq_merging/samples_table.concat.sh*

Local: *bash fastq_merging/samples_table.concat.sh*

d) Inspect the resulting merged ".fastq" files.

Output description:

The "samples_table.concat.sh" file is a script file for merging partial ".fastq" files into a single file. The associated "samples_table_err.log" file captures errors encountered during input data presence check. **Remember to check the error log file** before running file merging script to make sure that all input data is correct and present.

If **"samples_table.concat.sh"** script points to the wrong input files - check that parameter file **"samples_table_0.txt"** file is properly formatted and contains correct paths to the partial files, as described in the **"Preparation of the parameter files"** section above. Then re-run the "*concatenate_partfiles.sh"* script, using the updated parameter file.

Important: All changes and updates pertaining to the data should be done through the parameter file and not through the **"samples_table.concat.sh"** script itself. While it is possible to update the merging script directly, having incorrect parameter files

might cause problems if the pipeline is re-run in the future on this data.

2) Formatting parameter data

Step description:

This step organizes data from "samples_table.txt" and "factors_table.txt" files found in the "scripts/config_files" folder into a single bash file named "samples_table.sh". Pipeline scripts read parameters needed for their run from this file.

Run instructions:

a) Check that all parameter files are found in the "scripts/config_files" folder and include correct parameter data.

b) **Run script matching input data type:**

1) **Single-end data:** *bash setting_1-SE.sh* **or**

2) **Paired-end data:** *bash setting_1-PE.sh*

c) Inspect the output "samples_table.sh" file for presence of error messages or incorrect parameter values.

Output description:

"Samples_table.sh" bash file unites information from "samples_table.txt" and "factors_table.txt" parameter files into a single file. It is located in the "scripts/config_files" folder, same as the text parameter files. The data in the file is organized by sample\treatment name, all parameters following the "title" field which lists the name of the sample relate to this sample. Each parameter is listed in a separate line. Names of the parameters match column names in the "factors_table.txt".

If the **"samples_table.sh"** file contains error messages or wrong parameters - check that "samples_table.txt" and "factors_table.txt" files are properly formatted and contain correct data. For detailed explanation on preparation of these files see **"Preparation of the parameter files"** section above.

Important: All changes and updates pertaining to the data should be done through the parameter file and not through the **"samples_table.sh"** script itself. While it is possible to update the merging script directly, having incorrect parameter files might cause problems if the pipeline is re-run in the future on this data.

3) Quality control and cleaning of the raw data

Step description:

This step quality-filters input data from various contaminants and low quality bases and creates statistics regarding data before and after cleaning steps. The step contains several substeps that should be run in a consecutive order.

The users can also perform their own cleaning of the raw data and provide the resulting files as input for the next pipeline steps. To make manually created files compatible with the pipeline, make sure that:

1) The files are **uncompressed** and have **".fastq"** format.

2) Paired-end data is merged. For each analyzed sample, the pipeline expects to find a single read-containing file per treatment (control and experiment) at the end of the quality control step.

3) Manually created files should be placed in the pipeline output folder (as defined by the user in the "params_1.sh" file) under **"filtered"** directory. Their names should be as follows: **<sample_name>_idx$i.assembled.filtered.fastq** (paired-end data) or **<sample_name>_idx$i.filtered.fastq** (single-end data). Sample name and its accompanying index are defined by the "title" rows in the "samples_table.sh", for example:

* title[0]="contLL087" -> contLL087_idx0.assembled.filtered.fastq;
* title[1]="expLL087" -> expLL087_idx1.assembled.filtered.fastq;

Note: This step uses several external programs parameters to merge, trim and clean raw read data: Pear, Cutadapt and Trimmomatic. Some of the programs' parameters can be adjusted via **"params_1.sh"** file (see **"Preparation of the parameter files"** section above). Additional parameters can be adjusted by editing the **"fastq-filter_job_3.sh"** (Cutadapt and Trimmomatic) and the **"filter-PE4.sh"** (Pear) scripts.

Step 3-1: Data quality check

Run instructions:

a) **Run script matching input data type:**

1) **Single-end data:** *bash filter-SE4.sh 1* **or**

2) **Paired-end data:** *bash filter-PE4.sh 1*

b) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

c) Inspect the output files in the output folder under "fastqc" directory. Inspect run log files under **"logs"** directory in the"fastqc" folder.

Output description:

This substep uses **FastQC** program to assess sequencing quality of the input files. The output files are stored under **"fastqc"** directory in the output folder (as defined in the "params_1.sh" file).

In case of the paired-end data, forward and reverse read files are processed separately and output files are created for each one of them. Here "f" in the file name stands for forward read file output and "r" - for reverse read file output.

This substep creates the following files:

1) **<sample_name>_idx$i.err** and **<sample_name>_idx$i.out** - log files produced during the run of the job on the cluster. Check that the ".err" file doesn't contain any error messages and that the ".out" file contains the following line: "Analysis complete for *sample_name.fastq*". Log files are stored under **"logs"** directory in the results folder.

2) **<sample_name>_fastqc.html** - a graphical summary of the FastQC program results. Can be opened by any web-browser. For more information on the FastQC output refer to FastQC manual.

3) **<sample_name>_fastqc.zip** - Contains raw summary files produced by FastQC from which the program creates graphical output summary. Can be used for a more detailed assessment of the input data.

Step 3-2: Paired-end data merging

Run instructions:

a) **Run script:** *bash filter-PE4.sh 2*

b) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

c) Inspect the output files in the output folder under "filtered" directory. Inspect run log files under **"logs"** directory in the "filtered" folder.

Output description:

This substep uses **Pear** program to merge forward and reverse reads of paired-end data. The output files are stored under "filtered" directory in the output folder (as defined in the "params_1.sh" file).

Here**, Pear** is used with default run parameters for read merging. Users wishing to pass additional arguments to Pear program, can do so by modifying Pear call line in the "**filter-PE4.sh**" script (line 82 in the default script). For more information on Pear run options refer to Pear manual.

This substep creates the following files:

1) **<sample_name>.assembled.err** - error log file produced during the run of the job on the cluster. Check that the ".err" file doesn't contain error messages to ensure that the job was completed successfully.

2) **<sample_name>.assembled.out, <sample_name>.assembled.info** - Both files are identical and contain job summary log produced by Pear at the end of the run. Check either log file to ensure that Pear processed all the reads (log file contains the line "Assemblying reads: 100%"). Check assembly statistics listed in three last rows of the file to ensure that most reads were merged. Reads that remain unmerged would be removed from further analyses.

3) **<sample_name>.discarded.fastq, <sample_name>.unassembled.forward.fastq, <sample_name>.unassembled.reverse.fastq** - These files contain reads that Pear failed to merge or reads that were discarded because they didn't pass quality thresholds. Reads contained in these files are not processed by the pipeline.

4) **<sample_name>.assembled.fastq -** This file contains merged reads that are used by the pipeline in the subsequent analyses. A separate file is created for each analyzed sample and treatment.

Step 3-3: Read trimming

Run instructions:

a) **Run script matching input data type:**

1) **Single-end data:** *bash filter-SE4.sh 2* **or**

2) **Paired-end data:** *bash filter-PE4.sh 3*

b) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

c) Inspect the output files in the output folder under "filtered" directory. Inspect run log files under **"logs"** directory in the "filtered" folder.

Output description:

This substep uses **Cutadapt** and **Trimmomatic** programs to trim analyzed reads. The aim is to remove low quality bases and contaminants, such as adapter sequences, from the analyzed reads. Trimmed reads are considered of high quality and ready for further analyses. The output files are stored under **"filtered"** directory in the output folder (as defined in "params_1.sh" file). In case of the paired-end data, this step processes merged read data generated by **Pear** in the previous step.

This substep creates the following files:

1) **<sample_name>.assemb.filtered.err** - a log file produced during the run of the job on the cluster. Contains summary of Trimmomatic run. Check that the log doesn't contain error messages and indicates that Trimmomatic run was completed successfully. Check statistics of removed sequences - if a large portion of sequences doesn't pass Trimmomatic filters it might indicate issues with the quality of the input data.

2) **<sample_name>.assemb.filtered.out** - a log file produced during the run of the job on the cluster. Contains summary of Cutadapt run. Refer here for detailed explanation on the Cutadapt run summary log: https://cutadapt.readthedocs.io/en/stable/guide.html#how-to-read-the-report.

3) **<sample_name>.assembled.filtered.fastq** - Contains all reads surviving the trimming and the cleaning. These reads are considered of high-quality and constitute input for the next pipeline steps.

Step 3-4: Trimmed data quality check

Run instructions:

a) **Run script matching input data type:**

1) **Single-end data:** *bash filter-SE4.sh 3* **or**

2) **Paired-end data:** *bash filter-PE4.sh 4*

b) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

c) Inspect the output files in the output folder under "fastqc" directory. Inspect run log files under **"logs"** directory in the"fastqc" folder.

Output description:

This substep uses **FastQC** to analyze quality of the merged and trimmed ".fastq" files. The aim is to check that merged reads have expected length, the trimming improved the quality of the reads, relative to the initial data, and that contaminants were removed from the input. The output files are stored under **"fastqc"** directory in the output folder (as defined in the "params_1.sh" file).

This substep creates the following files:

1) **<sample_name>.assembled.fastqc.out** - log file produced during the run of the job on the cluster. Check that the file doesn't contain error messages and contains the following line: "Analysis complete for *sample_name.assembled.filtered.fastq*".

2) **<sample_name>.assembled.filtered_fastqc.html** - a graphical summary of the FastQC program results. For more information on the FastQC output refer to the https://www.bioinformatics.babraham.ac.uk/projects/fastqc/.

3) **<sample_name>.assembled.filtered_fastqc.zip** - Contains raw summary files produced by FastQC from which the program creates graphical output summary.

Step 3-5: (Optional) Sub-sampling ".fastq" files

Run instructions:

a) **Run script matching input data type:**

1) **Single-end data:** *bash filter-SE4.sh 4* **or**

2) **Paired-end data:** *bash filter-PE4.sh 5*

b) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

c) Inspect the output files in the output folder under "tests" directory. Inspect run log files under **"logs"** directory in the "tests" folder.

Output description:

This substep creates a sub-sample of the analyzed ".fastq" files with 10000 randomly chosen reads. It sub-samples both the raw input files and the cleaned ".fastq" files after the quality control steps (merging, trimming). Due to their small size these sub-samples can be quickly analyzed by the pipeline to provide some "feeling" of the data before the bulk of it is analyzed. Also, due to their small size, these files and any output based on them can be inspected manually to ensure that the calculations done by the pipeline doesn't contain any unexpected bugs. The sub-sampled ".fastq" files are stored under **"tests"** directory in the output folder (as defined in "params_1.sh" file).

Note: The test files need to be renamed and placed under "filtered" directory, as described at the beginning of this step, to be used with the pipeline. When doing so, be careful not to overwrite the actual sample files.

4) Barcode trimming

Step description:

This step filters reads based on the presence of the correct sample-identifier sequences attached to the read together with the primary and secondary barcodes. Additionally, it separates between the analyzed genetic sequence and the attached barcode sequences and moves barcode sequences into the header row of the relevant read.

To ensure that the barcodes were properly sequenced, the script checks per-base sequencing quality of the barcode sequences. Positions not passing the quality threshold would be marked by 'N'. Quality score is calculated using the "phred+33" score scheme. To define a different scoring scheme or adjust quality threshold, modify the "barcodeQuality_phred33" function in the "trim7.py" script.

Run instructions:

a) Check that the **"factors_table.txt"** contains correct values in the columns: "size_f", "size_r", "read_seq", "read_pos", "read_action". In case of incorrect values - adjust the parameter file and **repeat step 2** described above to produce correct bash parameter file, before starting this step.

b) **Run script:** *bash trim7.sh 1*

c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

d) Inspect the output files in the output folder under "filtered" directory.

Output description:

This step sorts reads into two ".fastq" files - one containing reads with correct identifier sequences and one with wrong identifiers in either 5' or 3' barcode or both. Upon step completion, the following files are added to the "filtered" directory:

1) **<sample_name>.assembled.filtered.fastq.trimmed.err,**

<sample_name>.assembled.filtered.fastq.trimmed.out - Log files produced during the run of the job on the cluster. Check that the ".err" file doesn't contain error messages and that the ".out" file contains correct data from "read_seq", "read_pos" and "read_action" columns of the "factors_table" file.

2) **<sample_name>.assembled.filtered.fastq.trimmed.fastq,**

<sample_name>.assembled.filtered.fastq.trimmed.barcodes - A ".fastq" file containing trimmed reads and its associated list of 5' and 3' barcode sequences found in these reads. Contain reads and barcode sequences (5' and 3') with correct identifiers.

Barcode list serves for diagnostic purposes. It can be used to check that the script identifies as barcode and trims correct part of the read. Additionally, it can be used to analyze variation of barcode sequences and check that no barcodes appear more frequently than can be expected from the MEMDS protocol.

3) **<sample_name>.assembled.filtered.fastq.trimmed.wrongId.fastq,**

<sample_name>.assembled.filtered.fastq.trimmed.wrongId.barcodes - A ".fastq" file containing trimmed reads and its associated list of 5' and 3' barcode sequences

found in these reads. Contain reads and barcode sequences (5' and 3') with identifiers that do not match user-defined pattern and, likely, represent contaminants.
A list of barcodes with non-matching identifier sequences can be compared against identifiers of other samples to check the possibility of sample cross-contamination. It can also be used to assess relabeling frequency, by counting relabeling oligo signature appearing in the list of wrong IDs.

4) **<sample_name>.assembled.filtered.fastq.trimmed.log** - A summary file listing how many analyzed sequences had wrong identifier in their barcodes. If high number of reads with wrong identifiers is found - check that the identifier parameters listed in the "factors_table.txt" file are correct. If parameters are correct - this might indicate an issue in the experimental procedures or large cross-contamination from other samples.

5) Trimmed read sorting

Step description:

This step sorts reads by their origin, using conserved positions in each gene of origin as an identifier of read identity. Reads whose origin can't be determined are put in a separate list.

Presence of gene-identifying bases is checked within the user-defined offset boundaries from their expected location in the read to account for possible indels (default windows - [-3, +3]). Defined offset can be changed in the **"params_1.sh"** parameter file. Parameter changes should be done before running the step.

Run instructions:

a) Check that the **"factors_table.txt"** contains correct values in the columns: "sort_pos", "sort_nucl", "sort_refs", "sort_ref", "sort_match". In case of incorrect values - adjust the parameter file and **repeat step 2** described above to produce correct bash parameter file, before starting this step.
b) **Run script:** *bash sort2.sh 1*
c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.
d) Inspect the output files in the output folder under "sorted" directory.

Output description:

The script organizes reads coming from potentially different genes into separate ".fastq" files. Upon step completion, the following files would be added to the "sorted" directory:

1) **<sample_name>.err, <sample_name>.out** - Log files produced during the run of the job on the cluster. For each sample, check that the ".err" file doesn't contain error messages and that the ".out" file contains the line "closing <sample_name>.<ref_name>" to indicate that all reads were analyzed and sorted.

2) **<sample_name>.<ref_name>.fastq** - Fastq files containing all reads of given sample/treatment matching sorting signature of a specific reference gene above the "sort_match" threshold. A separate file is created for each reference gene signature against which the reads are compared.

3) **<sample_name>.<default_ref_name>.others.fastq** - Fastq file containing reads that do not match any of the listed sorting signatures above the "sort_match" threshold. In the next step these reads will be mapped against a reference gene listed as "default" in the "factors_table.txt" parameter file. **Note:** In a good sample the majority of reads should be successfully sorted and the "others" file should contain only a small amount of data.

4) **<sample_name>.<ref_name>.withIndels.log,**
<sample_name>.<default_ref_name>.others.withIndels.log - **Depricated.** Log files listing reads whereby origin-identifying positions were found with offset. These files can be used to analyze indel-containing reads separately from the rest. By default they are not analyzed by the pipeline. To produce these logs files uncomment lines 162,168-169 in the "sort2.py" script.

6) Mapping reads to reference sequences

Step description:

This step aligns trimmed and sorted reads against their reference sequences. Reads having undetermined origin are aligned against the default reference chosen by the

user. Reads containing barcodes with low quality bases, as identified in step 4, are skipped.

Step 6-1: Reference sequence indexing

Run instructions:

a) Check that all reference sequence files are properly formatted and present in the reference file directory, as described in the **"Preparation of the reference files"** section.

b) **Run script:** *bash bwa9.sh 1*

c) Inspect the output index files in the reference file folder.

Output description:

For each reference file index files with the following extensions should be created: ".amb", ".ann", "bwt", ".dict", ".fai", ".pac", ".sa". Additionally, an empty file named "<ref_name>.fa.indices.OK" should be created to indicate that the job was completed successfully.

Step 6-2: Read alignment against reference

Run instructions:

a) Check that the "factors_table.txt" contains correct values in the columns: "reference_size", "sort_refs", "sort_ref". In case of incorrect values - adjust the table and **repeat step 2** described above to produce correct bash parameter file, before starting this step. Check that all reference files are present in the reference file directory.

b) **Run script:** *bash bwa9.sh 2*

c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

d) Inspect the output files in the output folder under "mapping" directory.

Output description:

This substep uses BWA program for read alignment. It outputs alignment files in SAM and BAM formats. The followng files are created:

1) **<sample_name>.<ref_name>.bwa.err, <sample_name>.<ref_name>.bwa.out** -

Log files produced during the run of the job on the cluster. Check that the ".err" file contains only summary of BWA program run and doesn't contain any error messages.

2) **<sample_name>.<ref_name>.bwa.0.sam** - Read alignment file produced by the BWA program in SAM format.

3) **<sample_name>.<ref_name>.bwa.sam** - A modified alignment file, SAM format. Here reference name in the RNAME field is substituted by the 5' barcode sequence of given read and tag containing 3' barcode sequence is added (XB:Z:bc3_seq) to each alignment line. If the read is missing primary barcode data, **"BC5_missing"** tag would appear in the RNAME field. If the read is missing secondary barcode date, **"BC3_missing"** tag would appear in the "XB:Z" field.

4) **<sample_name>.<ref_name>.bwa.sam.header** - A list of custom header lines created for the modified SAM file. Here the SN field contains 5' barcode sequences of the reads, instead of the reference name, so the header lines fit to the new RNAME column names. It is used to create a sorted BAM file from the modified alignment file.

5) **<sample_name>.<ref_name>.bwa.sorted.bam, <sample_name>.<ref_name>.bwa.sorted.bam.bai -** A sorted BAM file and its associated index created from the modified SAM file. BAM file is sorted by the alignment position of the query to the reference. This file is utilized by the pipeline for mutation calling in the next steps.

Note: In case of particularly large indels or other aberrations that prevent the read from mapping to the reference properly, BWA can produce split alignments - multiple partial alignments of different read parts, rather than single alignment of the read to the reference. **Currently, the pipeline is not designed to handle these cases.** The pipeline looks only at the primary alignment, starting at the beginning of the read, while discarding any secondary ones. As a result, the pipeline usually reports extremely long deletions, tens of base-pairs long, in place of secondary alignments. If such cases are encountered in the consensus mutation data produced by the pipeline, they should be carefully reviewed to distinguish between true large-scale deletions

and split alignment artifacts.

To map reads to the reference files, while filtering out potential split alignments, run "bwa_9.1.sh" script. To keep all alignments, and deal with potential split alignemnts at a later stage, run **"bwa_9.sh"** script, as described above.

7) Making alignments viewable in IGV

Step description:

This step creates files in the "mapping" folder needed to make alignment files ready for visual inspection in IGV (Integrative Genomics Viewer). Visual inspection of randomly selected alignments can be used to manually validate mutation reporting done in the next steps of the pipeline.

Run instructions:

a) Check that the "factors_table.txt" contains correct values in the columns: "sort_refs", "sort_ref". In case of incorrect values - adjust the table and **repeat step 2** described above to produce correct bash parameter file, before starting this step.

b) **Run script:** *bash create_dummy_genome5.sh 1*

c) Inspect the output files in the output folder under "mapping" directory.

Output description:

This script adds following files to the "mapping" directory:

1) **<sample_name>.<ref_name>.bwa.sam.header.barcode** - A list of all unique 5' barcode sequences belonging to the reads aligned against given reference sequence.

2) **<sample_name>.<ref_name>.bwa.sam.header.barcode.fasta** - A FASTA file where each unique 5' barcode sequence is used as the header of the reference sequence to which the reads were aligned and is followed by the reference sequence itself. The resulting file looks like this:

...

>5_BC_SEQ_1 Reference_name

Reference_sequence

>5_BC_SEQ_2 Reference_name

Reference_sequence

...

This creates a match between reference sequence headers and reference names in the RNAME field of the modified SAM/BAM file which is needed for IGV to visualize read families sharing the same 5' barcode against the correct reference sequence. Using 5' barcode sequences as read identifiers allows read grouping by families originating from the same stretch of donor DNA.

8) Listing mutations per read

Step description:

This step checks mutation presence in each read and outputs mutated positions and their sequencing quality. Each mutation is listed in a separate row. If no mutations are found the read is designated 'WT'.

By default, the output of this step is not utilized by the pipeline in the next steps. However, it can be used to filter mutations from mutation table produced in step 9 (see below), based on user-defined mutation quality criteria. This functionality is not included in the pipeline and would require external tools/scripts supplied by the user.

Note:

1) The script searches for mutations in a user defined window of interest. 'WT' designation means that no mutations were found in this interval and not along the whole read.

2) The script checks sequencing quality of mutated positions using "phred+33" scoring scheme. If different score is used, the **"phred" variable** in the **"sam_to_mutation-list-3.py"** script needs to be adjusted accordingly.

3) The script requires that the query and the reference sequences align at their first position (reference start and query alignment start, as listed in the SAM file, are '0') for the alignment to be included in the analysis. Alignments with non-first-base start for either reference or query are skipped.

Run instructions:

a) Check that the "factors_table.txt" contains correct values in the columns:

“limit_starts”, “limit_ends”, “sort_refs” and “sort_ref”. In case of incorrect values - adjust the table and **repeat step 2** described above to produce correct bash parameter file, before starting this step.

b) **Run script:** *bash sam_to_mutation-list-3.sh 1*

c) Wait for the job to end. Use the “sacct” and the “squeue” commands on the cluster to check job status.

d) Inspect the output files in the output folder under “mutations” directory.

Output description:

This step creates a list of mutations per read in the analyzed reads. It outputs the following files:

1) **<sample_name>.err, <sample_name>.out** - Log files produced during the run of the job on the cluster. Check that the “.err” file doesn’t contain error messages and that the last line in the ”.out” file is “ok” to indicate that all reads were analyzed.

2) **<sample_name>.<ref_name>.bwa.sorted.bam.mutations.log.txt** - A log file listing the total number of reads in the analyzed file, numbers of reads that were analyzed and number of reads that were skipped, either because they are unmapped or because read-reference alignment doesn’t start at the beginning of the reference and/or the read. Before continuing to the next pipeline steps note that most reads in the sample are processed by the script and not filtered out due to various reasons.

3) **<sample_name>.<ref_name>.bwa.sorted.bam.mutations.txt** - A list of mutations found in the analyzed reads, relative to the reference. Each mutation is listed in a separate line. The list contains the following information:

1) **read_name** - ID of the read in which mutation was found (e.g. M00654:20:000000000-J2GB9:1:1101:12401:9036).

2) **bc5** - sequence of the 5’ barcode associated with the read in which mutation was found.

3) **bc5_count** - number of reads having 5’ barcode listed in the ‘bc5’ field (read family size).

4) **bc3** - sequence of the 3’ barcode associated with the read in which the mutation was found.

5) **quality** - sequencing quality of the position in the read in which the mutation was found. For 'WT' reads and deletions this field gets a value of 'NA'.
6) **pos** - Position of the mutation in the reference sequence, relative to the first base of the sequence. For 'WT' reads this field gets a value of 'NA'.
7) **from** - Nucleotide found in the reference sequence at the position listed in the 'pos' field. Insertions in the read relative to the reference are marked by '-'. For 'WT' reads this field gets a value of 'NA'.
8) **to** - Nucleotide found in the query sequence (read) at the position listed in the 'pos' field. Deletions in the read relative to the reference are marked by '-'. For 'WT' reads this field gets a value of 'NA'.
9) **allele** - Full name of the mutated allele, listing the name of the SNP and its position (e.g.: 39CT). If no mutations are found - this field gets a value of 'WT'.

9) Listing mutations per read family

Step description:

This step groups analyzed reads into families based on their unique 5' barcode sequence and examines each family for presence of mutations. It outputs a list of mutations per each family, with each mutation listed in a separate row. Reads containing no mutations in each family are designated 'WT' and are outputted in a separate row.

The users can choose to process only a subset of aligned reads, using two types of filters:
1) **3' barcode type** - by modifying the line 78 ("for sam_BX_tag in "; do") in the **"sam_to_mutation-table_5.2.sh"** script the user can tell the script to process reads only having specific secondary barcode. By default, " " " tells the script to process all BC3 types. This option is useful to collect and investigate reads having "BC3_missing" tag, to understand why attachment of the secondary barcode has failed in them, if the pipeline is configured to produce such reads.

2) **BC3 minimum count** - specifies minimal number of different 3' barcodes to be associated with a specific mutation for it to be reported. Requiring more secondary barcodes to be associated with a particular mutation can increase confidence in its validity. By default, one barcode type is required, since in the next step observed

mutations are compared against a set of cut-offs to determine "consensus" mutations per read family (see below), which allows filtering out low confidence mutations. **This threshold can be changed by updating "BC3_min" field in the "params_1.sh" parameter file.**

Note:

1) The script searches for mutations in a window of interest defined by the user. 'WT' designation means that no mutations were found in this interval and not along the whole read.

2) The script checks sequencing quality of mutated positions using "phred+33" scoring scheme. If different scoring scheme is used, adjust the following lines in the **"sam_to_mutation-table_5.2.py"** script: "ord(asciiChar)-33" and "ord(qual1)-33" to the relevant scoring scheme. Also, update the line "min_phred33=28" with the relevant score value to denote high quality mutations.

3) Currently the script requires that the query and the reference sequences align at their first position (reference start and query alignment start, as listed in the SAM file, are '0') for the alignment to be included in the analysis. Alignments with non-first-base start for either the reference or the query are skipped.

Run instructions:

a) Check that the "factors_table.txt" contains correct values in the columns: "seq_pos", "seq_mut", "seq_action", "limit_starts", "limit_ends", "sort_refs" and "sort_ref". In case of incorrect values - adjust the table and **repeat step 2** described above to produce correct bash parameter file, before starting this step.

b) **Run script:** *bash sam_to_mutation-table_5.2.sh 1*

c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

d) Inspect the output files in the output folder under "tables.BC3cutoff1" directory.

Output description:

This step outputs information regarding mutated positions and WT reads in each read family (group of reads sharing same 5' barcode). It outputs the following files:

1) **<sample_name>.<ref_name>.bwa.sorted.bam..err, <sample_name>.<ref_name>.bwa.sorted.bam..out** - Log files produced during the run of the job on the cluster. Check that the ".err" file doesn't contain error messages and that the last line in the ".out" file is "OK" to indicate that all reads were analyzed. Additionally, the ".out" file reports reads in which mutation was found at the position of control insertion. If the mutation matches expected control insertion, it would be denoted as "True", otherwise it will be denoted as "False". By counting the number of reads containing control insertion, it is possible to estimate the extent of relabeling occuring in the analyzed sample.

2) **<sample_name>.<ref_name>.bwa.sorted.bam.mutationFrequncyPerBarcode.log.txt** - A log file listing the statistics of the processed reads:

a) reads unmapped - Count of unmapped reads in the analyzed sample. The pipeline is designed to filter out unmapped reads during the read mapping step (step 6 above), therefore unmapped reads are not expected to be present here. The goal of this check is to validate that all unmapped reads are indeed filtered out prior to mutation calling.

b) reads start_ne0, reads ref_start_ne0, reads query_start_ne0 - Count of reads that were skipped because the read (**reads query_start_ne0**) or the reference (**reads ref_start_ne0**) do not map to each other from their first position. "**reads start_ne0"** shows total count of such reads. **Note** that "**reads start_ne0"** count **is not** a sum of **"reads ref_start_ne0"** and **"reads query_start_ne0"** counts, since some alignments fall in both categories.

c) reads total - Total number of reads that were processed during mutation calling step.

d) reads used - Number of reads used for mutation calling, after filtering out unmapped reads and alignments not starting at first position of the read and/or the reference.

Before proceeding to the consensus mutation calling step it is important to check that majority of analyzed reads are used for mutation calling and are not filtered out due to some alignment issues. Also, validate that statistics of processed reads are similar between this step and log files produced in step 8, since both steps use the same filters

to process or discard alignments.

3)

<sample_name>.<ref_name>.bwa.sorted.bam.mutationFrequncyPerBarcode.txt

- File listing information on mutations found in the analyzed read families, relative to the reference. Each mutation within read family is listed in a separate line. Mutation list contains the following information:

1) **ID** - A unique 5' barcode sequence serving as the identifier of given read family.

2) **Mutation** - The name of the mutated allele, shown as **<mutation_position_reference><reference_nucleotide><read_nucleotide>**, e.g.: 57G-. Reads not containing mutations in the interval of interest defined by the user are designated 'WT'. INDELS are indicated by hypen ('-').

3) **Mutation_count** - Number of reads in the 5' barcode family containing mutation specified by the "Mutation" field.

4) **HQ_mutation_count** - Number of reads in the 5' barcode family containing mutation specified by the "Mutation" field when sequencing quality of the mutated position is above high-quality threshold (default - 28, using "phred+33" scoring scheme). Deletions and WT reads are considered high quality by default.

5) **Total_count** - Total number of reads in the 5' barcode family (mutated and WT).

6) **HQ_total_count** - Total number of reads in the 5' barcode family with high sequencing quality at the position of mutation listed in the "Mutation" field (both mutated and WT). High quality threshold used here is the same as for **HQ_mutation_count**.

7) **Mutation_freq** - A proportion of reads containing mutation at given position out of all reads in the family, when only high-quality positions are considered. Calculated as: **HQ_mutation_count/HQ_total_count**.

8) **barcode3_IDs** - A semi-column separated list of 3' barcode sequences associated with reads containing high-quality mutated position defined in the "Mutation" field.

9) **barcode3_HQ_mut_counts** - Number of reads in each 3' barcode group listed in the "barcode3_IDs" field that contains high-quality mutated position. Deletions and WT reads are considered high quality by default.

10) **barcode3_existing** - Total number of different 3' barcode groups associated with the high-quality mutated position. Doesn't include reads with the "BC3_Missing" tag.

11) **barcode3_missing** - A field indicating if any of the reads carrying high-quality

mutated position lacks 3' barcode. Has value of '0' if no reads exist and '1' if one or more reads exist.

Note that by default this field is expected to contain only '0' values, because during barcode trimming step (step 5) barcodes attached to the reads are checked for presence of correct sample IDs in the primary and secondary barcodes. Since reads with missing 3' barcode wouldn't have a correct sample ID for the secondary barcode, they would be filtered out, unless sample ID check is requested only for primary (5') barcodes.

12) **barcode3_HQ_mut_freq** - A proportion of reads in each 3' barcode group containing mutation defined by the "Mutation" field out of all reads in the read family, when only high-quality positions are considered. Calculated as: **barcode3_HQ_mut_counts/HQ_total_count**. Note that the sum of mutation frequencies across different 3' barcode groups should be equal to the total mutation frequency in the read family, as defined in the **"Mutation_freq"** field.

13) **barcode3_all** - A semi-column separated list of 3' barcode sequences associated with reads in a given 5' barcode read family (both mutated and WT).

14) **barcode3_all_counts** - Number of reads in each 3' barcode group listed in the "barcode3_all" field.

15) **barcode3_all_WTcounts** - Number of 'WT' reads in each 3' barcode group belonging to a given 5' barcode read family. For a 'WT' family this field gets a value of 'NA'.

16) **barcode3_all_WTcount** - Number of different 3' barcode groups associated with the high-quality mutated position that contain also 'WT' reads at this position. Doesn't include reads with the "BC3_Missing" tag. For a 'WT' allele this field gets a value of 'NA'.

17) **barcode3_missing_all_WTcount -** This field indicates if among reads lacking 3' barcode that are associated with the high-quality mutation (as listed in the 'Mutation' field) some reads are 'WT' at this position. It receives value of '0' if no 'WT' reads are present and '1' if one or more 'WT' reads are present. For a 'WT' allele this field gets a value of 'NA'.

10) Identifying consensus mutations

Step description:

This step compares mutation lists produced in the previous step to a set of user

defined cutoffs to decide which mutations are "true" and which might represent experimental artifacts. A detailed explanation on various cutoffs used here is given in a substep 10-2.

Step 10-1: Collating mutations per read family

This substep prepares mutation lists produced in step 9 for consensus analysis. In the previous step each mutation per read family was outputted in a separate line. Here this information is collated together.

By default, all mutation data produced in the previous step is processed here. This is controlled by the statement **"procede1=True"** (line 99 in the **"consensus_15.py"** script). Putting **"procede1"** variable under condition statement, so it receives "True" value only for specific data combinations, allows fine-tuned collation of mutation data. Conditional data processing option is not included in the pipeline and requires script adaption by the user, according to their needs.

Run instructions:

a) Check that the "params_1.sh" file contains correct value for the 'TSS' and 'filter_pos' variables.

b) **Run script:** *bash consensus_15.1.sh 1*

c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

d) Inspect the output files in the output folder under "tables_consensus.BC3cutoff1" directory.

Output description:

This substep produces a collated list of mutations per read family. In this substep the following files are added to the output directory:

1) **<sample_name>.<ref_name>.bwa.sorted.bam.consensus.err, <sample_name>.<ref_name>.bwa.sorted.bam.consensus.out** - Log files produced during the run of the job on the cluster. Check that the ".err" file doesn't contain error messages. Check that the last line in ".out" file contains the word "Done", to indicate

that the job was completed successfully. Also, check the ".out" file for comments on read families:

a) "Two types of mutations at the same position in X" - indicates that in family X (X stands for primary barcode serving as family ID) two types of mutation were found at the same position. When analyzing ultra-rare variants, it might be less likely to encounter two different mutation types at the same position, hence the reporting.

b) "skipped ID=X" - indicates that family X was excluded from consensus analysis. Read families are removed from the consensus analysis if they contain only low quality mutations (mutations at positions that doesn't pass sequencing quality threshold, as defined in step 9). If such family contains both 'WT' and mutated alleles - the 'WT' alleles are included in the consensus analysis, while mutations are filtered out. Tracking skipped families allows estimation of data loss due to presence of families with low quality mutations only.

c) "Frequency value outside [0,1] range for ID=X" - indicates that in family X frequency of one or more mutations was outside [0,1] range. Since number of mutation-containing reads can not be greater than the total number of reads in the family, presence of such message indicates bug in the calculation of mutation frequency that was done in the previous step. If such message appears in the log file, please report the issue to the authors.

2) **<sample_name>.<ref_name>.bwa.sorted.bam.consensus.txt** - A collated list of mutations found in the analyzed read families, relative to the reference. Collated mutation list contains the following information:

1) **Barcode** - A unique 5' barcode sequence serving as an identifier of given read family.

2) **Consensus** - A semicolon separated list of all high-quality mutations in the given read family (or 'WT' for reads not containing any mutation).

3) **Consensus_TSS** - A semicolon separated list of all high-quality mutations in the given read family, where position of each mutation is given relative to the translation start site defined in the "params_1.sh" file by the "TSS" variable. For 'WT' reads this field gets a 'WT' value.

4) **Mutation_freqs** - This field lists frequency of each allele appearing in the "Consensus" field, when only high quality positions are considered for frequency calculation (see explanation on **"Mutation_freq"** in step 9). If only 'WT' reads are

present in the read family, this field gets a value of '0'.

5) **Mutation_counts** - For each allele in the "Consensus" column this field lists number of reads in the family containing this allele, when sequencing quality of the position carrying the allele is above high-quality threshold (default - 28, phred+33). If only 'WT' reads are present in the read family, this field gets a value of '0'.

6) **total_count** - Total number of reads in a given 5' barcode family (both mutated and WT).

7) **Positions** - Count of different mutation types found in the read family. If only 'WT' reads are present in the read family, but no high-quality mutations, this field gets a value of '0'.

8) **WT_freqs_in_mutations** - This field lists frequencies of 'WT' alleles in the read family for each position where high-quality mutation was found. Calculated as: **WT_freqs_in_mutations = 1.0 - Mutation_frequency** for each mutated position.

If only 'WT' reads are present in the read family, but no high-quality mutations, **WT_freq** calculated as proportion of 'WT' reads out of total number of reads in a given read family.

9) **barcode3_existing** - This field lists how many different 3' barcode groups are associated with each mutated allele (or 'WT' sequence) listed in the "Consensus" fields. Reads carrying same 3' barcode sequence are counted as a same 3' barcode group.

10) **barcode3mut_above1reads** - This field lists how many different 3' barcode groups containing more than user specified number of reads are associated with each mutated allele listed in the "Consensus" field. If only 'WT' reads are present in the read family, but no high-quality mutations, this field gets a value of 'NA'.

11) **barcode3WT_above1reads** - This field lists how many different 3' barcode groups associated with each mutated allele listed in the "Consensus" field contain more than user specified number of reads with 'WT' nucleotide at this position. If only 'WT' reads are present in the read family, but no high-quality mutations, this value equals to the values listed in the "barcode3_above1reads" field (see below).

12) **barcode3_above1reads** - This field lists how many different 3' barcode groups within given 5' barcode read family contain more than more than user specified number of reads (mutation- or WT-linked). Since here **all** 3' barcode groups in the family are counted, this value should be the same for all alleles in the family.

13) **barcode3_all_WTcount** - For each mutation listed in the "Consensus" field this

field lists count of different 3' barcode groups in the read family that contain 'WT' allele at this position.

14) **unmutatedReads_counts** - This field lists total count of 'WT' reads in a given read family (reads that doesn't contain any mutations in the user defined interval of interest, as explained in step 9).

15) **unmutatedReads_freq** - This field lists frequency of 'WT' reads in a given read family (proportion of 'WT' reads counted in the **"unmutatedReads_counts"** field out of total number of reads in the family).

Note: The minimal size of 3' barcode groups to be counted in the **"barcode3mut_above1reads"**, **"barcode3WT_above1reads"** and **"barcode3_ above1reads"** fields is **"BC3_min+1"**. The **"BC3_min"** parameter is defined in the **"params_1.sh"** parameter file. To re-run the analysis with a different "BC3_min" parameter, run this substep again. In case of re-run, remember to move already existing output to a different folder or it will be overwritten.

Step 10-2: Detecting consensus mutations

This substep determines consensus between the "WT" and variant alleles at each mutated position for each read family. Collated mutation list, produced in the substep 10-1, is compared against a set of user defined thresholds to determine "consensus". Positions where both mutated and 'WT' alleles do not pass the thresholds are considered ambiguous and are outputted as 'N'.

Run instructions:

a) Check that the "factors_table.txt" contains correct value in the column: "size_r". In case of incorrect values - adjust the table and **repeat step 2** described above to produce correct bash parameter file, before starting this step. Check that cut-off criteria is properly defined in the "params_1.sh" (see **"Cut-off criteria description"** below).

b) **Run script:** *bash consensus_15.1.sh 2*

c) Wait for the job to end. Use the "sacct" and the "squeue" commands on the cluster to check job status.

d) Inspect the output files in the output folder under:

"tables_consensus.BC3cutoff1/<sample_name>.<ref_name>.bwa.sorted.bam.cutoff

s-update”

Cut-off criteria description:

The cut-off criteria used to identify consensus mutations can be set via the parameter script “params_1.sh”. If the user provides multiple cut-off values for the analyzed criteria, all possible combinations of the cut-off criteria would be analyzed. The following criteria can be set for analysis:

1) **min_count** - minimal 5’ barcode read family size to be considered for analysis. Read families having less reads than defined by this threshold are not analyzed. Multiple read count cut-offs should be supplied in a form of a comma-separated list, e.g.: *min_count="0,1,2"*

2) **min_freq** - minimal frequency of mutated or WT allele in the read family for it to be considered as “true”. Multiple mutation frequency cut-offs should be supplied in a form of a comma-separated list, e.g.: *min_freq="0.0,0.1,0.2"*.

3) **minCountBarcode** - minimal number of different 3’ barcodes that should be associated with a given allele within the read family for it to be considered as “true”. Multiple minimum 3’ barcode count cut-offs should be supplied in a form of a comma-separated list, e.g.: *min_bc3="0,1,2,3,4,5"*.

4) **bc3groupCountOK** - minimal number of different 3’ barcodes that should be associated with given allele within the read family for it to be considered as “true”. This criteria should be supplied as a single integer, e.g.: *bc3groupCountOK=0.* Alleles need to pass either this **or** “bc3groupsWithReadsAbove1” thresholds to count as “consensus”.

5) **bc3groupsWithReadsAbove1** - minimal number of different **large** 3’ barcode groups that should be associated with given allele within the read family for it to be considered as “true”. This criteria should be supplied as a single integer, e.g.: *bc3groupsWithReadsAbove1=0.* Alleles need to pass either this **or** “bc3groupCountOK” thresholds to count as “consensus”.

The minimal size of 3’ barcode group that is required for it to be counted as large group is **“BC3_min+1”**. The **“BC3_min”** parameter is defined in the **“params_1.sh”** parameter file. Counts of large 3’ barcode groups in the read family are added to the collated mutation table in the substep 10-1 described above. To re-run the analysis with a different **“BC3_min”** parameter re-run the substep 10-1 again before running this substep. In case of re-run, remember to move already existing output to a

different folder or it will be overwritten.

Notes on 3' barcode criteria: The full condition for checking 3' barcode group information is [*minCountBarcode* ***and*** *(bc3groupCountOK* ***or*** *bc3groupsWithReadsAbove1)*]. The "**minCountBarcode**" variable contains an array of threshold values against which the alleles are checked, while "**bc3groupCountOK**" and **"bc3groupsWithReadsAbove1"** hold a single value. Thus, if "**minCountBarcode**" >= "**bc3groupCountOK**" - any allele passing one threshold would also pass the other one. Otherwise the allele would need to pass either "**bc3groupsWithReadsAbove1"** or "**bc3groupCountOK**" thresholds to be considered as "consensus" (assuming that it has passed all other thresholds).

By adjusting these variables, the user can fine-tune the process of mutation vetoing based on the secondary barcode criteria. The common use is to have higher value for "**bc3groupCountOK**" variable and lower one for "**bc3groupsWithReadsAbove1"**. From data perspective this means that either larger number of 3' barcode groups of any size, or smaller number of large 3' barcode groups is required to consider the particular allele as "consensus". The aim is to reduce the possibility that identified allele is an artifact and not a real mutation by ensuring that each mutation counted as "consensus" is supported by large enough number of reads from different 3' barcode groups.

When exploring new data, it might be advisable to set value of both variables to '0' to get better sense of data distribution before applying more stringent filters to it.

Output description:

The consensus calling substep stores its output within the **"<sample_name>.<ref_name>.bwa.sorted.bam.cutoffs-update"** folders inside the "tables_consensus.BC3cutoff1" directory. A separate results folder is created for each collated mutation table generated in the substep 10-1. Within each folder two files are created for each tested combination of threshold values:

1)

<sample_name>.<ref_name>.bwa.sorted.bam.mutFreq<W>_readCount<X>_BC3WithMut<Y>_BC3above<Z>.txt - This file contains a list of "consensus" mutations that passed the thresholds. Its structure is the same as of the collated mutation list file created in the substep 10-1.

A separate file is created for each combination of cut-off criteria used to determine "consensus" mutations. The cut-off values are included in the filename:
a) **mutFreq<W>** - mutation frequency threshold.
b) **readCount<X>** - read family size threshold.
c) **BC3WithMut<Y>** - threshold of different 3' barcode groups associated with the allele within the read family.
d) **BC3above<Z>** - threshold of different 3' barcode groups associated with the allele within the read family that have above user-specified number of reads in each group.

In the "consensus" list the consensus is determined between mutated and 'WT' alleles in the 5' barcode family at each position of mutation. If at any position the mutated allele is rejected, because it didn't pass the combined thresholds, but 'WT' allele is accepted - this position would be considered as 'WT' and **wouldn't be reported** in the consensus mutation list. If in a read family **all** mutations are rejected, but their associated 'WT' alleles are **all** accepted - the family would be reported as 'WT' in the consensus list.
If both mutation and 'WT' alleles are not accepted - the position would be considered as ambiguous and reported as 'N' in the consensus mutation list, e.g.: 39CT -> 39N. If multiple mutations at the same position are found to be ambiguous, they would be represented by a single 'N' entry and their associated information (such as mutation frequencies) would be summed up.
If the family contains only 'WT' reads, but no mutations, it is included in the consensus list only if it passes all the thresholds. Otherwise, it is not reported.

2)
<sample_name>.<ref_name>.bwa.sorted.bam.mutFreq<W>_readCount<X>_BC3WithMut<Y>_BC3above<Z>.cons-count.txt - The accompanying file of the consensus mutation list that counts number of 5' barcode families in the list having particular mutation profile. In the left column it lists all mutation profiles found in the consensus list and in the right column - their count. 'WT' family count is listed in a separate row. In addition, it provides information on rejected families:
a) **rejected.low_count** - read families that were rejected because they didn't pass read

count threshold (size of the 5' barcode family is too small).
b) **rejected.from_WThaplotype** - 'WT' read families that were rejected because they didn't pass the 3' barcode group thresholds.
c) **rejected.other** - read families that were rejected for reasons different from above.

In addition, cluster run log files are created under the "tables_consensus.BC3cutoff1" folder in the output directory:
<sample_name>.<ref_name>.bwa.sorted.bam.consensus.cutoffs-update.err,
<sample_name>.<ref_name>.bwa.sorted.bam.consensus.cutoffs-update.out

Upon run completion check that the ".err" files do not contain any error messages and that the ".out" files contain the word "Done" to indicate that the job was completed successfully.
Additionally, check that the ".out" files contain correct cut-off criteria parameters. If not – fix the parameters inside the "params_1.sh" file and re-run the substep.