Package ‘MetaScope’

July 3, 2025

Type Package

Title Tools and functions for preprocessing 16S and metagenomic sequencing microbiome data

Version 1.8.1

Description This package contains tools and methods for preprocessing microbiome data. Functionality includes library generation, demultiplexing, alignment, and microbe identification. It is in part an R translation of the PathoScope 2.0 pipeline.

License GPL (>= 3)

URL `https://github.com/wejlab/metascope`
`https://wejlab.github.io/metascope-docs/`

BugReports `https://github.com/wejlab/MetaScope/issues`

Depends R (>= 4.4.0)

Imports BiocFileCache, Biostrings, data.table (>= 1.16.2), dplyr, ggplot2, magrittr, Matrix, MultiAssayExperiment, Rbowtie2, readr, rlang, Rsamtools, S4Vectors, stringr, SummarizedExperiment, taxonomizr, tibble, tidyr, tools

Suggests animalcules, BiocStyle, biomformat, GenomicRanges, IRanges, knitr, lintr, plyr, R.utils, rmarkdown, Rsubread, spelling, sys, testthat, usethis

Enhances BiocParallel

VignetteBuilder knitr

BiocType Software

biocViews MicrobiomeData, ReproducibleResearch, SequencingData

Encoding UTF-8

Language en-US

LazyData FALSE

RoxygenNote 7.3.2

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

git_branch RELEASE_3_21

git_last_commit b713f8c

git_last_commit_date 2025-05-04

Repository Bioconductor 3.21

Date/Publication 2025-07-03

Author Aubrey Odom [aut, cre] (ORCID: <https://orcid.org/0000-0001-7113-7598>),
Sean Lu [aut] (ORCID: <https://orcid.org/0009-0007-8005-6125>),
Rahul Varki [aut] (ORCID: <https://orcid.org/0009-0003-5721-9484>),
W. Evan Johnson [aut] (ORCID: <https://orcid.org/0000-0002-6247-6595>)

Maintainer Aubrey Odom <aodom@bu.edu>

Contents

MetaScope-package — *MetaScope: Tools and functions for preprocessing 16S and metagenomic sequencing microbiome data*

Description

This package contains tools and methods for preprocessing microbiome data. Functionality includes library generation, demultiplexing, alignment, and microbe identification. It is in part an R translation of the PathoScope 2.0 pipeline.

Author(s)

Maintainer: Aubrey Odom <`aodom@bu.edu`> (ORCID)

Authors:

- Sean Lu <`sl1729@njms.rutgers.edu`> (ORCID)
- Rahul Varki <`rvarki@bu.edu`> (ORCID)
- W. Evan Johnson <`wej@bu.edu`> (ORCID)

See Also

Useful links:

- `https://github.com/wejlab/metascopehttps://wejlab.github.io/metascope-docs/`
- Report bugs at `https://github.com/wejlab/MetaScope/issues`

align_details — *A universal parameter settings object for Rsubread alignment*

Description

This object is a named vector of multiple options that can be chosen for functions that involve alignment with Rsubread, namely `align_target()` and `filter_host()`. Both functions take an object for the parameter `settings`, which are provided by `align_details` by default, or may be given by a user-created object containing the same information.

Usage

```
data(align_details)
```

Format

list

Details

The default options included in `align_details` are `type = "dna"`, `maxMismatches = 3`, `nsubreads = 10`, `phredOffset = 33`, `unique = FALSE`, and `nBestLocations = 16`. Full descriptions of these parameters can be read by accessing `?Rsubread::align`.

Examples

```
data("align_details")
```

`align_target` *Align microbiome reads to a set of reference libraries*

Description

This is the main MetaScope target library mapping function, using Rsubread and multiple libraries. Aligns to each library separately, filters unmapped reads from each file, and then merges and sorts the .bam files from each library into one output file. If desired, output can be passed to 'filter_host()' to remove reads that also map to filter library genomes.

Usage

```
align_target(
  read1,
  read2 = NULL,
  lib_dir = NULL,
  libs,
  threads = 1,
  align_file = tools::file_path_sans_ext(read1),
  subread_options = align_details,
  quiet = TRUE
)
```

Arguments

`read1`	Path to the .fastq file to align.
`read2`	Optional: Location of the mate pair .fastq file to align.
`lib_dir`	Path to the index files for all libraries.
`libs`	A vector of character strings giving the basenames of the Subread index files for alignment. If ALL indices to be used are located in the current working directory, set `lib_dir = NULL`. Default is `lib_dir = NULL`.
`threads`	The number of threads that can be utilized by the function. Default is 1 thread.
`align_file`	The basename of the output alignment file (without trailing .bam extension).

`subread_options`	A named `list` specifying alignment parameters for the `Rsubread::align()` function, which is called inside `align_target()`. Elements should include type, nthreads, maxMismatches, nsubreads, phredOffset, unique, and nBestLocations. Descriptions of these parameters are available under ?`Rsubread::align`. Defaults to the global `align_details` object.
`quiet`	Turns off most messages. Default is `TRUE`.

Value

This function writes a merged and sorted .bam file after aligning to all reference libraries given, along with a summary report file, to the user's working directory. The function also outputs the new .bam filename.

Examples

```
#### Align example reads to an example reference library using Rsubread

## Create temporary directory
target_ref_temp <- tempfile()
dir.create(target_ref_temp)

tax <- "Ovine atadenovirus D"

## Create temporary taxonomizr accession
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Download genome
all_ref <- MetaScope::download_refseq(tax,
                                      reference = FALSE,
                                      representative = FALSE,
                                      compress = TRUE,
                                      out_dir = target_ref_temp,
                                      caching = TRUE,
                                      accession_path = tmp_accession)

## Create subread index
ind_out <- mk_subread_index(all_ref)

## Get path to example reads
readPath <- system.file("extdata", "reads.fastq",
                        package = "MetaScope")
## Copy the example reads to the temp directory
refPath <- file.path(target_ref_temp, "reads.fastq")
file.copy(from = readPath, to = refPath)

## Modify alignment parameters object
data("align_details")
align_details[["type"]] <- "rna"
align_details[["phredOffset"]] <- 50
# Just to get it to align - toy example!
align_details[["maxMismatches"]] <- 100
```

```
## Run alignment
target_map <- align_target(refPath,
                           libs = stringr::str_replace_all(tax, " ", "_"),
                           lib_dir = target_ref_temp,
                           subread_options = align_details)

## Remove temporary folder
unlink(target_ref_temp, recursive = TRUE)
```

`align_target_bowtie` *Align microbiome reads to set of indexed Bowtie2 libraries*

Description

This is the main MetaScope target library mapping function, using Rbowtie2 and multiple libraries. Aligns to each library separately, filters unmapped reads from each file, and then merges and sorts the .bam files from each library into one output file. If desired, output can be passed to 'filter_host_bowtie()' to remove reads that also map to filter library genomes.

Usage

```
align_target_bowtie(
  read1,
  read2 = NULL,
  lib_dir,
  libs,
  align_dir,
  align_file,
  bowtie2_options = NULL,
  threads = 1,
  overwrite = FALSE,
  quiet = TRUE
)
```

Arguments

`read1`	Path to the .fastq file to align.
`read2`	Optional: Location of the mate pair .fastq file to align.
`lib_dir`	Path to the directory that contains the Bowtie2 indexes.
`libs`	The basename of the Bowtie2 indexes to align against (without trailing .bt2 or .bt2l extensions).
`align_dir`	Path to the directory where the output alignment file should be created.
`align_file`	The basename of the output alignment file (without trailing .bam extension).

`bowtie2_options`	Optional: Additional parameters that can be passed to the align_target_bowtie() function. To see all the available parameters use Rbowtie2::bowtie2_usage(). See Details for default parameters. NOTE: Users should pass all their parameters as one string and if optional parameters are given then the user is responsible for entering all the parameters to be used by Bowtie2. The only parameter that should NOT be specified here is the number of threads.
`threads`	The number of threads that can be utilized by the function. Default is 1 thread.
`overwrite`	Whether existing files should be overwritten. Default is FALSE.
`quiet`	Turns off most messages. Default is `TRUE`.

Details

The default parameters are the same that PathoScope 2.0 uses. "–very-sensitive-local -k 100 –score-min L,20,1.0"

If you experience any issues with reading the input files, make sure that the file(s) are not located in a read-only folder. This can be circumvented by copying files to a new location before running the function.

Value

Returns the path to where the output alignment file is stored.

Examples

```
#### Align example reads to an example reference library using Rbowtie2

## Create temporary directory to store file
target_ref_temp <- tempfile()
dir.create(target_ref_temp)

tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Dowload reference genome
MetaScope::download_refseq("Morbillivirus hominis",
                           reference = FALSE,
                           representative = FALSE,
                           compress = TRUE,
                           out_dir = target_ref_temp,
                           caching = TRUE,
                           accession_path = tmp_accession
)

## Create temporary directory to store the indices
index_temp <- tempfile()
dir.create(index_temp)

## Create bowtie2 index
MetaScope::mk_bowtie_index(
  ref_dir = target_ref_temp,
  lib_dir = index_temp,
  lib_name = "target",
  overwrite = TRUE
)

## Create temporary directory for final file
output_temp <- tempfile()
dir.create(output_temp)

## Get path to example reads
readPath <- system.file("extdata", "virus_example.fastq",
                        package = "MetaScope")

## Align to target genomes
target_map <-
  MetaScope::align_target_bowtie(
    read1 = readPath,
    lib_dir = index_temp,
    libs = "target",
    align_dir = output_temp,
    align_file = "bowtie_target",
    overwrite = TRUE,
    bowtie2_options = "--very-sensitive-local"
  )

## Remove extra folders
unlink(target_ref_temp, recursive = TRUE)
unlink(index_temp, recursive = TRUE)
unlink(output_temp, recursive = TRUE)
```

bam_reheader_R *Replace the header from a .bam file*

Description

This function replaces the header from one .bam file with a header from a different .sam file. This function mimics the function of the 'reheader' function in samtools. It is not intended for use by users.

Usage

```
bam_reheader_R(
  head,
  old_bam,
  new_bam = paste(tools::file_path_sans_ext(old_bam), "h.bam", sep = "")
)
```

Arguments

`head`	A file name and location for the .sam file with the new header.
`old_bam`	A file name and location for the .bam file which you would
`new_bam`	A file name for the new .bam file with a replaced header. Defaults to the same base filename plus 'h.bam'. For example, 'example.bam' will be written as 'exampleh.bam'.

Value

This function will return a new .bam file with a replaced header. The function also outputs the new .bam filename.

`bt2_16S_params` *A universal parameter object for Bowtie 2 16S alignment*

Description

This character string provides several Bowtie 2 options to provide an optimized alignment specifically optimized for 16S amplicon sequencing data. This object can be used with functions that use the Bowtie 2 aligner through the `Rbowtie2` package, namely `align_target_bowtie()` and `filter_host_bowtie`. These settings can be substituted for default settings by passing to the `bowtie2_options` argument.

Usage

```
data(bt2_16S_params)
```

Format

list

Details

The default parameters listed in this object are "–local -R 2 -N 0 -L 25 -i S,1,0.75 -k 5 –score-min L,0,1.88"

Note that k is actually 10 and is doubled internally from 5. The `score-min` function was chosen such that the minimum alignment score allowed requires 98

Further delineation of Bowtie 2 parameters is provided in the Bowtie 2 manual.

Examples

```
data("bt2_16S_params")
```

`bt2_missing_params` *A universal parameter object for Bowtie 2 metagenomic alignment where the host genome is thought to be absent from the reference database*

Description

This character string provides several Bowtie 2 options to conduct an alignment useful for metagenomes, especially in the case where a genome may not be present in the reference database. This object can be used with functions that use the Bowtie 2 aligner through the `Rbowtie2` package, namely `align_target_bowtie()` and `filter_host_bowtie`. These settings can be substituted for default settings by passing to the `bowtie2_options` argument.

Usage

```
data(bt2_missing_params)
```

Format

list

Details

The default parameters listed in this object are "–local -R 2 -N 0 -L 25 -i S,1,0.75 -k 5 –score-min L,0,1.4".

Further delineation of Bowtie 2 parameters is provided in the Bowtie 2 manual.

Examples

```
data("bt2_missing_params")
```

`bt2_regular_params` *A universal parameter object for Bowtie 2 metagenomic or non-16S alignment*

Description

This character string provides several Bowtie 2 options to provide a 95 alignment useful for metagenomes. This object can be used with functions that use the Bowtie 2 aligner through the `Rbowtie2` package, namely `align_target_bowtie()` and `filter_host_bowtie`. These settings can be substituted for default settings by passing to the `bowtie2_options` argument.

Usage

```
data(bt2_regular_params)
```

Format

list

Details

The default parameters listed in this object are "–local -R 2 -N 0 -L 25 -i S,1,0.75 -k 5 –score-min L,0,1.7".

Further delineation of Bowtie 2 parameters is provided in the Bowtie 2 manual.

Examples

```
data("bt2_regular_params")
```

`check_samtools_exists` *Check if samtools exists on the system*

Description

This is an internal function that is not meant to be used outside of the package. It checks whether samtools exists on the system.

Usage

```
check_samtools_exists()
```

Value

Returns TRUE if samtools exists on the system, else FALSE.

`combined_header` *Create a combined .bam header*

Description

This function generates a combined header from multiple .bam files from different reference libraries (e.g. a split bacterial library). It is not intended for use by users.

Usage

```
combined_header(bam_files, header_file = "header_tmp.sam")
```

Arguments

`bam_files`	A character vector of the locations/file names of .bam files from which to combine the headers.
`header_file`	A file name and location for the output file for the combined header. This will be a .sam format file without any reads. Defaults to 'header_tmp.sam'.

Value

This function will return a combined header from all the supplied .bam files.

`convert_animalcules` *Create a multi-assay experiment from MetaScope output for usage with animalcules*

Description

Upon completion of the MetaScope pipeline, users can analyze and visualize abundances in their samples using the animalcules package. This function allows interoperability of `metascope_id` output with both animalcules and QIIME. After running this function, the user should save the returned MAE to an RDS file using a function like `saveRDS` to upload the output into the `animalcules` package.

Usage

```
convert_animalcules(
  meta_counts,
  annot_path,
  which_annot_col,
  end_string = ".metascope_id.csv",
  qiime_biom_out = FALSE,
  path_to_write = ".",
  accession_path = NULL
)
```

Arguments

`meta_counts`	A vector of filepaths to the counts ID CSVs output by `metascope_id()`.
`annot_path`	The filepath to the CSV annotation file for the samples. This CSV metadata/annotation file should contain at least two columns, one with names of all samples WITHOUT the extension listed in `end_string`, e.g. for output file "sample_x76.metascope_id.csv", the column specified in `which_annot_col` should contain the entry "sample_x76". Sample names containing characters "_", "-", and "." are fine, however sample names beginning with numbers should be renamed to have a prefix, e.g. "777897sample" should be renamed to "X777897sample" for both the output file name and the annotation name.

`which_annot_col`	The name of the column of the annotation file containing the sample IDs. These should be the same as the `meta_counts` root filenames.
`end_string`	The end string used at the end of the metascope_id files. Default is ".metascope_id.csv".
`qiime_biom_out`	Would you also like a qiime-compatible biom file output? If yes, two files will be saved: one is a biom file of the counts table, and the other is a specifically formatted mapping file of metadata information. Default is `FALSE`.
`path_to_write`	If `qiime_biom_out = TRUE`, where should output QIIME files be written? Should be a character string of the folder path. Default is '.', i.e. the current working directory.
`accession_path`	(character) Path to taxonomizr accessions. See `taxonomizr::prepareDatabase()`.

Value

Returns a MultiAssay Experiment file of combined sample counts data and/or biom file and mapping file for analysis with QIIME. The MultiAssay Experiment will have a counts assay ("MGX").

Examples

```
tempfolder <- tempfile()
dir.create(tempfolder)

# Create three different samples
samp_names <- c("X123", "X456", "X789")
all_files <- file.path(tempfolder,
                       paste0(samp_names, ".csv"))

create_IDcsv <- function (out_file) {
  final_taxids <- c("273036", "418127", "11234")
  final_genomes <- c(
    "Staphylococcus aureus RF122, complete sequence",
    "Staphylococcus aureus subsp. aureus Mu3, complete sequence",
    "Measles virus, complete genome")
  best_hit <- sample(seq(100, 1050), 3)
  proportion <- best_hit/sum(best_hit) |> round(2)
  EMreads <- best_hit + round(runif(3), 1)
  EMprop <- proportion + 0.003
  dplyr::tibble(TaxonomyID = final_taxids,
                Genome = final_genomes,
                read_count = best_hit, Proportion = proportion,
                EMreads = EMreads, EMProportion = EMprop) |>
    dplyr::arrange(dplyr::desc(.data$read_count)) |>
    utils::write.csv(file = out_file, row.names = FALSE)
  message("Done!")
  return(out_file)
}
out_files <- vapply(all_files, create_IDcsv, FUN.VALUE = character(1))

# Create annotation data for samples
annot_dat <- file.path(tempfolder, "annot.csv")
```

```
dplyr::tibble(Sample = samp_names, RSV = c("pos", "neg", "pos"),
              month = c("March", "July", "Aug"),
              yrsold = c(0.5, 0.6, 0.2)) |>
  utils::write.csv(file = annot_dat,
                   row.names = FALSE)

# Create temporary taxonomizr accession
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

# Convert samples to MAE
outMAE <- convert_animalcules(meta_counts = out_files,
                              annot_path = annot_dat,
                              which_annot_col = "Sample",
                              end_string = ".metascope_id.csv",
                              qiime_biom_out = FALSE,
                              accession_path = tmp_accession)

unlink(tempfolder, recursive = TRUE)
```

---

convert_animalcules_patho

Create a multi-assay experiment from PathoScope 2.0 output for usage with animalcules

---

Description

This function serves as a legacy integration method for usage with PathoScope 2.0 outputs. Upon completion of the PathoScope 2.0 pipeline, users can analyze and visualize abundances in their samples using the animalcules package. After running this function, the user should save the returned MAE to an RDS file using a function like `saveRDS` to upload the output into the `animalcules` package.

Usage

```
convert_animalcules_patho(
  patho_counts,
  annot_path,
  which_annot_col,
  end_string = "-sam-report.tsv"
)
```

Arguments

`patho_counts`	Character string, a directory filepath to the counts ID CSVs output by `metascope_id()`.
`annot_path`	The filepath to the CSV annotation file for the samples.
`which_annot_col`	The name of the column of the annotation file containing the sample IDs. These should be the same as the `meta_counts` root filenames.
`end_string`	The end string used at the end of the metascope_id files. Default is ".metascope_id.csv".

Value

Returns a MultiAssay Experiment file of combined sample counts data. The MultiAssay Experiment will have a counts assay ("MGX").

`convert_animalcules_silva`

Create a multi-assay experiment from MetaScope output for usage with animalcules with the SILVA 13_8 database

---

Description

Upon completion of the MetaScope pipeline, users can analyze and visualize abundances in their samples using the animalcules package. This function allows interoperability of `metascope_id` output with both animalcules and QIIME. After running this function, the user should save the returned MAE to an RDS file using a function like `saveRDS` to upload the output into the `animalcules` package. NOTE: This function is for outputs that were generated with the SILVA 13_8 database.

Usage

```
convert_animalcules_silva(
  meta_counts,
  annot_path,
  which_annot_col,
  end_string = ".metascope_id.csv",
  qiime_biom_out = FALSE,
  path_to_write = ".",
  caching = TRUE
)
```

Arguments

`meta_counts`	A vector of filepaths to the counts ID CSVs output by `metascope_id()` created with the SILVA database.
`annot_path`	The filepath to the CSV annotation file for the samples. This CSV metadata/annotation file should contain at least two columns, one with names of all samples WITHOUT the extension listed in `end_string`, e.g. for output file "sample_x76.metascope_id.csv", the column specified in `which_annot_col` should contain the entry "sample_x76". Sample names containing characters "_", "-", and "." are fine, however sample names beginning with numbers should be renamed to have a prefix, e.g. "777897sample" should be renamed to "X777897sample" for both the output file name and the annotation name.
`which_annot_col`	The name of the column of the annotation file containing the sample IDs. These should be the same as the `meta_counts` root filenames.
`end_string`	The end string used at the end of the metascope_id files. Default is ".metascope_id.csv".

`qiime_biom_out`	Would you also like a qiime-compatible biom file output? If yes, two files will be saved: one is a biom file of the counts table, and the other is a specifically formatted mapping file of metadata information. Default is `FALSE`.
`path_to_write`	If `qiime_biom_out` = `TRUE`, where should output QIIME files be written? Should be a character string of the folder path. Default is '.', i.e. the current working directory.
`caching`	Whether to use BiocFileCache when downloading genomes. Default is `FALSE`.

Value

Returns a MultiAssay Experiment file of combined sample counts data and/or saved biom file and mapping file for analysis with QIIME. The MultiAssayExperiment will have a counts assay ("MGX").

Examples

```
tempfolder <- tempfile()
dir.create(tempfolder)

# Create three different samples
samp_names <- c("X123", "X456", "X789")
all_files <- file.path(tempfolder,
                       paste0(samp_names, ".csv"))

create_IDcsv <- function (out_file) {
  final_taxids <- c("AY846380.1.2583", "AY909584.1.2313", "HG531388.1.1375")
  final_genomes <- rep("Genome name", 3)
  best_hit <- sample(seq(100, 1050), 3)
  proportion <- best_hit/sum(best_hit) |> round(2)
  EMreads <- best_hit + round(runif(3), 1)
  EMprop <- proportion + 0.003
  dplyr::tibble("TaxonomyID" = final_taxids,
                "Genome" = final_genomes,
                "read_count" = best_hit, "Proportion" = proportion,
                "EMreads" = EMreads, "EMProportion" = EMprop) |>
    dplyr::arrange(dplyr::desc(.data$read_count)) |>
    utils::write.csv(file = out_file, row.names = FALSE)
  message("Done!")
  return(out_file)
}
out_files <- vapply(all_files, create_IDcsv, FUN.VALUE = character(1))

# Create annotation data for samples
annot_dat <- file.path(tempfolder, "annot.csv")
dplyr::tibble(Sample = samp_names, RSV = c("pos", "neg", "pos"),
              month = c("March", "July", "Aug"),
              yrsold = c(0.5, 0.6, 0.2)) |>
  utils::write.csv(file = annot_dat,
                   row.names = FALSE)

# Convert samples to MAE
```

```
outMAE <- convert_animalcules_silva(meta_counts = out_files,
                                    annot_path = annot_dat,
                                    which_annot_col = "Sample",
                                    end_string = ".metascope_id.csv",
                                    qiime_biom_out = FALSE,
                                    caching = TRUE)

unlink(tempfolder, recursive = TRUE)
```

---

count_matches — *Count the number of base lengths in a CIGAR string for a given operation*

Description

The 'CIGAR' (Compact Idiosyncratic Gapped Alignment Report) string is how the SAM/BAM format represents spliced alignments. This function will accept a CIGAR string for a single read and a single character indicating the operation to be parsed in the string. An operation is a type of column that appears in the alignment, e.g. a match or gap. The integer following the operator specifies a number of consecutive operations. The `count_matches()` function will identify all occurrences of the operator in the string input, add them, and return an integer number representing the total number of operations for the read that was summarized by the input CIGAR string.

Usage

```
count_matches(x, char = "M")
```

Arguments

`x`	Character. A CIGAR string for a read to be parsed. Examples of possible operators include "M", "D", "I", "S", "H", "=", "P", and "X".
`char`	A single letter representing the operation to total for the given string.

Details

This function is best used on a vector of CIGAR strings using an apply function (see examples).

Value

an integer number representing the total number of alignment operations for the read that was summarized by the input CIGAR string.

Examples

```
# A single cigar string: 3M + 3M + 5M
cigar1 <- "3M1I3M1D5M"
count_matches(cigar1, char = "M")

# Parse with operator "P": 2P
cigar2 <- "4M1I2P9M"
count_matches(cigar2, char = "P")

# Apply to multiple strings: 1I + 1I + 5I
cigar3 <- c("3M1I3M1D5M", "4M1I1P9M", "76M13M5I")
vapply(cigar3, count_matches, char = "I",
       FUN.VALUE = numeric(1))
```

`download_accessions` *Download indexes required for MetaScope ID and MetaBlast modules*

Description

This is a necessary step for all samples utilizing NCBI and SILVA databases in the MetaScope pipeline. As specified by the user, `download_accessions` will automatically download the NCBI accessions database, the SILVA taxonomy database, and or the NCBI Blast 16S database and prepare consolidated databases for downstream use with the MetaID and MetaBLAST modules. This package relies on the `taxonomizr` package.

Usage

```
download_accessions(
  ind_dir,
  tmp_dir = file.path(ind_dir, "tmp"),
  remove_tmp_dir = TRUE,
  NCBI_accessions_database = TRUE,
  NCBI_accessions_name = "accessionTaxa",
  silva_taxonomy_database = TRUE,
  silva_taxonomy_name = "all_silva_headers"
)
```

Arguments

`ind_dir`	Character string. Directory filepath where indices should be saved. Required.
`tmp_dir`	Character path to directory for storing temp files. (Useful to avoid redownloading) Defaults to `file.path(ind_dir, "tmp")`
`remove_tmp_dir`	Delete tmp_dir after downloads are complete? Defaults to `TRUE`
`NCBI_accessions_database`	Logical. Download taxonomizr NCBI accessions database? Defaults to `TRUE`.

`NCBI_accessions_name`	Character string. Filename (with or without extension) to save taxonomizr NCBI accessions database. Defaults to `"accessionTaxa.sql"`.
`silva_taxonomy_database`	Logical. Download SILVA taxonomy database? Defaults to `TRUE`.
`silva_taxonomy_name`	Character string. Filename (with or without extension) to save SILVA taxonomy database. Defaults to the file supplied with the package, `"all_silva_headers.rds"`.

Value

Exports database(s) with names and to location specified by the user.

Examples

```
## Not run:
  download_accessions(
    ind_dir = "C:/Users/JohnSmith/Research",
    tmp_dir = file.path(ind_dir, "tmp"),
    remove_tmp_dir = TRUE,
    NCBI_accessions_database = TRUE,
    NCBI_accessions_name = "accessionTaxa.sql",
    silva_taxonomy_database = TRUE,
    silva_taxonomy_name = "all_silva_headers.rds")

## End(Not run)
```

`download_refseq` *Download RefSeq genome libraries*

Description

This function will automatically download RefSeq genome libraries in a fasta format from the specified taxon. The function will first download the summary report at: `ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/**ki` and then use this file to download the genome(s) and combine them in a single compressed or uncompressed .fasta file.

Usage

```
download_refseq(
  taxon,
  reference = TRUE,
  representative = FALSE,
  compress = TRUE,
  patho_out = FALSE,
  out_dir = NULL,
  caching = FALSE,
  quiet = TRUE,
  accession_path = NULL
)
```

Arguments

`taxon`	Name of single taxon to download. The taxon name should be a recognized NCBI scientific or common name, with no grammatical or capitalization inconsistencies. All available taxonomies are visible by accessing the `MetaScope:::taxonomy_table` object included in the package.
`reference`	Download only RefSeq reference genomes? Defaults to `TRUE`. Automatically set to `TRUE` if `representative = TRUE`.
`representative`	Download RefSeq representative and reference genomes? Defaults to `FALSE`. If `TRUE`, reference is automatically set at `TRUE`.
`compress`	Compress the output .fasta file? Defaults to `TRUE`.
`patho_out`	Create duplicate outpute files compatible with PathoScope? Defaults to `FALSE`.
`out_dir`	Character string giving the name of the directory to which libraries should be output. Defaults to creation of a new temporary directory.
`caching`	Whether to use BiocFileCache when downloading genomes. Default is `FALSE`.
`quiet`	Turns off most messages. Default is `TRUE`.
`accession_path`	(character) Filepath to NCBI accessions SQL database. See `taxonomzr::prepareDatabase()`.

Details

When selecting the `taxon` to be downloaded, if you receive an error saying `Your input is not a valid taxon`, please take a look at the `taxonomy_table` object, which can be accessed with the command `MetaScope:::taxonomy_table)`. Only taxa with exact spelling as they appear at any level of the table will be acknowledged.

Value

Returns a .fasta or .fasta.gz file of the desired RefSeq genomes. This file is named after the kingdom selected and saved to the current directory (e.g. 'bacteria.fasta.gz'). This function also has the option to return a .fasta file formatted for PathoScope as well (e.g. bacteria.pathoscope.fasta.gz') if `path_out = TRUE`.

Examples

```
#### Download RefSeq genomes

## Create temporary taxonomizr accession
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Download all RefSeq reference Bovismacovirus genus genomes
download_refseq('Bovismacovirus', reference = FALSE, representative = FALSE,
                out_dir = NULL, compress = TRUE, patho_out = FALSE,
                caching = TRUE, accession_path = tmp_accession)
```

`extract_reads` *Helper function for demultiplexing*

Description

Helper function for demultiplexing sequencing reads, designed in a way to allow for parallelization across barcodes (parallel extraction of reads by barcode). This function takes a specific barcode (numeric index) from lists of sample names/barcodes, a `Biostrings::DNAStringSet` of barcodes by sequence header, and a `Biostrings::QualityScaledXStringSet` of reads corresponding to the barcodes. Based on the barcode index given, it extracts all reads for the indexed barcode and writes all the reads from that barcode to a separate .fastq file.

Usage

```
extract_reads(
  barcodeIndex,
  barcodes,
  sampleNames,
  index,
  reads,
  location = "./demultiplex_fastq",
  rcBarcodes = TRUE,
  hDist = 0,
  quiet = TRUE
)
```

Arguments

`barcodeIndex`	Which barcode (integer number or index) in the barcodes or sample name to use for read extraction.
`barcodes`	A list of all barcodes in the sequencing dataset. Correlates and in same order as `sampleNames`.
`sampleNames`	A list of sample names or identifiers associated with each barcode in the barcodes list.
`index`	A `Biostrings::DNAStringSet` that contains the read headers and barcode sequence for each header in the sequence slot.
`reads`	A `Biostrings::QualityScaledXStringSet` that has the same headers and order as the index file, but contains the read sequences and their quality scores.
`location`	A directory location to store the demultiplexed read files. Defaults to generate a new subdirectory at './demultiplex_fastq'
`rcBarcodes`	Should the barcode indices in the barcodes list be reverse complemented to match the sequences in the index DNAStringSet? Defaults to `TRUE`.
`hDist`	Uses a Hamming Distance or number of base differences to allow for inexact matches for the barcodes/indexes. Defaults to 0. Warning: if the Hamming Distance is >=1 and this leads to inexact index matches to more than one barcode, that read will be written to more than one demultiplexed read files.

quiet Turns off most messages. Default is TRUE.

Value

Writes a single .fastq file that contains all reads whose index matches the barcode specified. This file will be written to the location directory, and will be named based on the specified sampleName and barcode, e.g. './demultiplex_fastq/SampleName1_GGAATTATCGGT.fastq.gz'

Examples

```
## Create temporary directory
ref_temp <- tempfile()
dir.create(ref_temp)

## Load example barcode, index, and read data into R session
barcodePath <- system.file("extdata", "barcodes.txt", package = "MetaScope")
bcFile <- read.table(barcodePath, sep = "\t", header = TRUE)

indexPath <- system.file("extdata", "virus_example_index.fastq",
package = "MetaScope")
inds <- Biostrings::readDNAStringSet(indexPath, format = "fastq")

readPath <- system.file("extdata", "virus_example.fastq",
                        package = "MetaScope")
reads <- Biostrings::readQualityScaledDNAStringSet(readPath)

## Extract reads from the first barcode
results <- extract_reads(1, bcFile[, 2], bcFile[, 1], inds, reads,
                        rcBarcodes = FALSE, location = ref_temp)

## Extract reads from multiple barcodes
more_results <- lapply(1:6, extract_reads, bcFile[, 2], bcFile[, 1], inds,
                       reads, rcBarcodes = FALSE, location = ref_temp)

## Remove temporary directory
unlink(ref_temp, recursive = TRUE)
```

`filter_host` *Use Rsubread to align reads against one or more filter libraries and subsequently remove mapped reads*

Description

After aligning your sample to a target library with `align_target()`, use `filter_host()` to remove unwelcome host contamination using filter reference libraries. This function takes as input the name of the .bam file produced via `align_target()`, and produces a sorted .bam file with any reads that match the filter libraries removed. This resulting .bam file may be used upstream for further analysis. This function uses Rsubread. For the Rbowtie2 equivalent of this function, see `filter_host_bowtie`.

Usage

```
filter_host(
  reads_bam,
  lib_dir = NULL,
  libs,
  make_bam = FALSE,
  output = paste(tools::file_path_sans_ext(reads_bam), "filtered", sep = "."),
  subread_options = align_details,
  YS = 1e+05,
  threads = 1,
  quiet = TRUE
)
```

Arguments

`reads_bam`	The name of a merged, sorted .bam file that has previously been aligned to a reference library. Likely, the output from running an instance of `align_target()`.
`lib_dir`	Path to the directory that contains the filter Subread index files.
`libs`	The basename of the filter libraries (without index extension).
`make_bam`	Logical, whether to also output a bam file with host reads filtered out. A .csv.gz file will be created instead if `FALSE`. Creating a bam file is costly on resources over creating a compressed csv file with only relevant information, so default is `FALSE`.
`output`	The desired name of the output .bam or .csv.gz file. Extension is automatically defined by whether `make_bam = TRUE`. Default is the basename of `unfiltered_bam` + `.filtered` + extension.
`subread_options`	A named `list` specifying alignment parameters for the `Rsubread::align()` function, which is called inside `align_target()`. Elements should include type, nthreads, maxMismatches, nsubreads, phredOffset, unique, and nBestLocations. Descriptions of these parameters are available under ?`Rsubread::align`. Defaults to the global `align_details` object.
`YS`	yieldSize, an integer. The number of alignments to be read in from the bam file at once for chunked functions. Default is 100000.
`threads`	The amount of threads available for the function. Default is 1 thread.
`quiet`	Turns off most messages. Default is `TRUE`.

Details

A compressed .csv can be created to produce a smaller output file that is created more efficiently and is still compatible with `metascope_id()`.

Value

The name of a filtered, sorted .bam file written to the user's current working directory. Or, if `make_bam = FALSE`, a .csv.gz file containing a data frame of only requisite information to run `metascope_id()`.

Examples

```
#### Filter reads from bam file that align to any of the filter libraries

## Assuming a bam file has been created previously with align_target()

## Create temporary directory
filter_ref_temp <- tempfile()
dir.create(filter_ref_temp)

## Create temporary taxonomizr accession
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")


## Download filter genome
all_species <- c("Staphylococcus aureus subsp. aureus str. Newman")
all_ref <- vapply(all_species, MetaScope::download_refseq,
                  reference = FALSE, representative = FALSE, compress = TRUE,
                  out_dir = filter_ref_temp, caching = FALSE,
                  accession_path = tmp_accession,
                  FUN.VALUE = character(1))
ind_out <- vapply(all_ref, mk_subread_index, FUN.VALUE = character(1))

## Get path to example reads
readPath <- system.file("extdata", "subread_target.bam",
                        package = "MetaScope")

## Copy the example reads to the temp directory
refPath <- file.path(filter_ref_temp, "subread_target.bam")
file.copy(from = readPath, to = refPath)
utils::data("align_details")
align_details[["type"]] <- "rna"
align_details[["phredOffset"]] <- 10
# Just to get it to align - toy example!
align_details[["maxMismatches"]] <- 10

## Align and filter reads
filtered_map <- filter_host(
  refPath, lib_dir = filter_ref_temp,
  libs = stringr::str_replace_all(all_species, " ", "_"),
  threads = 1, subread_options = align_details)

## Remove temporary directory
unlink(filter_ref_temp, recursive = TRUE)
```

`filter_host_bowtie` *Use Rbowtie2 to align reads against one or more filter libraries and subsequently remove mapped reads*

Description

After a sample is aligned to a target library with `align_target_bowtie()`, we may use `filter_host_bowtie()` to remove unwelcome host contamination using filter reference libraries. This function takes as input the name of the .bam file produced via `align_target_bowtie()`, and produces a sorted .bam or .csv.gz file with any reads that match the filter libraries removed. This resulting .bam file may be used downstream for further analysis. This function uses Rbowtie2 For the Rsubread equivalent of this function, see `filter_host`.

Usage

```
filter_host_bowtie(
  reads_bam,
  lib_dir,
  libs,
  make_bam = FALSE,
  output = paste(tools::file_path_sans_ext(reads_bam), "filtered", sep = "."),
  bowtie2_options = NULL,
  YS = 1e+05,
  threads = 1,
  overwrite = FALSE,
  quiet = TRUE
)
```

Arguments

`reads_bam`	The name of a merged, sorted .bam file that has previously been aligned to a reference library. Likely, the output from running an instance of `align_target_bowtie()`.
`lib_dir`	Path to the directory that contains the filter Bowtie2 index files.
`libs`	The basename of the filter libraries (without .bt2 or .bt2l extension).
`make_bam`	Logical, whether to also output a bam file with host reads filtered out. A .csv.gz file will be created instead if `FALSE`. Creating a bam file is costly on resources over creating a compressed csv file with only relevant information, so default is `FALSE`.
`output`	The desired name of the output .bam or .csv.gz file. Extension is automatically defined by whether `make_bam = TRUE`. Default is the basename of `unfiltered_bam` + `.filtered` + extension.
`bowtie2_options`	Optional: Additional parameters that can be passed to the filter_host_bowtie() function. To see all the available parameters use Rbowtie2::bowtie2_usage(). See Details for default parameters. NOTE: Users should pass all their parameters as one string and if optional parameters are given then the user is responsible for entering all the parameters to be used by Bowtie2. The only parameters that should NOT be specified here is the threads.
`YS`	yieldSize, an integer. The number of alignments to be read in from the bam file at once for chunked functions. Default is 100000.
`threads`	The amount of threads available for the function. Default is 1 thread.

`overwrite`	Whether existing files should be overwritten. Default is FALSE.
`quiet`	Turns off most messages. Default is `TRUE`.

Details

A compressed .csv can be created to produce a smaller output file that is created more efficiently and is still compatible with `metascope_id()`.

The default parameters are the same that PathoScope 2.0 uses. "–very-sensitive-local -k 100 –score-min L,20,1.0"

Value

The name of a filtered, sorted .bam file written to the user's current working directory. Or, if `make_bam = FALSE`, a .csv.gz file containing a data frame of only requisite information to run `metascope_id()`.

Examples

```
#### Filter reads from bam file that align to any of the filter libraries

## Assuming a bam file has already been created with align_target_bowtie()
# Create temporary filter library
filter_ref_temp <- tempfile()
dir.create(filter_ref_temp)

## Create temporary taxonomizr accession
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Download reference genome
MetaScope::download_refseq("Orthoebolavirus zairense",
                           reference = FALSE,
                           representative = FALSE,
                           compress = TRUE,
                           out_dir = filter_ref_temp,
                           caching = TRUE,
                           accession_path = tmp_accession)

## Create temp directory to store the indices
index_temp <- tempfile()
dir.create(index_temp)

## Create filter index
MetaScope::mk_bowtie_index(
  ref_dir = filter_ref_temp,
  lib_dir = index_temp,
  lib_name = "filter",
  overwrite = TRUE
)

## Create temporary folder to hold final output file
output_temp <- tempfile()
dir.create(output_temp)
```

```
## Get path to example bam
bamPath <- system.file("extdata", "bowtie_target.bam",
                       package = "MetaScope")
target_copied <- file.path(output_temp, "bowtie_target.bam")
file.copy(bamPath, target_copied)

## Align and filter reads
filter_out <-
  filter_host_bowtie(
    reads_bam = target_copied,
    lib_dir = index_temp,
    libs = "filter",
    threads = 1
  )

## Remove temporary directories
unlink(filter_ref_temp, recursive = TRUE)
unlink(index_temp, recursive = TRUE)
unlink(output_temp, recursive = TRUE)
```

`filter_unmapped_reads` *Filter unmapped reads*

Description

This function will remove all unmapped reads or lines in a .bam file (warning: overwrites the original file!). This function is needed because combining multiple .bam files from different microbial libraries may lead to some reads that mapped to one library and have unmapped entries from another library. This will remove any unmapped entries and leave all reference mapped lines in the .bam file.

Usage

```
filter_unmapped_reads(bamfile)
```

Arguments

`bamfile`	Location for the .bam file to filter & remove all unmapped reads

Details

It is not intended for direct use.

Value

This function will overwrite the existing .bam file with a new .bam file in the same location that has only mapped lines. The function itself returns the output .bam file name.

get_children *Get child nodes from NCBI taxonomy*

Description

This function will utilize an organism classification table to obtain all children species and/or strains with available NCBI reference sequences given a parent taxon and its rank.

Usage

```
get_children(input_taxon, input_rank, tax_dat = NULL)
```

Arguments

`input_taxon`	The parent taxon.
`input_rank`	The taxonomic rank of the input taxon.
`tax_dat`	A dataframe of organism classification information. At minimum, should have a column indicating "strain", and and all others should be taxonomic ranks. Each row should be a taxonomic relationship. This defaults to `NULL`, which calls the ‘taxonomy_table‘ object.

Value

Returns a vector of all the child species and/or strains of the input taxon.

Examples

```
## Get all child species and strains in bacteria superkingdom
get_children('Bacteria','superkingdom')

## Get all child species and strains in fungi kingdom
get_children('Fungi', 'kingdom')

## Get all child species in primate order
get_children('Primates', 'order')
```

locations *Helper Function for MetaScope ID*

Description

Used to create plots of genome coverage for any number of accession numbers

Usage

```
locations(
  which_taxid,
  which_genome,
  accessions,
  taxids,
  reads,
  out_base,
  out_dir
)
```

Arguments

`which_taxid`	Which taxid to plot
`which_genome`	Which genome to plot
`accessions`	List of accessions from `metascope_id()`
`taxids`	List of accessions from `metascope_id()`
`reads`	List of reads from input file
`out_base`	The basename of the input file
`out_dir`	The path to the input file

Value

A plot of the read coverage for a given genome

`merge_bam_files` *Merge multiple .bam files*

Description

This function merges .bam files. It first used the combined_header function to generate a combined header for all the files, reheaders the files, and then merges and sorts the .bam files. It is similar to the 'samtools merge' function, but it allows the .bam files to have different headers. It is not intended for direct use.

Usage

```
merge_bam_files(
  bam_files,
  destination,
  head_file = paste(destination, "_header.sam", sep = ""),
  quiet = TRUE
)
```

Arguments

`bam_files`	A list of file names for the .bam files to be merged.
`destination`	A file name and location for the merged .bam file.
`head_file`	A file name and location for the combined header file. Defaults to the destination. For example, 'example.bam' will be written as 'example.bam'.
`quiet`	Turns off most messages. Default is `TRUE`.

Value

This function merges .bam files and combines them into a single file. The function also outputs the new .bam filename.

`metascope_id` *Identify which genomes are represented in a processed sample*

Description

This function will read in a .bam or .csv.gz file, annotate the taxonomy and genome names, reduce the mapping ambiguity using a mixture model, and output a .csv file with the results. Currently, it assumes that the genome library/.bam files use NCBI accession names for reference names (rnames in .bam file).

Usage

```
metascope_id(
  input_file,
  input_type = "csv.gz",
  aligner = "bowtie2",
  db = "ncbi",
  db_feature_table = NULL,
  accession_path = NULL,
  tmp_dir = dirname(input_file),
  out_dir = dirname(input_file),
  convEM = 1/10000,
  maxitsEM = 25,
  update_bam = FALSE,
  num_species_plot = NULL,
  out_fastas = FALSE,
  num_genomes = 100,
  num_reads = 50,
  quiet = TRUE
)
```

Arguments

`input_file`	The .bam or .csv.gz file of sample reads to be identified.
`input_type`	Extension of file input. Should be either "bam" or "csv.gz". Default is "csv.gz".
`aligner`	The aligner which was used to create the bam file. Default is "bowtie2" but can also be set to "subread" or "other".
`db`	Currently accepts one of `c("ncbi", "silva", "other")` Default is `"ncbi"`, appropriate for samples aligned against indices compiled from NCBI whole genome databases. Alternatively, usage of an alternate database (like Greengenes2) should be specified with `"other"`.
`db_feature_table`	If `db = "other"`, a data.frame must be supplied with two columns, "Feature ID" matching the names of the alignment indices, and a second `character` column supplying the taxon identifying information.
`accession_path`	(character) Filepath to NCBI accessions SQL database. See `taxonomzr::prepareDatabase()`.
`tmp_dir`	Path to a directory to which bam and updated bam files can be saved. Required.
`out_dir`	The directory to which the .csv output file will be output. Defaults to `dirname(input_file)`.
`convEM`	The convergence parameter of the EM algorithm. Default set at `1/10000`.
`maxitsEM`	The maximum number of EM iterations, regardless of whether the convEM is below the threshhold. Default set at `50`. If set at `0`, the algorithm skips the EM step and summarizes the .bam file 'as is'.
`update_bam`	Whether to update BAM file with new read assignments. Default is `FALSE`. If `TRUE`, requires `input_type = "bam"` such that a BAM file is the input to the function.
`num_species_plot`	The number of genome coverage plots to be saved. Default is `NULL`, which saves coverage plots for the ten most highly abundant species.
`out_fastas`	Logical, whether or not to output fasta files of reads. Default is `FALSE`.
`num_genomes`	Number of genomes to output fasta files for `out_fastas`. Default is `100`.
`num_reads`	Number of reads per genome per fasta file for `out_fastas`. Default is `50`.
`quiet`	Turns off most messages. Default is `TRUE`.

Value

This function returns a .csv file with annotated read counts to genomes with mapped reads. The function itself returns the output .csv file name. Depending on the parameters specified, can also output an updated BAM file, and fasta files for additional analysis downstream.

Examples

```
#### Align reads to reference library and then apply metascope_id()
## Assuming filtered bam files already exist

## Create temporary directory
file_temp <- tempfile()
dir.create(file_temp)
```

```
## Get temporary accessions database
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

#### Subread aligned bam file

## Create object with path to filtered subread csv.gz file
filt_file <- "subread_target.filtered.csv.gz"
bamPath <- system.file("extdata", filt_file, package = "MetaScope")
file.copy(bamPath, file_temp)

## Run metascope id with the aligner option set to subread
metascope_id(input_file = file.path(file_temp, filt_file),
             aligner = "subread", num_species_plot = 0,
             input_type = "csv.gz", accession_path = tmp_accession)

#### Bowtie 2 aligned .csv.gz file

## Create object with path to filtered bowtie2 bam file
bowtie_file <- "bowtie_target.filtered.csv.gz"
bamPath <- system.file("extdata", bowtie_file, package = "MetaScope")
file.copy(bamPath, file_temp)

## Run metascope id with the aligner option set to bowtie2
metascope_id(file.path(file_temp, bowtie_file), aligner = "bowtie2",
             num_species_plot = 0, input_type = "csv.gz",
             accession_path = tmp_accession)

## Remove temporary directory
unlink(file_temp, recursive = TRUE)
```

`meta_demultiplex` *Demultiplexing sequencing reads*

Description

Function for demultiplexing sequencing reads arranged in a common format provided by sequencers (such as Illumina) generally for 16S data. This function takes a matrix of sample names/barcodes, a .fastq file of barcodes by sequence header, and a .fastq file of reads corresponding to the barcodes. Based on the barcodes given, the function extracts all reads for the indexed barcode and writes all the reads from that barcode to separate .fastq files.

Usage

```
meta_demultiplex(
  barcodeFile,
  indexFile,
  readFile,
  rcBarcodes = TRUE,
  location = NULL,
  threads = 1,
  hammingDist = 0,
  quiet = TRUE
)
```

Arguments

`barcodeFile`	Path to a file containing a .tsv matrix with a header row, and then sample names (column 1) and barcodes (column 2).
`indexFile`	Path to a .fastq file that contains the barcodes for each read. The headers should be the same (and in the same order) as `readFile`, and the sequence in the `indexFile` should be the corresponding barcode for each read. Quality scores are not considered.
`readFile`	Path to the sequencing read .fastq file that corresponds to the `indexFile`.
`rcBarcodes`	Should the barcode indexes in the barcodeFile be reverse complemented to match the sequences in the `indexFile`? Defaults to `TRUE`.
`location`	A directory location to store the demultiplexed read files. Defaults to generate a new temporary directory.
`threads`	The number of threads to use for parallelization (BiocParallel). This function will parallelize over the barcodes and extract reads for each barcode separately and write them to separate demultiplexed files.
`hammingDist`	Uses a Hamming Distance or number of base differences to allow for inexact matches for the barcodes/indexes. Defaults to `0`. Warning: if the Hamming Distance is `>=1` and this leads to inexact index matches to more than one barcode, that read will be written to more than one demultiplexed read files.
`quiet`	Turns off most messages. Default is `TRUE`.

Value

Returns multiple .fastq files that contain all reads whose index matches the barcodes given. These files will be written to the location directory, and will be named based on the given sampleNames and barcodes, e.g. './demultiplex_fastq/SampleName1_GGAATTATCGGT.fastq.gz'

Examples

```
## Get barcode, index, and read data locations
barcodePath <- system.file("extdata", "barcodes.txt", package = "MetaScope")
indexPath <- system.file("extdata", "virus_example_index.fastq",
                         package = "MetaScope")
readPath <- system.file("extdata", "virus_example.fastq",
                        package = "MetaScope")

## Demultiplex
demult <- meta_demultiplex(barcodePath, indexPath, readPath, rcBarcodes = FALSE,
                      hammingDist = 2)
demult
```

mk_bowtie_index *Make a Bowtie2 index*

Description

This function is a wrapper for the Rbowtie2::bowtie2_build function. It will create either small (.bt2) or large Bowtie2 indexes (.bt2l) depending on the combined size of the reference fasta files.

Usage

```
mk_bowtie_index(
  ref_dir,
  lib_dir,
  lib_name,
  bowtie2_build_options,
  threads = 1,
  overwrite = FALSE
)
```

Arguments

ref_dir	The path to the directory that contains the reference files either uncompressed or compressed (.gz). NOTE: This directory should contain only the reference fasta files to be indexed.
lib_dir	The path to the directory where Bowtie2 index files should be created.
lib_name	The basename of the index file to be created (without the .bt2 or .bt2l extension)
bowtie2_build_options	Optional: Options that can be passed to the mk_bowtie_index() function. All options should be passed as one string. To see all the available options that can be passed to the function use Rbowtie2::bowtie2_build_usage(). NOTE: Do not specify threads here.
threads	The number of threads available to the function. Default is 1 thread.
overwrite	Whether existing files should be overwritten. Default is FALSE.

Value

Creates the Bowtie2 indexes of the supplied reference .fasta files. Returns the path to the directory containing these files.

Examples

```
#### Create a bowtie index from the example reference library

## Create a temporary directory to store the reference library
ref_temp <- tempfile()
dir.create(ref_temp)
```

```
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Download reference genome
download_refseq('Bovismacovirus', reference = FALSE, representative = FALSE,
                out_dir = ref_temp, compress = TRUE, patho_out = FALSE,
                caching = TRUE, accession_path = tmp_accession)

## Create the reference library index files in the current directory
mk_bowtie_index(ref_dir = ref_temp, lib_dir = ref_temp,
                lib_name = "target", threads = 1, overwrite = FALSE)

## Remove temporary directory
unlink(ref_temp, recursive = TRUE)
```

mk_subread_index *Make a Subread index*

Description

This function is a wrapper for the `Rsubread::buildindex` function. It will generate one or more Subread indexes from a .fasta file. If the library is too large (default >4GB) it will automatically be split into multiple indexes, with _1, _2, etc at the end of the ref_lib basename.

Usage

```
mk_subread_index(ref_lib, split = 4, mem = 8000, quiet = TRUE)
```

Arguments

`ref_lib`	The name/location of the reference library file, in (uncompressed) .fasta format.
`split`	The maximum allowed size of the genome file (in GB). If the `ref_lib` file is larger than this, the function will split the library into multiple parts.
`mem`	The maximum amount of memory (in MB) that can be used by the index generation process (used by the Rsubread::buildindex function).
`quiet`	Turns off most messages. Default is `TRUE`.

Value

Creates one or more Subread indexes for the supplied reference .fasta file. If multiple indexes are created, the libraries will be named the `ref_lib` basename + "_1", "_2", etc. The function returns the names of the folders holding these files.

Examples

```
#### Create a subread index from the example reference library

## Create a temporary directory to store the reference library
ref_temp <- tempfile()
dir.create(ref_temp)
tmp_accession <- system.file("extdata", "example_accessions.sql", package = "MetaScope")

## Download reference genome
out_fasta <- download_refseq('Orthoebolavirus zairense', reference = FALSE,
                             representative = FALSE, out_dir = ref_temp,
                             compress = TRUE, patho_out = FALSE,
                             caching = TRUE, accession_path = tmp_accession)

## Make subread index of reference library
mk_subread_index(out_fasta)
unlink(ref_temp)
```

`remove_matches` *Helper function to remove reads matched to filter libraries*

Description

Using the `filter_host()` function, we align our sequencing sample to all filter libraries of interest. The `remove_matches()` function allows for removal of any target reads that are also aligned to filter libraries.

Usage

```
remove_matches(
  reads_bam,
  read_names,
  output,
  YS,
  threads,
  aligner,
  make_bam,
  quiet
)
```

Arguments

`reads_bam`	The name of a merged, sorted .bam file that has previously been aligned to a reference library. Likely, the output from running an instance of `align_target()`.
`read_names`	A `list` of target query names from `reads_bam` that have also aligned to a filter reference library. Each `list` element should be a vector of read names.

`output`	The name of the .bam or .csv.gz file that to which the filtered alignments will be written.
`YS`	yieldSize, an integer. The number of alignments to be read in from the bam file at once for chunked functions. Default is 100000.
`threads`	The number of threads to be used in filtering the bam file. Default is 1.
`aligner`	The aligner which was used to create the bam file.
`make_bam`	Logical, whether to also output a bam file with host reads filtered out. A .csv.gz file will be created instead if `FALSE`. Creating a bam file is costly on resources over creating a compressed csv file with only relevant information, so default is `FALSE`.
`quiet`	Turns off most messages. Default is `TRUE`.

Details

This function is not intended for direct use.

Value

Depending on input `make_bam`, either the name of a filtered, sorted .bam file written to the user's current working directory, or an RDS file containing a data frame of only requisite information to run `metascope_id()`.

Index