GIbPSs 1.0 Manual Part 4: Documentation of fdm_07.0.pl

5/22/2015 Andreas Hapke,
Institute of Anthropology, Johannes Gutenberg University Mainz, Germany, ahapke2@gmail.com

This program **f**ilters, **d**emultiplexes, and **m**odifies your sequence data. It analyzes a set of fastq-files and produces fastq files that are ready for analysis with **indloc**, the next program in the GIbPSs package. **fdm** is a very flexible tool that offers numerous options to process your sequence data:

fdm can

-analyze single (forward) reads or paired (forward and reverse) reads,

-analyze gzip compressed fastq infiles or plain text files,

-produce gzip compressed fastq outfiles or plain text files,

-filter sequences with low Phred scores with a sliding window analysis,

-demultiplex sequences according to inline barcodes at the beginning of forward reads with mismatch and repair,

-check restriction sites at the beginning of sequences or after the barcode with mismatch and repair,

-detect internal restriction sites,

-search common-adapters in forward reads and barcode adapters in reverse reads with mismatch,

-construct *rc-duplicates* from paired reads,

-truncate sequences before and/or after analysis.

Barcodes may have different lengths. You can specify any restriction site as a sequence that may contain ambiguity characters. The program can search for different restriction sites in forward and reverse reads.

Usage

The program needs a settings-file with your instructions how to process your data and where to find them. The settings-file must be in the same directory as the program. See the example file `settings_fdm.txt`. All settings are explained in detail below. I recommend that you create a directory, e.g. `fdm`, and store a copy of the program and your settings-file therein. Open a command prompt, cd to the directory, and call the program with the name of the settings-file as argument:

```
fdm_07.0.pl settings_fdm.txt
```
or
```
perl fdm_07.0.pl settings_fdm.txt
```

The program starts, creates a subdirectory named after the settings-file (here: `fdm/settings_fdm_out`) and stores outfiles therein.

You will need to prepare some additional files before you start the program. These files are explained below. The settings-file contains the names of these files. When a file is stored in the same directory as the program, **fdm** just needs the filename. Else, it needs the full path to the file.

Format of the settings-file:

The file contains comments that start with "#". Do not remove the "#". Blank lines are allowed. Settings consist of a variable-name and one value, separated by one or more whitespace characters. Values can be Strings (S), Booleans (B) (1 for True and 0 for False), Integers (I) or Decimals (D). The appropriate type of value is explained in the comment for each variable. Do not modify the names of the variables. If a name is not recognized or if there is more than one value, the program displays an error message and stops execution.

Algorithm overview

The flowchart in Figure 1 illustrates the steps taken by the program when processing sequence data. Boxes are functions, which you can activate or inactivate. Relevant setting variables for each function are written in italics close to the boxes.

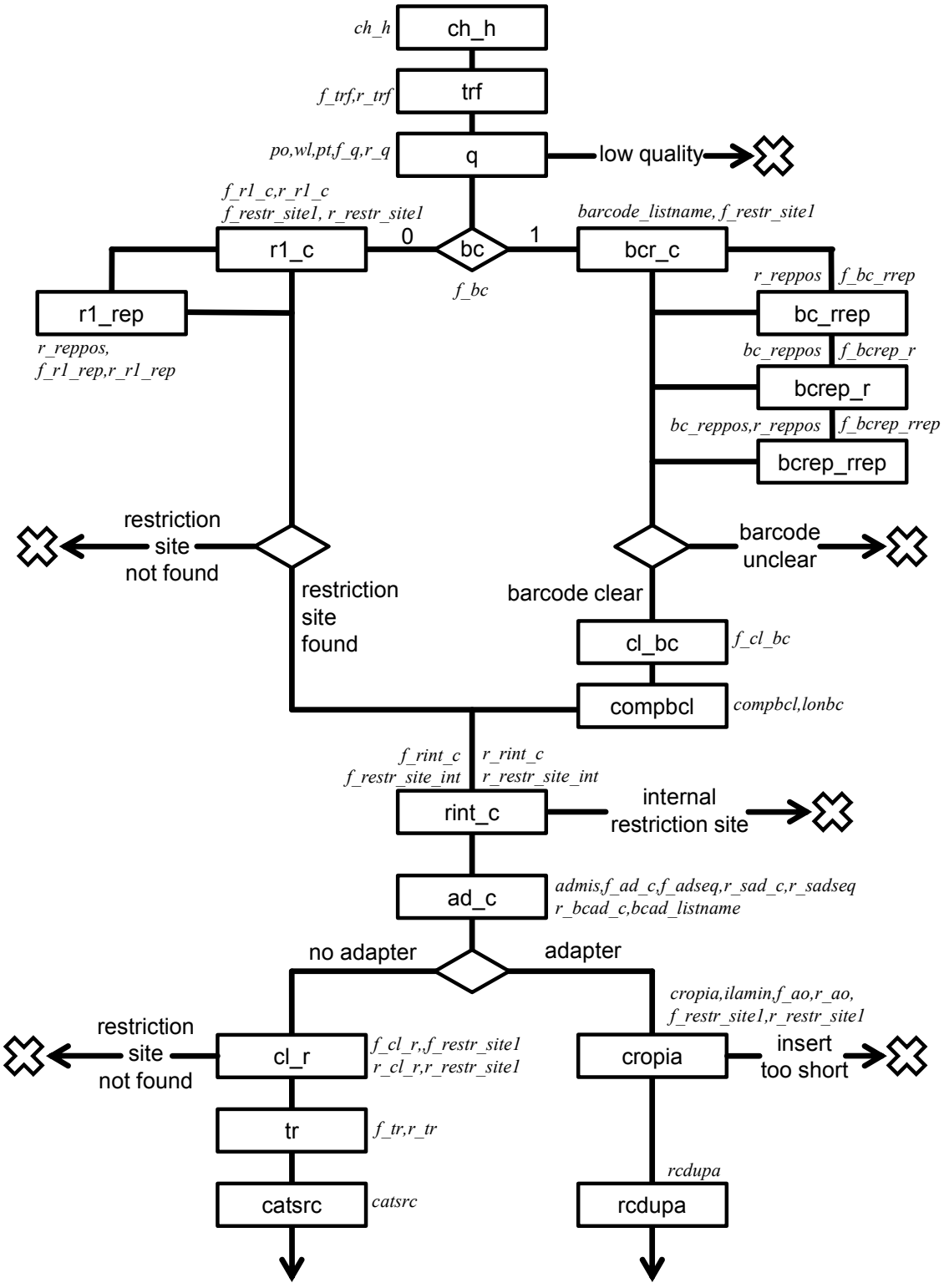

Figure 1: Algorithm overview. The flowchart illustrates the order of steps taken by the program. Boxes are functions. Relevant setting variables for each function are written in italics.

ch_h (change header) changes Illumina fastq-headers to a simpler format.

trf (truncate first) truncates sequences before the analysis to a configurable length.

q (qality check) performs a sliding window analysis of average Phred scores and discards sequences that do not pass.

bc (barcodes)

Forward reads go right here when barcode demultiplexing is active; else, they go left. Reverse reads always go left.

Rigth way: barcode demultiplexing

The next four functions identify barcodes, which must be followed by a specified restriction site.

bcr_c (barcode-restriction-site check)

This function identifies barcode-restriction-site combinations without mismatch. When it fails to identify the barcode, the next functions try to identify barcode-restriction-site combinations with configurable mismatch and repair. You may activate all three or just one or two of them. The program calls them in the order shown in the flowchart. As soon as it has identified the barcode, it skips the remaining repair functions.

bc_rrep tries to repair the restriction site.

bcrep_r tries to repair the barcode.

bcrep_rrep tries to repair the barcode and the restriction site.

A sequence is discarded when the barcode is still unclear now.

cl_bc (clip barcode) removes the barcode.

compbcl (compensate barcode length) truncates sequences with shorter barcodes to the length of sequences with the longest barcode after barcode removal. This function ensures equal chances to detect adapters in sequences with different barcodes.

Left way: sequences without barcodes: optional restriction site check

r1_c checks if the sequence begins with a specified restriction site (without mismatch). If this is not the case,

r1_rep tries to repair the restriction site.

The sequence is discarded, when **r1_c** and/or **r1_rep** are active and the first characters of the sequence still do not match the restriction site now.

rint_c checks for an internal restriction site and discards the sequence if it finds one.

ad_c checks for adapter sequences. Sequences that contain a full or partial adapter go right. Others go left.

cropia crops the insert of adapter containing sequences, i.e. removes the adapter and all characters that are part of the restriction site at both ends of the sequence unless the cropped insert would be shorter than a configurable minimum. In the latter case, it discards the sequence.

rcdupa produces *rc-duplicates* from cropped inserts of adapter containing sequences. This option is available for paired reads only.

cl_r removes the restriction site from the beginning of the sequence. It discards the sequence when it does not find the restriction site (without mismatch).

tr truncates the sequence to a specified length.

catsrc produces *rc-duplicates* from paired reads that do not contain adapters.

Special features for paired reads

Reverse reads are demultiplexed according to inline-barcodes in forward reads.

If forward and reverse read pass all filters, they can be saved inline in paired outfiles or consecutively in one outfile.

If only one of them passes all filters, it is saved in a separate file, e.g. for the corresponding barcode.

If an internal restriction site is detected in one read, both are discarded.

If an adapter is detected in one read, both reads are treated as containing adapters.

If the cropped insert of an adapter-containing read is shorter than a configurable threshold, both reads are discarded.

Settings

The example file `settings_fdm.txt` contains settings for a typical processing of paired reads from a GBS library for subsequent analysis with **indloc**. Many functions depend on each other. For instance, you must search for adapters before you can crop the insert of adapter containing sequences. You must crop the insert from adapter containing sequencing in order to produce useful *rc-duplicates*. It is entirely up to you to make meaningful settings. The program does not verify your settings. Most functions (except for `trf` and `tr`) are activated/inactivated by Boolean variables (1 for True/activate, 0 for False/inactivate). Variable types are indicated as follows:

```
#S String
#B Boolean
#I Integer
#D Decimal
```

MAIN SETTINGS

`report_filename #S`

Name of the report-file. The report file contains counts of sequences that passed/did not pass different filters and that have been sorted into different categories (e.g. according to barcodes).

`fastq_listname #S`

Name of a text file containing a list of fastq-infiles. Single reads: one filename per line, paired reads: two filenames per line (forward-reads, reverse-reads), separated by any number of whitespace-characters. If the fastq infiles are stored in another directory than **fdm**, replace the filenames with full pathnames. Paired reads must appear in corresponding order in the paired infiles. The fastq infiles may be gzip compressed files or plain text files. Names of gzip compressed files must have the extension `".gz"`. Plain text files may have any other or no extension. Each gzip compressed file must contain one single fastq file. The program cannot unpack a tarball.

Format of the fastq-infiles: Four lines per read (header, sequence, third line with anything, phred-symbols), each line ending with newline, no additional blank lines.

`P #B`

Paired reads in paired fastq-infiles. `#B` means this is a Boolean. Type 1 for True or 0 for False.

`save_f #B`

Save forward reads. If true (1), the program produces fastq-outfiles with forward reads. Type 0 if you just need the report file, e.g. to test settings.

`save_r #B`

Save reverse reads. If true (1), the program produces fastq-outfiles with reverse reads. Inactivate this option to save disk space. **indloc** does not need the fastq-outfiles from reverse reads. You can construct *rc-duplicates* with **fdm** and then save only the forward reads. **indloc** is able to construct the reverse complements of the forward reads for an *rc-duplicate* analysis.

`save_d #B`

Save discarded reads. If true (1), the program produces fastq-outfiles with discarded reads. Inactivate this option to save disk space.

`adsf #B`

Save sequences containing adapters to separate outfiles. **indloc** accepts one fastq-infile per individual and can handle sequences of different lengths. Inactivate `adsf` and activate `cropia` (see below) to analyze all sequences together with **indloc**.

`r1r2sf #B`

Save sequences from forward and reverse reads to separate outfiles. This variable is meaningless unless `save_f` and `save_r` are active. When `r1r2sf` is active, forward and reads are saved inline in paired outfiles with extensions *_R1.fq and *_R2.fq. When it is inactive, the two reads of a pair are saved consecutively (forward first) in the same outfile with a name ending with *_R12.fq. If one of the two reads is discarded, the other one is always saved to a separate outfile with a name containing *_sg_*.

`z #B`

Use `z 1` to produce gzip compressed fastq outfiles if you wish to save disk space. Use `z 0` to produce plain text fastq outfiles, which is a little bit faster. The next program, **indloc**, can read both types of fastq files. Again, **indloc** is a bit faster with plain text infiles than with gzip compressed infiles.

`max_p #I, must be 1 or greater`

Maximum number of processes to execute in parallel. The program can process several infiles in parallel, which speeds up computation. I recommend that you test this variable carefully starting with small values and gradually increasing them.

`ch_h #B`

Change fastq headers to a simpler format (the same format as used by STACKS). You must activate this option when you plan to use **pair_finder**. Your original fastq-headers must be in Illumina format: example:

`@HWI-ST558:200:C3BVRACXX:4:1101:1178:2228 1:N:0:ATCACG`

`ch_h` changes this header to

`@4_1101_1178_2228_1`

When you have paired reads, the corresponding reverse read will have this header:

`@4_1101_1178_2228_2`

The headers are identical except for _1 or _2 in the fifth field. **fdm** always uses 2 as number for the reverse read, even when your original fastq-header contains another number.

QUALITY CHECK BY SLIDING WINDOW ANALYSIS

This analysis moves a sliding window along the sequence with a step length of 1 position and calculates the average Phred score in each window. A sequence is discarded when the average Phred score in any window is smaller than a configurable threshold.

`f_q #B` forward reads: activate sliding window quality check

`r_q #B` reverse reads: activate sliding window quality check

`po #I` Phred score offset (33 for Sanger).

`pt #I` Phred score threshold

`wl #I` Window length

TRUNCATION OF SEQUENCES

`f_trf #I` and `r_trf #I`

Truncate forward reads (f_) or reverse reads (r_) to a specified length before any analysis. A value of 0 inactivates these functions.

`f_tr #I` and `r_tr #I`

Truncate forward reads (f_) or reverse reads (r_) to a specified length after analysis before construction of *rc-duplicates* (catsrc). A value of 0 inactivates these functions. These functions act on sequences that do not contain adapters.

RESTRICTION SITES: CHECK AND REPAIR

`f_restr_site1 #S` and `r_restr_site1 #S`

These variables specify the restriction sites at the beginning of forward and reverse reads or after the barcode in forward reads. Examples:

`BamHI (G^GATCC):`

`f_restr_site1 GATCC`

`ApeKI (G^CWGC):`

`f_restr_site1 CWGC`

The program evaluates all possible permutations of degenerate recognition sites (with IUPAC ambiguity symbols).

`f_r1_c #B` and `r_r1_c #B`

activate restriction site checks in forward and reverse reads.

`f_r1_rep #B` and `r_r1_rep #B`

activate restriction site repair in forward and reverse reads.

`r_reppos #I`

specifies the number of positions that may be repaired in restriction sites.

Special considerations are necessary when your restriction enzyme has a degenerate recognition site and when you activate restriction site repair. Example: You used ApeKI (G^CWGC). A sequence starts with barcode ACTA and contains an error at the ambiguous position of the restriction site: ACTACGGC. The program can repair this error, but there is no way to find out if it should replace the G with A or T. In such cases, the program always selects the recognized character that comes

alphabetically first and repairs to: ACTACAGC. This can be problematic when the correct character would have been T. In such cases, restriction site repair creates artificial variation between repaired and error-free sequences. I thus strongly recommend removing the restriction site from all sequences when it contains ambiguities and when restriction site repair is active. Activate `cropia` to remove restriction sites from adapter containing sequences. Activate `f_cl_r` and `r_cl_r` to remove them from reads without adapters.

`f_cl_r #B` and `r_cl_r #B`

clip restriction sites from forward and reverse reads without adapters (recommended, speeds up locus identification with **indloc**).

`f_rint_c #B` and `r_rint_c #B`

activate the search for internal restriction sites in forward and reverse reads. Sequences containing internal restriction sites are discarded. The program discards both reads of a pair when it finds an internal restriction site in one of them.

`f_restr_site_int #S` and `r_restr_site_int #S`

specify internal restriction sites in forward and reverse reads, e.g. `GGATCC` for BamHI (`G^GATCC`).

BARCODE DEMULTIPLEXING

The program can identify barcodes at the beginning of forward reads. In all steps of barcode identification, the program evaluates all barcodes and the following restriction site (`f_restr_site1`). That means, the first characters of the sequence must match the barcode and the following characters must match `f_restr_site1`. The barcode remains undetermined when these conditions are met with more than one barcode. The program evaluates all possible permutations of degenerate restriction sites.

`f_bc #B`

This option activates barcode identification with function bcr_c, which searches barcodes and restriction sites without mismatch.

`barcode_listname #S`

Name (or path) of a textfile containing all barcodes, one per line. The file must not contain blank lines. Barcodes may have different lengths and must be written in capital letters.

It happens that bcr_c cannot identify a barcode because of sequencing/polymerase errors in the barcode or restriction site. The next three functions try to identify a barcode with mismatch and repair. They act only on sequences with (still) unknown barcode and do not repair the sequence when they could not unambiguously identify a single barcode:

`f_bc_rrep #B` activates barcode identification with restriction site repair.

`f_bcrep_r #B` activates barcode identification with barcode repair.

`f_bcrep_rrep #B` activates barcode identification with repair of barcode and restriction site.

`bc_reppos #I` specifies the number of positions in the barcode that may be repaired.

`r_reppos #I` (see above) specifies the number of positions in the restriction site that may be repaired.

The appropriate number of positions to be repaired depends on the differences between your barcodes.

`f_cl_bc #I` activates clipping (removal) of the barcode, which is mandatory for genotyping with **indloc**.

`compbcl #B` activates compensation of different barcode lengths. This function needs the next variable:

`lonbc #I` length of the longest barcode (in the whole study, not necessarily in this particular dataset).

Function `compcl` truncates sequences with shorter barcodes to the length of sequences with the longest barcode after barcode removal. With `lonbc 8`, the function removes 4 characters from the end of a sequence with barcode ACTA. This function ensures equal chances to detect adapters in sequences with different barcodes.

ADAPTER SEARCH

The program searches for adapters in this order:

1: Search for the full-length adapter in the whole sequence starting at its beginning.

2: Search for fragments of the adapter that start with the first adapter-position at the end of the sequence.

The program stops searching as soon as it found an adapter. A minimum of 5 positions of the adapter must be present at the end of the sequence in order to be detected. Mismatch can be allowed (`admis`, see below). The program searches for one adapter sequence (common adapter) in forward reads. When it does not detect an adapter, it searches in reverse reads. In reverse reads, it can search either for a single adapter sequence or for one of several barcode adapter sequences. For the latter option, the program must have determined the barcode of the forward read. It then searches the corresponding barcode adapter in the reverse read.

When the adapter sequence contains an ambiguity, the program evaluates all possible variants of the sequence. Theoretically, different adapter variants could match at different positions in the sequence. The program always selects the leftmost matching variant.

`admis #D`

specifies the fraction of acceptable mismatch positions in adapter sequences. The number of positions is rounded down: A fraction of 10% (`admis 0.1`) with an adapter length of 28 evaluates to a maximum of 2 mismatch positions in the full-length adapter. One mismatch position is accepted in fragments of 10-19 positions and no mismatch in shorter fragments.

`f_ad_c #B` activates adapter search in forward reads.

`f_adseq #S` specifies the sequence of the (common) adapter in forward reads.

`r_sad_c #B` activates search of a single adapter sequence in reverse reads.

`r_sadseq #S` specifies the sequence of a single adapter in reverse reads.

`r_bcad_c #B` activates search of barcode adapters in reverse reads.

`bcad_listname #S` specifies the name (or path) of a text file that contains all barcodes and the corresponding barcode adapters: Each line contains a barcode and the corresponding barcode adapter separated by any number of whitespace characters, both in capital letters. The barcode is written as it appears in the forward read. The barcode adapter is written as it appears in the reverse read. IUPAC ambiguity symbols are allowed in the adapter sequence.

CONSTRUCTION OF RC-DUPLICATES FROM PAIRED READS WITHOUT ADAPTER

`catsrc #B`

activates the construction of *rc-duplicates* from paired reads without adapter. The function works when both reads have passed all filters. When the reads have different length (e.g. because of barcode removal) it truncates the longer read to the length of the shorter one. It appends the reverse complement of the reverse read to the forward read. Next, it replaces the reverse read with the reverse complement of the new, elongated forward read. You may remove restriction sites (`f_cl_r`, `r_cl_r`, see above) before `catsrc` but be sure to remove them from both reads.

CONSTRUCTION OF RC-DUPLICATES FROM PAIRED READS WITH ADAPTER

`cropia #B`

This function crops the insert of adapter containing sequences: It removes the restriction site at the begin of the sequence (`f_restr_site1` or `r_restr_site1`, barcode is already removed). It removes the adapter (and any sequence thereafter). The function first calculates the length of the cropped insert. It does not crop and discards the sequence when it would be shorter than a configurable minimum (`ilamin`). You must remove the adapter together with all characters that are part of the restriction site from the right sequence end for the proper construction of *rc-duplicates*. For this reason, you must specify an offset for adapter removal:

`f_ao #I` and `r_ao #I` specify the offsets for adapter removal in forward and reverse reads. With an offset of -1, `cropia` removes the adapter and one position left of the adapter. With an offset of 1, it removes the adapter except for its first position.

Example of an adapter containing sequence: The sequence starts with barcode ACTA. Restriction sites (BamHI) are written in bold. The insert is shaded in gray. The (common) adapter is underlined.

ACTA**GATCC**TTCA...AGCTACT**GGATC**AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

Example of an adapter dimer: The adapter sequence follows immediately after the barcode ACTA.

ACTA**GATC**AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

You could search for an adapter sequence starting with **GGATC**`AG....` and use an offset of zero (`f_ao 0`) to remove the adapter and all parts of the restriction site. The problem with these settings is that the program cannot detect adapter dimers. (The barcode is already removed when the program searches for adapters.) The solution is to search for an adapter starting with **GATC**`AG....` and to use and offset of -1 for adapter removal. Adapter dimers are then detected and discarded by cropia.

`ilamin #I` specifies the minimum length of the cropped insert. For an analysis with **indloc**, I recommend values of 20 or greater.

`rcdupa #B`

This variable activates the construction of *rc-duplicates* from adapter containing sequences. You must activate `cropia` and specify meaningful values for `f_ao`, `r_ao` and `ilamin` to obtain useful *rc-duplicates*. When the adapter has been detected in the forward read, this function replaces the reverse read with the reverse complement of the (cropped) forward read. In fact, the reverse read is not analyzed at all in this case. When the adapter was detected in the reverse read, `rcdupa` replaces the forward read with the reverse complement of the (cropped) reverse read.

Outfiles

The program creates outfiles in a folder that is named according to the settings-file. It always produces the following files:

report.txt — summary counts of analysis results.

used_settings.txt — A printout of all settings you have made. The file can be used as settingsfile again.

If you activated the saving of fastq-outfiles (`save_f 1` and/or `save_r 1`), it produces:

fastq_out.txt — names of all fastq-outfiles that have been produced.

Finally, there will be several or many fastq-outfiles for different barcodes, discarded reads, reads with adapters and other categories, depending on your settings:

NN_*: Discarded reads.

SAMP_* good reads with unknown barcode: When barcode demultiplexing is active, such files exist for reverse reads only. They contain good reverse reads that could not be assigned to a barcode because the barcode could not be determined in the forward read. When barcode demultiplexing is inactive, all filenames except “NN_*” start with “SAMP_*”.

ACTA_* reads assigned to barcode ACTA.

*_ad_* reads containing adapter (with setting `adsf 1`).

When paired reads are analyzed and saved to the same outfile (`r1r2sf 0`), complete read pairs are contained in files such as:

ACTA_R12.fq
ACTA_ad_R12.fq
SAMP_R12.fq
SAMP_ad_R12.fq

When one of the two reads in a pair is discarded, the retained read is written to a file with a name containing “_sg_”:

SAMP_sg_*
SAMP_ad_sg_*
ACTA_sg_*
ACTA_ad_sg_*.

When paired reads are analyzed and saved to separate outfiles (`r1r2sf 1`), sorted pairs of sequences are contained in pairs of files such as:

ACTA_R1.fq	ACTA_R2.fq
ACTA_ad_R1.fq	ACTA_ad_R2.fq
SAMP_R1.fq	SAMP_R2.fq
SAMP_ad_R1.fq	SAMP_ad_R2.fq

*_R1.fq single reads or forward reads from paired reads.

*_R2.fq reverse reads from paired reads.