Package 'fraq'

April 14, 2026

Type Package

Title A High-Throughput and Extensible Toolkit for Processing FASTQ Data

Version 0.99.2

Date 2026-02-12

Description High-throughput extensible toolkit for processing FASTQ data. The goal of this package is to empower users to quickly build out small programmatic 'kernels' to define any FASTQ processing task they may need. Builds on Intel TBB's flow graph to orchestrate concurrent I/O and data processing; throughput can be as fast as compression and disk speed allows. The package also ships with a suite of predefined kernels for common FASTQ tasks.

License GPL-3

biocViews Software, Infrastructure, Sequencing, DNASeq, QualityControl, Alignment

URL https://github.com/traversc/fraq

BugReports https://github.com/traversc/fraq/issues

Depends R (>= 4.5.0)

Imports Rcpp, ShortRead, Biostrings, RcppParallel, edlibR, stringfish

LinkingTo Rcpp, RcppParallel, edlibR

Suggests knitr, rmarkdown, BiocStyle, processx

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

SystemRequirements GNU make

Copyright This package includes code from the 'zstd' library created by Yann Collet and owned by Facebook, Inc.

git_url https://git.bioconductor.org/packages/fraq

git_branch devel

git_last_commit d2f1ab3

git_last_commit_date 2026-02-12

Repository Bioconductor 3.23

Date/Publication 2026-04-14

Author Travers Ching [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-5577-3516>),
Yann Collet [ctb, cph] (Author of the bundled zstd library),
Facebook, Inc. [cph] (Copyright holder of the bundled zstd code),
Reichardt Tino [ctb, cph] (Contributor/copyright holder of bundled zstd code),
Skibinski Przemyslaw [ctb, cph] (Contributor/copyright holder of bundled zstd code),
Mori Yuta [ctb, cph] (Contributor/copyright holder of bundled zstd code)

Maintainer Travers Ching <traversc@gmail.com>

Contents

`fraq-package` *fraq: fastq processing with TBB flow graphs*

Description

fraq is a high-throughput toolkit for FASTQ processing. It uses a TBB flow graph to coordinate concurrent I/O and compute, provides prebuilt kernels for common tasks, and lets you extend it by writing custom kernels.

Details

See individual function help pages for details on each kernel. The README contains a short walkthrough with synthetic data.

Author(s)

Maintainer: Travers Ching <traversc@gmail.com> (ORCID) [copyright holder]

Other contributors:

- Yann Collet (Author of the bundled zstd library) [contributor, copyright holder]
- Facebook, Inc. (Copyright holder of the bundled zstd code) [copyright holder]
- Reichardt Tino (Contributor/copyright holder of bundled zstd code) [contributor, copyright holder]
- Skibinski Przemyslaw (Contributor/copyright holder of bundled zstd code) [contributor, copyright holder]
- Mori Yuta (Contributor/copyright holder of bundled zstd code) [contributor, copyright holder]

See Also

Useful links:

- https://github.com/traversc/fraq
- Report bugs at https://github.com/traversc/fraq/issues

`fraq_align` *Align a query to a target*

Description

Calculate distances between query sequences and a target under a chosen boundary model using Levenshtein or Hamming distance.

Usage

```
fraq_align(
  query,
  target,
  max_distance = 2147483647L,
  ambiguity_base = "",
  boundary = "contains",
  distance_metric = "lv"
)
```

Arguments

query	Character vector or Biostrings XString/XStringSet of query sequences.
target	Character vector or Biostrings XString/XStringSet of target sequences.
max_distance	Integer maximum allowed distance; defaults to .Machine$integer.max.
ambiguity_base	Single character to treat as ambiguity when matching, or empty string "" to disable; must be length 0 or 1.
boundary	One of "contains", "global", or "starts".
distance_metric	One of "lv" (Levenshtein) or "hm" (Hamming). "hm" requires query and target to be the same length.

Value

A data frame with the Biostrings inputs as the first two columns followed by the alignment metadata.

Examples

```
fraq_align("ACGTNT", "ACGTAT", max_distance = 2L, ambiguity_base = "N",
           boundary = "contains", distance_metric = "lv")
fraq_align(Biostrings::DNAString("ACGT"), Biostrings::DNAString("ACGA"),
           max_distance = 1L, boundary = "global", distance_metric = "hm")
```

---

fraq_chunk *Chunk sequencing files into fixed-size batches*

Description

Split input datasets into sequential chunks. Each chunk is written using output_prefix suffixed with _chunk{N} and the format indicated by output_suffix.

Usage

```
fraq_chunk(input, output_prefix, output_suffix, chunk_size, nthreads = 1L)
```

Arguments

input	Character vector of source files/keys.
output_prefix	Character vector of prefixes used when naming chunked outputs. Must be the same length as input.
output_suffix	Character scalar describing the output format; use "fastq", "fraq", "mem", "gz", "zst".
chunk_size	Numeric chunk size in reads; each output chunk contains up to this many records (per input stream).
nthreads	Integer number of worker threads.

Details

The suffix mapping follows:

- "fastq" -> .fastq
- "gz" -> .fastq.gz
- "zst" -> .fastq.zst
- "fraq" -> .fraq
- "mem" -> .mem

Value

Invisibly returns NULL after writing all chunked outputs.

Examples

```
r1 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 25, read_length = 75)
fraq_chunk(r1,
           output_prefix = tempfile("chunked_R1"),
           output_suffix = "fastq",
           chunk_size = 10,
           nthreads = 1L)
```

---

fraq_concat *Concatenate sequencing files*

Description

Concatenate one or more FRAQ/FASTQ inputs (plain, .gz, .zst, .fraq, or .mem) into a single output file.

Usage

```
fraq_concat(input, output, nthreads = 1L)
```

Arguments

input	Character vector of input paths/keys to concatenate. Mixed formats are supported.
output	Character scalar giving the destination path (or .mem key).
nthreads	Integer number of threads for reading, compression, and writing.

Value

Invisibly returns NULL after writing the concatenated output.

Examples

```
tmp_dir <- tempdir()
inputs <- file.path(tmp_dir, sprintf("reads_%d.fastq", 1:2))
lapply(inputs, generate_random_fastq, n_reads = 10, read_length = 50)
out <- file.path(tmp_dir, "all_reads.fastq.gz")
fraq_concat(inputs, out, nthreads = 1L)
```

---

fraq_convert *Convert sequencing files between supported formats*

Description

Re-encode sequencing files in any supported FRAQ/FASTQ format. Input and output vectors must be the same length.

Usage

```
fraq_convert(input, output, nthreads = 1L)
```

Arguments

input	Character vector of source files/keys.
output	Character vector of destination files/keys, same length as input.
nthreads	Integer number of threads for reading/writing.

Details

FIFO pipes (paths ending with .fifo) are only available on Unix-like systems; on Windows they are not supported and will trigger an error.

Value

Invisibly returns NULL after writing the converted outputs.

Examples

```
src <- tempfile(fileext = ".fastq")
generate_random_fastq(src, n_reads = 10, read_length = 50)
dest <- tempfile(fileext = ".fastq.gz")
fraq_convert(src, dest, nthreads = 1L)
```

---

fraq_count_barcodes *Count barcodes in FASTQ file(s)*

Description

Count occurrences of provided short barcodes in reads, allowing up to `max_distance` mismatches. Accepts one or more FASTQ files (e.g., R1/R2).

Usage

```
fraq_count_barcodes(
  input,
  barcodes,
  max_distance = 1L,
  allow_revcomp = FALSE,
  nthreads = 1L
)
```

Arguments

`input`	Character vector of one or more input FASTQ file paths (e.g., R1 and R2).
`barcodes`	Character vector of barcode sequences to count.
`max_distance`	Integer maximum number of mismatches allowed for a match.
`allow_revcomp`	Logical; if TRUE, also match reverse complements.
`nthreads`	Integer number of threads.

Value

A data frame with counts per barcode.

Examples

```
r1 <- tempfile(fileext = ".fastq")
r2 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
generate_random_fastq(r2, n_reads = 1000, read_length = 100,
    name_prefix = "read_R2_")
short_barcodes <- c("ACGT", "TGCA", "GTAC")
counts <- fraq_count_barcodes(c(r1, r2), short_barcodes, max_distance = 1L,
    allow_revcomp = FALSE, nthreads = 1L)
counts
```

fraq_demux *Demultiplex FASTQ file(s) by barcode prefix*

Description

Write barcode-specific outputs by matching a prefix on the first read of each record. Each output path is formed by substituting {barcode} in the supplied format string.

Usage

```
fraq_demux(input, output_format, barcodes, max_distance = 1L, nthreads = 1L)
```

Arguments

input	Character vector of one or more input FASTQ file paths (e.g., R1 and R2).
output_format	Character vector of format strings, same length as input. Each string must contain the literal {barcode} placeholder.
barcodes	Character vector of barcode sequences to test as prefixes.
max_distance	Integer maximum Hamming distance allowed between barcode and read prefix.
nthreads	Integer number of threads.

Details

When no barcode matches, the literal NO_MATCH is substituted in place of {barcode}. If multiple barcodes match the same read, MULTI_MATCH is used.

Value

Invisibly, NULL. Files are written to disk according to output_format.

Examples

```
r1 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
out <- tempfile("R1_", fileext = "_{barcode}.fastq")
barcodes <- c("ACGT", "TGCA", "GTAC")
fraq_demux(r1, out, barcodes, max_distance = 1L, nthreads = 1L)
```

`fraq_downsample` *Downsample FASTQ file(s)*

Description

Write deterministically downsampled FASTQ file(s) to disk. `input` and `output` must be vectors of the same length (e.g., R1/R2 pairs).

Usage

```
fraq_downsample(input, output, amount, nthreads = 1L)
```

Arguments

`input`	Character vector of one or more input FASTQ file paths. Vectors must be the same length as `output` (e.g., R1 and R2 pairs).
`output`	Character vector of output FASTQ file paths, same length as `input`.
`amount`	Numeric scalar in (0, 1); proportion of reads to retain.
`nthreads`	Integer number of threads.

Details

Downsampling is deterministic: given the same inputs, `fraq_downsample()` keeps the same records every run while matching the requested proportion as closely as possible.

Value

Invisibly returns `NULL` after writing the downsampled outputs.

Examples

```
r1 <- tempfile(fileext = ".fastq")
r2 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
generate_random_fastq(r2, n_reads = 1000, read_length = 100,
    name_prefix = "read_R2_")
out <- c(tempfile(fileext = ".fastq"), tempfile(fileext = ".fastq"))
fraq_downsample(c(r1, r2), out, amount = 0.1)
```

fraq_fifo_supported *Detect FRAQ FIFO support*

Description

Report whether the current build of **fraq** was compiled with named pipe (FIFO) support. FIFO outputs (paths ending in `.fifo`) are only available on Unix-like platforms where the build detected `S_IFIFO`.

Usage

```
fraq_fifo_supported()
```

Details

The result is determined at compile time; reinstalling the package on an operating system that exposes FIFOs is required to enable support.

Value

Logical scalar indicating whether FIFO inputs/outputs are supported.

Examples

```
if (fraq_fifo_supported()) {
    message("FIFO streams are available on this platform.")
} else {
    message("Use regular files instead of .fifo paths on this build.")
}
```

fraq_mem_list *Manage in-memory FASTQ datasets*

Description

`fraq_mem_list()` summarizes the `.mem` datasets currently stored in the session. `fraq_mem_remove()` deletes one or more `.mem` entries, freeing the associated memory. `fraq_mem_load()` is a convenience wrapper around `fraq_convert()` that loads on-disk FASTQ/FRAQ inputs into the in-memory store after validating that the outputs end with `.mem`.

The in-memory store lives in the current R session. For consistent results, call the helper functions when no other `fraq` jobs are actively writing to the same `.mem` keys.

Usage

```
fraq_mem_list()

fraq_mem_remove(mem_key)

fraq_mem_load(input, mem_key, nthreads = 1L)
```

Arguments

mem_key	Character vector of .mem keys to remove.
input	Character vector of FASTQ/FRAQ paths to load into memory.
nthreads	Positive integer parallelism for the load.

Value

- fraq_mem_list() returns a data frame with columns mem_key and n_reads.
- fraq_mem_remove() returns a logical vector indicating which keys were removed.
- fraq_mem_load() returns the target .mem keys invisibly.

Examples

```
tmp <- tempfile(fileext = ".fastq")
generate_random_fastq(tmp, n_reads = 100, read_length = 75)
mem_path <- tempfile(fileext = ".mem")
fraq_mem_load(tmp, mem_path)
fraq_mem_list()
fraq_mem_remove(mem_path)
```

---

fraq_merge_pairs *Merge paired-end reads into a consensus*

Description

Merge R1/R2 pairs by overlapping sequences (optionally reverse-complementing R2), emitting merged reads and optional unmerged outputs.

Usage

```
fraq_merge_pairs(
  input,
  output_merged,
  output_unmerged = NULL,
  min_overlap = 12L,
  max_mismatch_rate = 0.1,
  consensus_mode = c("max", "mean", "r1", "r2"),
  trim_overhang = TRUE,
```

```
  revcomp_R2 = TRUE,
  nthreads = 1L
)
```

Arguments

input	Character vector of length 2 with input FASTQ paths (R1, R2).
output_merged	Character scalar path/key receiving merged single-end reads.
output_unmerged	Optional character vector of length 2 for unmerged R1/R2 outputs. Use NULL to drop unmerged pairs.
min_overlap	Integer minimum overlap required to attempt merging.
max_mismatch_rate	Numeric maximum mismatch fraction allowed within the overlap.
consensus_mode	Character string controlling consensus base selection: "max", "mean", "r1", or "r2".
trim_overhang	Logical; if TRUE, include non-overlapping tails when constructing the merged read.
revcomp_R2	Logical; if TRUE, reverse-complement R2 before merging.
nthreads	Integer number of worker threads.

Details

Qualities are interpreted as PHRED+33.

Value

A list summarising merge statistics (merged_reads, unmerged_reads, mean_insert_size, etc.).

Examples

```
r1 <- tempfile(fileext = ".fastq")
r2 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 100, read_length = 100,
    name_prefix = "read_R1_")
generate_random_fastq(r2, n_reads = 100, read_length = 100,
    name_prefix = "read_R2_")
out_merged <- tempfile(fileext = ".fastq")
fraq_merge_pairs(c(r1, r2), out_merged, output_unmerged = NULL,
    min_overlap = 20L, max_mismatch_rate = 0.05)
```

fraq_options *Get or set FRAQ options*

Description

Get or set FRAQ options.

Usage

```
fraq_options(option, value = NULL)
```

Arguments

option	Character string name of the option. Valid options are: "blocksize", "fraq_compress_level", "zstd_compress_level", "gzip_compress_level".
value	Optional value to set the option to; if NULL, the current value is returned.

Value

The current option value (if input value is NULL) or previous option value.

Examples

```
# Get current blocksize
fraq_options("blocksize")
# # Set blocksize to 16384
fraq_options("blocksize", 16384)
```

fraq_quality_filter *Filter reads by whole-read quality*

Description

Drop read sets when any mate fails the quality thresholds. Qualities are interpreted as PHRED+33.

Usage

```
fraq_quality_filter(
  input,
  output,
  min_mean_quality = 20,
  max_low_q_bases = .Machine$integer.max,
  low_q_threshold = 20L,
  nthreads = 1L
)
```

Arguments

input	Character vector of one or more input FASTQ file paths. Must be the same length as output.
output	Character vector of output FASTQ paths.
min_mean_quality	Numeric minimum mean base quality required for each mate.
max_low_q_bases	Integer maximum number of bases below low_q_threshold allowed per mate.
low_q_threshold	Integer PHRED cutoff used to count low-quality bases.
nthreads	Integer number of worker threads.

Details

Both thresholds are evaluated separately on every mate. If any mate fails, the entire read set is discarded.

Value

Invisibly, NULL. Reads are written to output paths for records that pass the filters.

Examples

```
r1 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
out <- tempfile(fileext = ".fastq")
fraq_quality_filter(r1, out, min_mean_quality = 25, max_low_q_bases = 2L,
    low_q_threshold = 20L, nthreads = 1L)
```

---

fraq_rcpp_template *Generate an example fraq Rcpp script*

Description

Writes a minimal Rcpp example file showing how to write custom kernels via Rcpp.

Usage

```
fraq_rcpp_template(output_file)
```

Arguments

output_file	Character path where the C++ source file will be written.

Value

NULL invisibly.

Examples

```
cpp <- tempfile(fileext = ".cpp")
fraq_rcpp_template(cpp)
# Rcpp::sourceCpp(cpp)  # optionally compile the example
```

---

fraq_shortread *Bridge FRAQ formats with ShortReadQ*

Description

fraq_export_shortreadq() converts FRAQ/FASTQ inputs into in-memory ShortReadQ objects via fraq_convert. fraq_import_shortreadq converts ShortReadQ objects to any supported FRAQ format.

Usage

```
fraq_export_shortreadq(input, nthreads = 1L, tmpdir = tempdir())

fraq_import_shortreadq(shortreadq, output, nthreads = 1L, tmpdir = tempdir())
```

Arguments

input	Character vector of input paths/keys accepted by fraq_convert().
nthreads	Positive integer passed to fraq_convert.
tmpdir	Directory used for staging temporary files.
shortreadq	A ShortReadQ or list of ShortReadQ objects to be written via FRAQ encoders.
output	Character vector of destination paths/keys, same length as shortreadq.

Value

- fraq_export_shortreadq() returns a single ShortReadQ when input has length 1, otherwise a list of ShortReadQ objects.
- fraq_import_shortreadq() invisibly returns the normalized output vector after conversion.

See Also

ShortRead::readFastq(), ShortRead::writeFastq(), fraq_convert()

Examples

```
fq <- tempfile(fileext = ".fastq")
generate_random_fastq(fq, n_reads = 10, read_length = 50)
fraq_path <- tempfile(fileext = ".fraq")
fraq_convert(fq, fraq_path)

reads <- fraq_export_shortreadq(fraq_path)
roundtrip_fastq <- tempfile(fileext = ".fastq")
fraq_import_shortreadq(reads, roundtrip_fastq)
stopifnot(file.exists(roundtrip_fastq))
```

---

fraq_slice *Slice reads by index or limit*

Description

Write a subset of reads from `input` to `output`, either the first `limit` reads or specific zero-based indices in `select`.

Usage

```
fraq_slice(input, output, limit = NULL, select = NULL, nthreads = 1L)
```

Arguments

`input`	Character vector of source files/keys.
`output`	Character vector of destination files/keys, same length as `input`.
`limit`	Numeric scalar; keep the first `limit` reads (per record index). Defaults to `NULL`.
`select`	Numeric vector of zero-based indices to keep. Defaults to `NULL`.
`nthreads`	Integer number of threads for reading/writing.

Details

Exactly one of `limit` or `select` must be supplied.

Value

Invisibly returns `NULL` after writing the selected reads.

Examples

```
src <- tempfile(fileext = ".fastq")
generate_random_fastq(src, n_reads = 10, read_length = 50)
dest <- tempfile(fileext = ".fastq")
fraq_slice(src, dest, limit = 5)
```

`fraq_summary` *Summarize FASTQ quality metrics (single- or paired-end)*

Description

Compute QC summaries for single- or paired-end FASTQ files. When two inputs are provided, R1 and R2 are summarized separately and an insert-size histogram is reported (estimated from R1 vs reverse-complemented R2 overlap).

Usage

```
fraq_summary(
  input,
  phred33 = TRUE,
  min_overlap = 12L,
  max_mismatch_rate = 0.1,
  limit = 0L,
  nthreads = 1L
)
```

Arguments

`input`	Character vector of length 1 or 2 with input FASTQ paths. Length 1 = single-end; length 2 = paired-end (first element maps to R1, second to R2).
`phred33`	Logical; TRUE if qualities are PHRED+33, FALSE for PHRED+64.
`min_overlap`	Integer minimum overlap used for insert-size estimation (paired-end only).
`max_mismatch_rate`	Numeric between 0 and 1 (inclusive); maximum allowed mismatch rate within the overlapped region (paired-end only).
`limit`	Numeric cap on the number of read sets to process. Use `0` to process all available reads.
`nthreads`	Integer number of threads.

Details

Outputs per-mate tables:

- `basic_stats_R{1,2}`: total sequences, total bases, min/mean/max length, GC percent.
- `per_base_quality_R{1,2}`: mean PHRED by 1-based position (with counts).
- `per_base_content_R{1,2}`: long format base usage by position (A/C/G/T/N/other).
- `length_distribution_R{1,2}`: histogram of sequence lengths.
- `avg_read_quality_R{1,2}`: histogram of rounded per-read average quality (columns `avg_quality`, `count`).

For paired-end inputs, `insert_size` is included when overlaps are found.

Value

A named list of data frames. For single-end: R1-only tables. For paired-end: R1/R2 tables plus optional `insert_size`. Each mate includes `basic_stats`, per-base quality/content, length distributions, and average read quality histograms.

Examples

```
# Single-end example
r1 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
res_se <- fraq_summary(r1, nthreads = 1L)
res_se$basic_stats_R1

# Paired-end example
r1 <- tempfile(fileext = ".fastq"); r2 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
generate_random_fastq(r2, n_reads = 1000, read_length = 100,
    name_prefix = "read_R2_")
res_pe <- fraq_summary(c(r1, r2), nthreads = 1L)
res_pe$basic_stats_R1
# dplyr example using pipes
# library(dplyr)
# res_pe$insert_size %>% arrange(desc(count)) %>% head()
```

`fraq_trim_adapters` *Trim adapters from FASTQ file(s)*

Description

Trim occurrences of adapter sequence(s) at the start of the first fastq input. `input` and `output` must be vectors of the same length (e.g., R1/R2 pairs). Adapters will be trimmed only for the first fastq, but all inputs will be filtered if `filter_untrimmed` is `TRUE`.

Usage

```
fraq_trim_adapters(
  input,
  output,
  adapters,
  max_distance = 1L,
  filter_untrimmed = TRUE,
  nthreads = 1L
)
```

Arguments

input	Character vector of one or more input FASTQ file paths. Vectors must be the same length as output (e.g., R1 and R2 pairs).
output	Character vector of output FASTQ file paths, same length as input.
adapters	Character vector of adapter sequences to trim. Adapters are given priority based on the order they appear.
max_distance	Integer maximum number of mismatches for adapter matching.
filter_untrimmed	Logical; if TRUE, drop reads with no trim.
nthreads	Integer number of threads.

Value

A data frame of counts of trimmed adapters.

Examples

```
r1 <- tempfile(fileext = ".fastq")
r2 <- tempfile(fileext = ".fastq")
generate_random_fastq(r1, n_reads = 1000, read_length = 100,
    name_prefix = "read_R1_")
generate_random_fastq(r2, n_reads = 1000, read_length = 100,
    name_prefix = "read_R2_")
out <- c(tempfile(fileext = ".fastq"), tempfile(fileext = ".fastq"))
adapters <- c("ACGT", "TGCA", "GTAC")
fraq_trim_adapters(c(r1, r2), out, adapters, max_distance = 1L,
    filter_untrimmed = TRUE, nthreads = 1L)
```

---

generate_random_fastq *Generate a random FASTQ file (optionally gzipped)*

Description

Creates a synthetic FASTQ file with random DNA sequences and Illumina-like Phred+33 quality strings (high early-cycle quality with a gentle tail drop). If output_file ends with .gz, the file is written gzip-compressed via a connection.

Usage

```
generate_random_fastq(
  output_file,
  n_reads = 1e+05,
  read_length = 100,
  name_prefix = "read_"
)
```

Arguments

output_file	Character vector of length 1 (single-end) or 2 (paired-end) outputs. .gz suffixes create gzip-compressed files; otherwise plain-text FASTQ is written.
n_reads	Integer number of reads to generate. Default 1e5.
read_length	Integer read length (number of bases per read). Default 100.
name_prefix	Character prefix for read names. Default "read_".

Details

Each read comprises four lines: header, sequence, +, and quality. Headers are generated as @<name_prefix><index>. Sequences are sampled uniformly from ACGT. Qualities follow a tapered profile that starts near Q37 and falls toward the low 30s, with occasional low-quality spikes to mimic typical Illumina output.

Value

Invisibly returns the path(s) written in output_file.

Examples

```
# Example: small test files
tmp_fastq <- tempfile(fileext = ".fastq")
tmp_fastq_gz <- tempfile(fileext = ".fastq.gz")

# Create plain FASTQ (500 reads, length 100)
generate_random_fastq(tmp_fastq, n_reads = 500, read_length = 100)

# Create gzipped FASTQ (500 reads, length 100)
generate_random_fastq(tmp_fastq_gz, n_reads = 500, read_length = 100)

# Paired-end example with overlapping mates
generate_random_fastq(c(tmp_fastq, tmp_fastq_gz),
                      n_reads = 100,
                      read_length = 150)
```

Index