Package 'seekr'

July 11, 2026

Title Search, Inspect, and Replace Text Across Files

Version 0.2.0

Description An inspectable and composable workflow for search-and-replace in text files. Files can be listed, filtered, and searched separately, with inspectable exclusions showing what was excluded and why. Matches are represented as structured vectors that can be printed with context, summarized, and filtered. Replacements can be defined at search time or set and updated after the search. Only matches present in the vector are modified when files are written. Backup and restore helpers are provided for file workflows. Text that has already been read can also be searched and updated directly, giving users control over input, output, and encoding when needed.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1)

Suggests testthat, withr, diffobj, dplyr

Config/testthat/edition 3

Imports checkmate, cli (>= 3.6.1), fs, glue, methods, mime, pillar, processx, purrr, rappdirs, readr, rlang (>= 1.1.1), stringi, stringr, tibble, utils, vctrs (>= 0.6.3)

URL https://github.com/smartiing/seekr, https://smartiing.github.io/seekr/

BugReports https://github.com/smartiing/seekr/issues

Config/roxygen2/version 8.0.0

Config/Needs/website rmarkdown

NeedsCompilation no

Author Sacha Martingay [aut, cre, cph]

Maintainer Sacha Martingay <martingay.sacha@hotmail.com>

Repository CRAN

Date/Publication 2026-07-10 23:10:02 UTC

Contents

as_seekr_path . . . 2
as_tibble.seekr_match . . . 3
backups . . . 5
empty_stage . . . 8
exclude_functions . . . 9
exclusions . . . 10
filter_files . . . 11
filter_match . . . 13
list_files . . . 15
match_files . . . 17
match_text . . . 19
print.seekr_match . . . 20
replace_files . . . 22
replace_text . . . 25
restore_files . . . 26
seek . . . 29
seekr_match . . . 35
seekr_option . . . 37
seekr_options . . . 38
sort_within_files . . . 40
str.seekr_match . . . 40
summary.seekr_match . . . 41
with_capture_groups_matrix . . . 42

Index **44**

`as_seekr_path` *Normalize file paths for seekr*

Description

`as_seekr_path()` converts paths to the normalized format used internally by seekr for listing, filtering, matching, and replacement.

This is useful when writing custom `exclude` functions, comparing paths with seekr results, or building file vectors before calling `filter_files()` or `match_files()`.

Usage

```
as_seekr_path(x)
```

Arguments

`x`	A character vector of paths.

Details

A **seekr path** is a character path that has been:

- expanded, so ~ is resolved,
- normalized, so redundant path components are removed,
- resolved to an absolute path,
- represented with forward slashes.

Where seekr paths are created:
`list_files()` returns seekr paths: it starts from user-supplied directories and returns the listed files as normalized absolute paths.
`filter_files()` and `match_files()` normalize their input `path` before filtering or matching. This means path-based filters such as `path_pattern` are applied to seekr paths, regardless of whether the input paths were originally relative, absolute, or written with platform-specific separators.

Why this matters:
Normalizing paths makes path filtering more predictable. A `path_pattern` is matched against a stable representation of the file path instead of depending on how the path was originally written by the user.

Value

A character vector of normalized absolute paths, with the same length and order as `x`.

See Also

`list_files()`, `filter_files()`, `match_files()`

Examples

```
as_seekr_path(".")
as_seekr_path(c(".", "~"))
```

---

as_tibble.seekr_match *Convert* seekr_match *vectors to and from data frames*

Description

`as_tibble()` and `as.data.frame()` convert a `seekr_match` vector into a tibble or plain data frame, with one row per match and one column per field.

`as_match()` is the reverse: it converts a data frame back into a `seekr_match` vector, validating all fields and checking for overlapping matches within each file before returning.

Together, these functions unlock the full tidyverse toolkit for manipulating match metadata. A typical pattern is to convert to a tibble, use `dplyr::mutate()` or `dplyr::filter()` to derive or modify columns, including the `replacement` field, and then convert back with `as_match()` before calling `replace_files()`.

Usage

```
## S3 method for class 'seekr_match'
as_tibble(x, ...)

## S3 method for class 'seekr_match'
as.data.frame(x, ...)

as_match(x)
```

Arguments

x	For `as_tibble()` and `as.data.frame()`: a `seekr_match` vector. For `as_match()`: a data frame with at least the columns listed in the **Fields** section of seekr_match. Additional columns are silently ignored. All required columns must have the correct type: `character` for string fields, `integer` for position fields.
...	Not used. Present for compatibility with S3 generics.

Value

`as_tibble()` returns a `tbl_df`. `as.data.frame()` returns a plain `data.frame`. In both cases the result has one row per match and one column per field (see the **Fields** section of seekr_match).

`as_match()` returns a `seekr_match` vector. Matches within each file are sorted by position. Overlapping or incoherent matches within the same file cause an error.

Note: the `empty_stage` and `exclusions` attributes of the original `seekr_match` vector are not preserved through the conversion.

See Also

- seekr_match for the list of available fields.
- `filter_match()` for simpler subsetting that does not require conversion.
- `vctrs::field()` to access or modify a single field in place.
- `replace_files()` to apply staged replacements after converting back.

Examples

```
## Not run:
library(dplyr)
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("TODO", path = ext_path)

# Convert to tibble
df <- as_tibble(x)
df

# Convert to plain data frame
as.data.frame(x)
```

```
# Convert back to seekr_match
as_match(df)

# Set a replacement for all matches
df$replacement <- "DONE"
as_match(df)

# Suppose you want to replace `"foo"` with `"bar"`, but only for the last
# match in each file, and only in files with at least three matches:
x <- seekr("foo", "bar", path = ext_path)

y <-
  x |>
  as_tibble() |>
  mutate(
    ith_match_per_file_rev = n():1L,
    n_match_per_file = n(),
    .by = path
  ) |>
  filter(ith_match_per_file_rev == 1L, n_match_per_file >= 3L) |>
  as_match()

# replace_files(y)

## End(Not run)
```

---

backups *List, inspect, open, and delete backups*

Description

By default, seekr creates a backup before `replace_files()` and `restore_files()` modify files. These functions let you inspect and manage existing backups in a particular `backup_dir`.

- `list_backups()` returns a tibble of all existing backups.
- `last_backup()` returns only the most recent backup.
- `delete_backups()` permanently deletes one or more backups by their `id`.
- `open_backup_dir()` opens the current default backup directory.

Usage

```
list_backups(backup_dir = seekr_option("seekr.backup_dir"))

last_backup(backup_dir = seekr_option("seekr.backup_dir"))

delete_backups(
  id,
  ...,
```

```
  backup_dir = seekr_option("seekr.backup_dir"),
  .progress = seekr_option("seekr.progress")
)

open_backup_dir(backup_dir = seekr_option("seekr.backup_dir"))
```

Arguments

backup_dir	Path to the backup directory. Defaults to `seekr_option("seekr.backup_dir")`, which resolves to a platform-specific user data directory. Change the default globally with `options(seekr.backup_dir = "/your/path")`.
id	Integer vector of backup ids to delete, as found in the `id` column of `list_backups()`. Duplicate ids are silently deduplicated. Ids that do not correspond to any existing backup are silently ignored.
...	These dots are for future extensions and must be empty.
.progress	Whether to display progress messages. Default is `TRUE` in interactive sessions and `FALSE` otherwise (see `rlang::is_interactive()`). Can be set globally with `options(seekr.progress = FALSE)`.

Value

- `list_backups()` and `last_backup()` return a tibble as described in the **Backup table columns** section. An empty tibble with the correct columns is returned when no backups exist. `last_backup()` returns all rows belonging to the most recent backup operation.
- `delete_backups()` invisibly returns the paths of the deleted backup subdirectories.
- `open_backup_dir()` invisibly returns the path to the backup directory.

Backups

Backups are stored under `backup_dir`, which defaults to a platform-specific user data directory managed by `rappdirs::user_data_dir()`. Each backup occupies its own numbered subdirectory (`000001/`, `000002/`, etc.) containing a copy of each file and a `backup.RDS` metadata file.

The default backup directory can be changed globally with:

```
options(seekr.backup_dir = "/your/preferred/path")
```

Backup table columns

`list_backups()` and `last_backup()` return a tibble with one row per backed-up file and the following columns:

- `id`: integer backup identifier, derived from the subdirectory name. All rows belonging to the same backup operation share the same `id`.
- `created_at`: date-time when the backup was created.
- `operation`: either `"replace"` (backup created by `replace_files()`) or `"restore"` (backup created by `restore_files()`).
- `description`: optional description provided at backup time, or `NA`.

- `original`: absolute path to the original file.
- `backup`: absolute path to the backup copy.
- `original_exists`: whether the original file still exists on disk.
- `backup_exists`: whether the backup copy still exists on disk.
- `size`: size of the backup copy, as an `fs_bytes` value.

See Also

- `replace_files()` to apply replacements and create backups.
- `restore_files()` and `restore_files_interactive()` to restore files.

Examples

```
# Set up a temporary project with two files
project_dir <- tempfile("seekr_project")
dir.create(project_dir)

backup_dir <- tempfile("seekr_backups")
dir.create(backup_dir)

file1 <- file.path(project_dir, "script1.R")
file2 <- file.path(project_dir, "script2.R")
writeLines("old_name <- function(x) x + 1", file1)
writeLines("y <- old_name(2)", file2)

# Search for the pattern we want to replace
x <- seekr("old_name", replacement = "new_name", path = project_dir)
x

# No backup exists yet
list_backups(backup_dir = backup_dir)

# Apply the replacement; a backup is created automatically
replace_files(x, backup_dir = backup_dir)

# The backup now contains the original (pre-replacement) version
# of both files, under id = 1
list_backups(backup_dir = backup_dir)

# The files on disk have changed
readLines(file1)
readLines(file2)

# Searching for the original pattern no longer finds anything
seekr("old_name", path = project_dir)

# Restore the files from the backup; this itself creates a second
# backup (id = 2) of the current (replaced) version, before overwriting
b <- last_backup(backup_dir = backup_dir)
restore_files(from = b$backup, to = b$original, backup_dir = backup_dir)
```

```
# We now have two backups: id = 1 is the state before replace_files(),
# id = 2 is the state before restore_files() (i.e. the replaced version)
list_backups(backup_dir = backup_dir)

# The files are back to their original content
readLines(file1)
readLines(file2)

# The original pattern is found again
seekr("old_name", path = project_dir)

## Not run:
# open_backup_dir() opens a file browser, so it is not run here
open_backup_dir()

# Review each file interactively before restoring
restore_files_interactive(from = b$backup, to = b$original)

# Customize the diff display (unified format)
restore_files_interactive(
  from = b$backup,
  to = b$original,
  mode = "unified",
  color.mode = "yb"
)

# Finally, particular backups in a backup_dir can be deleted
b <- list_backups(backup_dir = backup_dir)
b
delete_backups(id = b$id, backup_dir = backup_dir)

## End(Not run)


unlink(project_dir, recursive = TRUE)
unlink(backup_dir, recursive = TRUE)
```

`empty_stage` *Diagnose where a workflow became empty*

Description

`empty_stage()` retrieves the pipeline stage that produced an empty `seekr_match` result returned by `seek()` or `seekr()`.

Empty results can happen at different stages of the pipeline:

- `"input"`: the input `path` was empty.
- `"list"`: no files were found by `list_files()`.

- "filter": all files were excluded by filter_files().
- "match": files were searched, but no match was found.

For non-empty results, empty_stage() returns NULL.

Usage

```
empty_stage(x)
```

Arguments

x	A seekr_match object, returned by either seek() or seekr().

Value

Either NULL for non-empty results, or one of "input", "list", "filter", or "match" for empty results.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seek("pattern_that_does_not_exist", path = ext_path)
empty_stage(x)
```

---

exclude_functions *Default file exclusion functions*

Description

exclude_functions is the default named list of functions used by filter_files() to exclude files that are usually not useful for text search.

Usage

```
exclude_functions

is_git_dir(path)

is_dependency_dir(path)

is_minified_file(path)

is_not_text_mime(path)
```

Arguments

path	A character vector of file paths to filter.

Format

A named list of exclude functions.

Details

Each function receives a character vector of normalized file paths and must return a logical vector of the same length. `TRUE` means that the corresponding file should be excluded.

The default pipeline includes:

- `is_git_dir()`: excludes files located inside `.git/`.
- `is_dependency_dir()`: excludes files in common dependency folders such as `node_modules/`, `renv/`, `.venv/`, `vendor/`, and `__pycache__/`.
- `is_minified_file()`: excludes minified or bundled files such as `.min.js`, `.bundle.css`, etc.
- `is_not_text_mime()`: excludes files not recognized as text based on their MIME type.

You can disable all exclude functions with `exclude = NULL`, remove one of the defaults by setting it to `NULL` in a copy of exclude_functions, or add your own named function to the list.

See Also

`filter_files()`

Examples

```
ext_path <- system.file("extdata", package = "seekr")
files <- list_files(path = ext_path)
names(exclude_functions)

# Disable default exclude functions
filter_files(files, exclude = NULL)

# Add a custom exclude function
my_fns <- exclude_functions
my_fns$generated <- function(path) grepl("/generated/", path)
filter_files(files, exclude = my_fns)
```

`exclusions` *Inspect why files were excluded*

Description

`exclusions()` retrieves the exclusion details stored on objects returned by `filter_files()`, `seek()`, and `seekr()`.

Usage

```
exclusions(x)
```

Arguments

x	An object containing exclusion details, typically the result of `filter_files()`, `seek()`, or `seekr()`.

Value

A data frame with one row per input file and one column per exclusion filter, describing which files were excluded and why. If no exclusion details are available, returns `NULL`.

See Also

`filter_files()` for the filtering step, exclude_functions for the default exclude-function pipeline.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
files <- list_files(path = ext_path)
filtered <- filter_files(files, extension = "R")
exclusions(filtered)
```

`filter_files` *Filter files to search*

Description

`filter_files()` keeps files matching `extension` and `path_pattern`, and not exceeding `max_file_size`. Finally, the `exclude` functions are applied to the remaining files, discarding common non-text or irrelevant files by default. It is the second step of the `seek()` pipeline, applied after `list_files()` and before `match_files()`.

Exclusion filters are applied in this order:

1. `extension`: keeps only files whose extension is in the provided list.
2. `path_pattern`: keeps files whose normalized path matches the pattern.
3. `max_file_size`: excludes files larger than the given size.
4. `exclude_functions`: applies each named function to the remaining files.

Files are only passed to each subsequent filter if they have not already been excluded by a previous one.

Details about excluded files are stored on the result and can be retrieved with `exclusions()`.

Usage

```
filter_files(
  path,
  ...,
  extension = NULL,
  path_pattern = NULL,
  max_file_size = Inf,
  exclude = seekr::exclude_functions,
  .progress = seekr_option("seekr.progress")
)
```

Arguments

path	A character vector of file paths to filter.
...	These dots are for future extensions and must be empty.
extension	Optional character vector of file extensions to keep. Either: • NULL (default): no filtering by extension; all extensions are kept. • A character vector of extensions to keep, with or without a leading dot (e.g. c("R", ".Rmd", "qmd")). Extensions are normalized before matching: leading dots are stripped, matching is case-insensitive, and duplicates are ignored. Only the last component of compound extensions is used (e.g. "tar.gz" uses "gz"), with a warning.
path_pattern	Optional pattern applied to filter normalized file paths (see as_seekr_path()). Either: • NULL (default): no filtering by path. • A string, interpreted as a regular expression via stringr::regex(). • A stringr_pattern object such as stringr::regex() or stringr::fixed().
max_file_size	Maximum file size in bytes. Files larger than this value are excluded. Default is Inf, meaning no files are excluded by size. Zero and negative values are treated as Inf.
exclude	Named list of functions used to exclude unwanted files during filtering. Either: • NULL, to disable additional exclude functions. • A named list of functions, each taking a character vector of normalized file paths and returning a logical vector of the same length, where TRUE means the file should be excluded. Defaults to exclude_functions, which excludes common non-text or irrelevant files.
.progress	Whether to display progress messages. Default is TRUE in interactive sessions and FALSE otherwise (see rlang::is_interactive()). Can be set globally with options(seekr.progress = FALSE).

Value

A character vector of normalized absolute paths that passed all filters. Paths use the same representation as `as_seekr_path()`.

An attribute `"exclusions"` is always attached to the result, containing a data frame with one row per input file and one column per exclusion function, detailing which files were excluded and why. Retrieve it with `exclusions()`.

See Also

- `list_files()` to produce the input paths.
- `match_files()` to search the filtered files for a pattern.
- `exclusions()` to inspect which files were removed and why.
- `seek()` to run the full pipeline.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
files <- list_files(path = ext_path)

# Keep only R files
filter_files(files, extension = "R")

# Keep only files in the R/ subfolder
filter_files(files, path_pattern = "/R/")

# Exclude files larger than 1 MB
filter_files(files, max_file_size = fs::fs_bytes("1MB"))

# Inspect which files were excluded and why
filtered <- filter_files(files, extension = "R")
exclusions(filtered)

# Disable default exclude functions
filter_files(files, exclude = NULL)

# Add a custom exclude function
my_fns <- exclude_functions
my_fns$generated <- function(path) grepl("/generated/", path)
filter_files(files, exclude = my_fns)
```

`filter_match` *Filter matches*

Description

`filter_match()` subsets a `seekr_match` vector using expressions evaluated directly against its fields, without needing to call `vctrs::field()` explicitly.

The two calls below are equivalent:

```
x[field(x, "start_line") > 10]
filter_match(x, start_line > 10)
```

`filter_match()` is modelled after `dplyr::filter()`: field names can be used directly in expressions, and multiple expressions are combined with `&`. This makes it easy to write readable multi-condition filters:

```
x |> filter_match(
  grepl("/R/", path),
  match == "TODO",
  start_line > 10
)
```

Usage

```
filter_match(x, ...)
```

Arguments

`x`	A `seekr_match` vector.
`...`	One or more filtering expressions, evaluated against the fields of `x`. Field names (`path`, `match`, `start_line`, `replacement`, etc.) can be used directly. Each expression must return a logical vector of the same length as `x`, with no missing values. Multiple expressions are combined with `&`.

Value

A `seekr_match` vector containing only the matches for which all expressions evaluated to `TRUE`. If no expressions are supplied, `x` is returned unchanged.

Differences from base R subsetting

Unlike `x[condition]`, `filter_match()` does not recycle logical vectors. Each expression must return exactly `length(x)` values. Missing values (`NA`) in any expression cause an error. This prevents silent mistakes from implicit recycling or incomplete conditions.

See Also

- `vctrs::field()` to access a field directly for use in base R subsetting.
- `as_tibble.seekr_match()` and `as_match()` for more complex workflows that require tabular manipulation.
- `replace_files()` to apply staged replacements after filtering.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("TODO|FIXME", path = ext_path)

# Filter by line number
filter_match(x, start_line > 10)

# Filter by file path
filter_match(x, grepl("/R/", path))

# Filter by matched text
filter_match(x, match == "TODO")

# Combine multiple conditions
filter_match(x, match == "TODO", start_line > 10, grepl("/R/", path))

# Equivalent base R subsetting (more verbose)
x[
  field(x, "match") == "TODO" &
  field(x, "start_line") > 10 &
  grepl("/R/", field(x, "path"))
]
```

---

list_files *List files to search*

Description

`list_files()` starts from `path` and lists candidate files. It can recurse into subdirectories with `recurse`, include hidden files and directories with `all`, and optionally restrict discovery inside Git repositories with `use_git`. It is the first step of the `seek()` pipeline.

Listing is intentionally simple: it does not know about patterns, extensions, file sizes, or MIME types. Its only job is to turn directories into a character vector of file paths. Filtering happens in the next step, `filter_files()`.

If `use_git = TRUE`, Git is used for each input path independently. For each path, `list_files()` asks Git whether that path is inside a Git repository. If it is, `list_files()` finds the repository root by walking upward from that path, then keeps only the files also returned by `git ls-files --cached --others --exclude-st` for that repository.

Git is used to restrict the files discovered from the input path. It does not expand the search. The `path`, `recurse`, and `all` arguments still define the initial candidate files. For example, Git-tracked hidden files are not returned unless `all = TRUE`, and Git-tracked files below the requested recursion depth are not returned.

`list_files()` does not search downward for nested Git repositories. If an input path is not inside a Git repository, it is listed normally, even if it contains Git repositories in subdirectories. If you want Git-aware discovery for nested repositories, pass those repository directories explicitly in `path`.

If `use_git = TRUE`, Git must be installed and available on `PATH`.

The returned paths are normalized as described in `as_seekr_path()`.

Usage

```
list_files(
  path = ".",
  ...,
  recurse = TRUE,
  all = FALSE,
  use_git = FALSE,
  .progress = seekr_option("seekr.progress")
)
```

Arguments

path	A character vector of one or more existing directories to search in. Defaults to "." (the current working directory).
...	These dots are for future extensions and must be empty.
recurse	Controls how deep the directory traversal goes to list files. Either: • TRUE (default): recurse into all subdirectories. • FALSE: only list files at the top level of each directory in path. • A positive integer: limit recursion to that many levels deep.
all	Whether to list hidden files and directories. Default is FALSE.
use_git	Should Git be used to restrict file discovery inside Git repositories? If TRUE, list_files() keeps only files that were first discovered according to path, recurse, and all, and are also returned by git ls-files --cached --others --exclude-standard. use_git = TRUE looks for the Git root by walking upward from each supplied path, but it does not recursively search downward for Git repositories in subdirectories. Git must be installed and available on PATH.
.progress	Whether to display progress messages. Default is TRUE in interactive sessions and FALSE otherwise (see rlang::is_interactive()). Can be set globally with options(seekr.progress = FALSE).

Value

A character vector of normalized absolute file paths. Returns an empty character vector if no files are found or if path is empty.

See Also

- filter_files() to filter the listed files before searching matches.
- seek() to run the full listing, filtering, and matching pipeline.

Examples

```
ext_path <- system.file("extdata", package = "seekr")

# List all files in the example directory
list_files(path = ext_path)
```

```
# List only files at the top level, without recursing
list_files(path = ext_path, recurse = FALSE)

# Recurse at most 2 levels deep
list_files(path = ext_path, recurse = 2L)

# Include hidden files and directories
list_files(path = ext_path, all = TRUE)

## Not run:
# Use Git to restrict discovery inside Git repositories
list_files(path = ".", use_git = TRUE)

## End(Not run)
```

`match_files` *Find matches in files*

Description

`match_files()` reads each file, decodes them using `encoding`, finds `pattern` matches, and captures surrounding `context` lines. A `replacement` can be provided to stage changes for later application with `replace_files()`. It is the third and final step of the `seek()` pipeline, applied after `list_files()` and `filter_files()`.

Usage

```
match_files(
  path,
  pattern,
  replacement = NULL,
  ...,
  context = 5L,
  encoding = "UTF-8",
  .progress = seekr_option("seekr.progress")
)
```

Arguments

`path`	A character vector of file paths to read and search.
`pattern`	Pattern to search for, matched using stringr (ICU regular expressions). Either: • A string, automatically wrapped as `stringr::regex()` with `ignore_case = FALSE`, `multiline = TRUE`, `comments = FALSE`, and `dotall = FALSE`. • A `stringr_pattern` object such as `stringr::regex()`, `stringr::fixed()`, or `stringr::coll()`, used as-is for more control.
`replacement`	Replacement to associate with each match. Replacements are computed immediately during the search and stored in the result. Either:

- NULL (default): no replacement. replace_files() cannot be called without setting replacements first.
- A plain string, used literally as replacement text.
- A string with backreferences of the form \1, \2, etc., replaced with the corresponding capture group from pattern.
- A function, called once per file with a character vector of all matches found in that file, and expected to return a character vector of the same length (e.g. toupper).
- A function wrapped with with_capture_groups_matrix(), called once per file with a character matrix where the first column is the full match and the remaining columns are the capture groups.

`...` These dots are for future extensions and must be empty.

`context` Number of surrounding lines to capture around each match. Either:

- A single non-negative integer (default: 5L): captures the same number of lines before and after each match.
- A pair of non-negative integers c(before, after): captures before lines before and after lines after each match.

`encoding` Encoding used to decode file content during the matching step. Either:

- A single string (default: "UTF-8"), applied to all files.
- NULL: encoding is guessed for each file individually using stringi::stri_enc_detect(), falling back to "UTF-8" when detection fails.

Note: replace_files() always writes files in UTF-8. A warning is issued once per session when any file is read with a non-UTF-8 encoding. By default, replace_files() refuses to write those matches unless allow_encoding_change = TRUE is set.

`.progress` Whether to display progress messages. Default is TRUE in interactive sessions and FALSE otherwise (see rlang::is_interactive()). Can be set globally with options(seekr.progress = FALSE).

Value

A seekr_match vector. Each element represents one match and carries the file path, match positions, matched text, optional replacement, context lines, encoding, and a hash of the searched text used for replacement safety. Returns an empty seekr_match vector when no matches are found.

Files that no longer exist before matching are skipped with a warning. Files that contain unsupported null bytes are also skipped with a warning. Other read or decoding errors abort.

Note

For advanced use cases where you want to search for a pattern in-memory text, see match_text().

See Also

- match_text() to search for a pattern in in-memory text.
- filter_files() to filter files before matching.
- replace_files() to apply staged replacements.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
files <- ext_path |> list_files() |> filter_files(extension = "R")

# Search for a pattern
match_files(files, "TODO")

# Capture more context lines
match_files(files, "TODO", context = 10L)

# Stage a replacement
match_files(files, "old_fn", replacement = "new_fn")
```

---

match_text *Find matches in text*

Description

`match_text()` searches text that has already been read into R. It is the text-level counterpart of `match_files()`: it does not read from disk and does not record file encoding information.

Usage

```
match_text(text, path, pattern, replacement = NULL, ..., context = 5L)
```

Arguments

`text`	Text content as a single string.
`path`	Source identifier associated with `text`. This is stored in the resulting `seekr_match` object and used later for inspection and diagnostics. It may be a real file path, but it does not need to point to an existing file unless the result is later passed to `replace_files()`.
`pattern`	Pattern to search for, matched using stringr (ICU regular expressions). Either: • A string, automatically wrapped as `stringr::regex()` with `ignore_case = FALSE`, `multiline = TRUE`, `comments = FALSE`, and `dotall = FALSE`. • A `stringr_pattern` object such as `stringr::regex()`, `stringr::fixed()`, or `stringr::coll()`, used as-is for more control.
`replacement`	Replacement to associate with each match. Replacements are computed immediately during the search and stored in the result. Either: • `NULL` (default): no replacement. `replace_files()` cannot be called without setting replacements first. • A plain string, used literally as replacement text. • A string with backreferences of the form `\1`, `\2`, etc., replaced with the corresponding capture group from `pattern`.

- A function, called once per file with a character vector of all matches found in that file, and expected to return a character vector of the same length (e.g. toupper).
- A function wrapped with `with_capture_groups_matrix()`, called once per file with a character matrix where the first column is the full match and the remaining columns are the capture groups.

`...` These dots are for future extensions and must be empty.

`context` Number of surrounding lines to capture around each match. Either:

- A single non-negative integer (default: `5L`): captures the same number of lines before and after each match.
- A pair of non-negative integers `c(before, after)`: captures `before` lines before and `after` lines after each match.

Details

The resulting `seekr_match` vector can be inspected, summarized, filtered, updated, and passed to `replace_text()`. It is not intended to be passed to `replace_files()`, because `seekr` did not read the source file and does not control how the text was decoded.

Use `match_files()` or `seek()`/`seekr()` for the usual file workflow.

Value

A `seekr_match` vector.

Examples

```
text <- "Commodo labore culpa ullamco TODO irure laboris FIXME Lorem sunt sint"
x <- match_text(text = text, path = "lorem.txt", pattern = "TODO")
y <- match_text(text = text, path = "lorem.txt", pattern = "FIXME")
z <- c(x, y)
z
```

`print.seekr_match` *Print matches with context and replacement preview*

Description

`print()` displays a `seekr_match` vector in a readable console format, grouped by source. Each printed match is shown with its source, match index, line number, matched text, and, when available, its staged replacement.

The amount of output can be controlled with `n`, which limits the number of matches printed. `context` controls how many surrounding lines are shown.

Usage

```
## S3 method for class 'seekr_match'
print(x, ..., n = NULL, context = 0L)
```

Arguments

x	A seekr_match vector.
...	Not used. Present for compatibility with the print() generic.
n	Maximum number of matches to print. If NULL, a compact default is used: all matches are printed for small vectors, and only the first matches are printed for larger vectors. Use Inf to print all matches.
context	Number of context lines to print around each match. Either: • A single non-negative integer: print that many lines before and after each match. • A pair of non-negative integers c(before, after): print before lines before and after lines after each match. Only context lines captured when the seekr_match vector was created can be printed.

Value

Invisibly returns the original seekr_match vector x.

See Also

summary() for a compact summary of matches, filter_match() to subset matches before printing, and seek() for the context argument that controls how many surrounding lines are captured.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seekr(
  pattern = "(\\w+)(?= <- function)",
  replacement = toupper,
  path = ext_path
)

# Print up to 10 matches (default)
print(x)

# Print all matches
print(x, n = Inf)

# Print only 3 matches
print(x, n = 3L)

# Reduce context to 1 line around each match
print(x, context = 1L)
```

```
# Show 3 lines before and 1 line after each match
print(x, context = c(3L, 1L))
```

`replace_files` *Replace selected matches in files*

Description

`replace_files()` applies the replacements stored in a seekr_match object to the corresponding files and writes (in **UTF-8**) the modified files back to disk.

It is the final step of the usual seekr workflow: search with `seek()`/`match_files()`, inspect or modify the resulting matches, then apply the replacements with `replace_files()`.

Usage

```
replace_files(
  x,
  ...,
  backup = TRUE,
  description = NA_character_,
  allow_encoding_change = FALSE,
  backup_dir = seekr_option("seekr.backup_dir"),
  .progress = seekr_option("seekr.progress")
)
```

Arguments

`x`	A `seekr_match` object with replacement values. Typically created by `seek()`, `seekr()`, `match_files()`, or `match_text()`. Replacement field must be set for all matches, either computed when searching for matches or modified manually before calling `replace_files()`.
`...`	These dots are for future extensions and must be empty.
`backup`	Whether to create a backup of the current files before overwriting the files. Default is `TRUE`. Set to `FALSE` to skip the backup, for example if the files are already tracked by another version control system.
`description`	Optional character string stored in the backup metadata when `backup = TRUE`. Use it to describe why the backup was created. Defaults to `NA`.
`allow_encoding_change`	Should `replace_files()` allow files that were read with a non-UTF-8 encoding to be written back in UTF-8? The default is `FALSE` as this should be intentional.
`backup_dir`	Path to the backup directory. Defaults to `seekr_option("seekr.backup_dir")`, which resolves to a platform-specific user data directory. Change the default globally with `options(seekr.backup_dir = "/your/path")`.
`.progress`	Whether to display progress messages. Default is `TRUE` in interactive sessions and `FALSE` otherwise (see `rlang::is_interactive()`). Can be set globally with `options(seekr.progress = FALSE)`.

Details

`replace_files()` is designed to be conservative. Before any file is modified, it checks that the replacements stored in `x` can still be applied safely.

Missing files are detected before the replacement process starts.

This safety check is intentionally performed in two passes. In the first pass, before creating backups, `replace_files()` re-reads every target file and verifies that its current text has the same hash as the text that was searched when the matches were created. If any file fails this check, the operation aborts before backups are created and before any file is modified. This helps avoid partial replacements, for example replacing the first few files successfully and then failing on a later file.

If all files pass the first check, backups are created when `backup = TRUE`. Then `replace_files()` loops over the files again. For each file, it re-reads the current content, verifies the hash a second time, applies the replacements in memory, and writes the modified text back to disk in UTF-8. This second check protects against concurrent edits that may happen between the initial verification and the actual write.

Missing files are skipped with a warning. In that case, the returned `seekr_match` object contains only the matches that were actually replaced.

Replacements are applied file by file. Within each file, matches are replaced from the end of the file to the beginning, so earlier replacements do not shift the recorded positions of later replacements.

By default, a backup of the current version of each modified file is created before replacement. Use `list_backups()` and `last_backup()` to inspect backups, and `restore_files()` or `restore_files_interactive()` to restore them.

For advanced in-memory text workflows where you do not want seekr to read, write, or create backups, use `replace_text()`.

Value

Invisibly returns the `seekr_match` object containing the matches that were actually replaced. If missing files were skipped, matches from those files are not included in the returned object.

See Also

- `seek()` and `seekr()` to create matches with staged replacements.
- `filter_match()` to keep only some matches before replacing.
- `list_backups()` and `last_backup()` to inspect backups.
- `restore_files()` and `restore_files_interactive()` to restore files.

Examples

```
# Set up a temporary project with two files
project_dir <- tempfile("seekr_project")
dir.create(project_dir)

backup_dir <- tempfile("seekr_backups")
dir.create(backup_dir)

file1 <- file.path(project_dir, "script1.R")
```

```
file2 <- file.path(project_dir, "script2.R")
writeLines("old_name <- function(x) x + 1", file1)
writeLines("y <- old_name(2)", file2)

# Search for the pattern we want to replace
x <- seekr("old_name", replacement = "new_name", path = project_dir)
x

# No backup exists yet
list_backups(backup_dir = backup_dir)

# Apply the replacement; a backup is created automatically
replace_files(x, backup_dir = backup_dir)

# The backup now contains the original (pre-replacement) version
# of both files, under id = 1
list_backups(backup_dir = backup_dir)

# The files on disk have changed
readLines(file1)
readLines(file2)

# Searching for the original pattern no longer finds anything
seekr("old_name", path = project_dir)

# Restore the files from the backup; this itself creates a second
# backup (id = 2) of the current (replaced) version, before overwriting
b <- last_backup(backup_dir = backup_dir)
restore_files(from = b$backup, to = b$original, backup_dir = backup_dir)

# We now have two backups: id = 1 is the state before replace_files(),
# id = 2 is the state before restore_files() (i.e. the replaced version)
list_backups(backup_dir = backup_dir)

# The files are back to their original content
readLines(file1)
readLines(file2)

# The original pattern is found again
seekr("old_name", path = project_dir)

## Not run:
# open_backup_dir() opens a file browser, so it is not run here
open_backup_dir()

# Review each file interactively before restoring
restore_files_interactive(from = b$backup, to = b$original)

# Customize the diff display (unified format)
restore_files_interactive(
  from = b$backup,
  to = b$original,
  mode = "unified",
```

```
  color.mode = "yb"
)

# Finally, particular backups in a backup_dir can be deleted
b <- list_backups(backup_dir = backup_dir)
b
delete_backups(id = b$id, backup_dir = backup_dir)

## End(Not run)


unlink(project_dir, recursive = TRUE)
unlink(backup_dir, recursive = TRUE)
```

`replace_text` *Replace selected matches in text*

Description

`replace_text()` is the in-memory counterpart of `replace_files()`. It applies the replacements stored in a `seekr_match` object to text and returns the modified text.

It does not read files, write files, or create backups. Use `replace_files()` for the usual file-based workflow.

Usage

```
replace_text(text, x)
```

Arguments

`text`	Text content as a single string.
`x`	A `seekr_match` object with replacement values. All matches in `x` must be associated with the same file and must refer to positions in `text`.

Details

`replace_text()` verifies that the current text has the same hash as the text that was searched when the matches were created. If the text has changed, replacement is considered unsafe and the function aborts.

Matches are replaced from the end of the file to the beginning, so earlier replacements do not shift the recorded positions of later replacements.

`replace_text()` requires `x` to contain matches from a single source, the one corresponding to text.

Value

A single character string containing `text` after applying the replacements stored in `x`.

See Also

- match_text() to create matches from already-read text.
- replace_files() to apply replacements directly to files.
- filter_match() to keep only some matches before replacing.

Examples

```
text <- "hello old_name\nbye old_name"

x <- match_text(
  text = text,
  path = "example.txt",
  pattern = "old_name",
  replacement = "new_name"
)

replace_text(text, x)
```

---

restore_files *Restore files from backups*

Description

restore_files() copies files from a backup location back to their original paths, overwriting the current versions. By default, it creates a new backup of the current files before overwriting, so the restore itself can be undone.

restore_files_interactive() works the same way but shows a side-by-side diff (powered by diffobj::diffFile()) for each file before asking whether to restore it. This is useful when you want to review the difference between the current version of the file and its backup.

The typical workflow is to call list_backups() or last_backup() to find the backup you want, then pass the backup and original columns to restore_files():

```
b <- last_backup()
restore_files(from = b$backup, to = b$original)
```

Usage

```
restore_files(
  from,
  to,
  ...,
  backup = TRUE,
  description = NA_character_,
  backup_dir = seekr_option("seekr.backup_dir"),
  .progress = seekr_option("seekr.progress")
```

```
)

restore_files_interactive(
  from,
  to,
  ...,
  backup = TRUE,
  description = NA_character_,
  backup_dir = seekr_option("seekr.backup_dir"),
  .progress = seekr_option("seekr.progress")
)
```

Arguments

`from`	Character vector of source file paths to copy from. Typically the `backup` column of `list_backups()` or `last_backup()`. All files must exist.
`to`	Character vector of destination file paths to copy to. Typically the `original` column of `list_backups()` or `last_backup()`. Must be the same length as `from`, with no duplicates. Destination files do not need to exist.
`...`	For `restore_files()`: these dots are for future extensions and must be empty. For `restore_files_interactive()`: additional arguments passed to `diffobj::diffFile()`, allowing you to customize the diff display. For example, `mode = "unified"` switches to a unified diff format.
`backup`	Whether to create a backup of the current files before overwriting the files. Default is `TRUE`. Set to `FALSE` to skip the backup, for example if the files are already tracked by another version control system.
`description`	Optional character string stored in the backup metadata when `backup = TRUE`. Use it to describe why the backup was created. Defaults to `NA`.
`backup_dir`	Path to the backup directory. Defaults to `seekr_option("seekr.backup_dir")`, which resolves to a platform-specific user data directory. Change the default globally with `options(seekr.backup_dir = "/your/path")`.
`.progress`	Whether to display progress messages. Default is `TRUE` in interactive sessions and `FALSE` otherwise (see `rlang::is_interactive()`). Can be set globally with `options(seekr.progress = FALSE)`.

Value

Both functions invisibly return the `to` paths of the files that were actually restored. For `restore_files()` this is always the full `to` vector. For `restore_files_interactive()` this is only the subset of files the user chose to restore, an empty character vector if the user cancelled or skipped all files.

Interactive restore

`restore_files_interactive()` iterates over each file and shows a diff between the backup (`from`) and the current version (`to`). For each file, it prompts with the following choices:

- **Restore this file**: restore this file and move to the next.

- **Ignore this file**: skip this file and move to the next.
- **Restore all remaining files**: restore this file and all subsequent ones without further prompts.
- **Ignore all remaining files**: skip this file and all subsequent ones.
- **Cancel all planned changes**: abort immediately without restoring anything, including files already confirmed in this session.

No files are modified until after the last prompt. All choices are collected first, then applied in a single `restore_files()` call.

`restore_files_interactive()` requires the diffobj package. It can only be called in an interactive session.

See Also

- `list_backups()` and `last_backup()` to find the backup you want to restore.
- `delete_backups()` to remove backups you no longer need.
- `replace_files()` which creates a backup before writing any changes.

Examples

```
# Set up a temporary project with two files
project_dir <- tempfile("seekr_project")
dir.create(project_dir)

backup_dir <- tempfile("seekr_backups")
dir.create(backup_dir)

file1 <- file.path(project_dir, "script1.R")
file2 <- file.path(project_dir, "script2.R")
writeLines("old_name <- function(x) x + 1", file1)
writeLines("y <- old_name(2)", file2)

# Search for the pattern we want to replace
x <- seekr("old_name", replacement = "new_name", path = project_dir)
x

# No backup exists yet
list_backups(backup_dir = backup_dir)

# Apply the replacement; a backup is created automatically
replace_files(x, backup_dir = backup_dir)

# The backup now contains the original (pre-replacement) version
# of both files, under id = 1
list_backups(backup_dir = backup_dir)

# The files on disk have changed
readLines(file1)
readLines(file2)

# Searching for the original pattern no longer finds anything
```

```
seekr("old_name", path = project_dir)

# Restore the files from the backup; this itself creates a second
# backup (id = 2) of the current (replaced) version, before overwriting
b <- last_backup(backup_dir = backup_dir)
restore_files(from = b$backup, to = b$original, backup_dir = backup_dir)

# We now have two backups: id = 1 is the state before replace_files(),
# id = 2 is the state before restore_files() (i.e. the replaced version)
list_backups(backup_dir = backup_dir)

# The files are back to their original content
readLines(file1)
readLines(file2)

# The original pattern is found again
seekr("old_name", path = project_dir)

## Not run:
# open_backup_dir() opens a file browser, so it is not run here
open_backup_dir()

# Review each file interactively before restoring
restore_files_interactive(from = b$backup, to = b$original)

# Customize the diff display (unified format)
restore_files_interactive(
  from = b$backup,
  to = b$original,
  mode = "unified",
  color.mode = "yb"
)

# Finally, particular backups in a backup_dir can be deleted
b <- list_backups(backup_dir = backup_dir)
b
delete_backups(id = b$id, backup_dir = backup_dir)

## End(Not run)


unlink(project_dir, recursive = TRUE)
unlink(backup_dir, recursive = TRUE)
```

---

seek *Find matches in text files*

Description

seek() searches text files for a pattern and returns a seekr_match vector. The result can be inspected, filtered, and passed to replace_files() to apply replacements.

seekr() is a convenience wrapper around seek() that restricts the search to R, R Markdown, and Quarto files (.R, .Rmd, .qmd).

list_files(), filter_files(), and match_files() are the three building blocks of seek(). They can be called individually when you need more control over each step.

Steps:

- list_files() starts from path, optionally use_git to restrict file discovery, and recurses into subdirectories to list files. By default, not all files are listed, with hidden files and directories excluded.
- filter_files() keeps files matching extension and path_pattern and not exceeding max_file_size. Finally, the exclude functions are applied to the remaining files, discarding common non-text or irrelevant files by default.
- match_files() reads each file, decodes them using encoding, finds pattern matches, and captures surrounding context lines. A replacement can be provided to stage changes for later application with replace_files().

Usage

```
seek(
  pattern,
  replacement = NULL,
  ...,
  path = ".",
  recurse = TRUE,
  all = FALSE,
  use_git = FALSE,
  extension = NULL,
  path_pattern = NULL,
  max_file_size = Inf,
  exclude = seekr::exclude_functions,
  context = 5L,
  encoding = "UTF-8",
  .progress = seekr_option("seekr.progress")
)

seekr(
  pattern,
  replacement = NULL,
  ...,
  path = ".",
  recurse = TRUE,
  all = FALSE,
  use_git = FALSE,
  path_pattern = NULL,
```

```
  max_file_size = Inf,
  exclude = seekr::exclude_functions,
  context = 5L,
  encoding = "UTF-8",
  .progress = seekr_option("seekr.progress")
)
```

Arguments

pattern	Pattern to search for, matched using stringr (ICU regular expressions). Either: • A string, automatically wrapped as `stringr::regex()` with `ignore_case = FALSE`, `multiline = TRUE`, `comments = FALSE`, and `dotall = FALSE`. • A `stringr_pattern` object such as `stringr::regex()`, `stringr::fixed()`, or `stringr::coll()`, used as-is for more control.
replacement	Replacement to associate with each match. Replacements are computed immediately during the search and stored in the result. Either: • `NULL` (default): no replacement. `replace_files()` cannot be called without setting replacements first. • A plain string, used literally as replacement text. • A string with backreferences of the form `\1`, `\2`, etc., replaced with the corresponding capture group from `pattern`. • A function, called once per file with a character vector of all matches found in that file, and expected to return a character vector of the same length (e.g. toupper). • A function wrapped with `with_capture_groups_matrix()`, called once per file with a character matrix where the first column is the full match and the remaining columns are the capture groups.
...	These dots are for future extensions and must be empty.
path	A character vector of one or more existing directories to search in. Defaults to `"."` (the current working directory).
recurse	Controls how deep the directory traversal goes to list files. Either: • `TRUE` (default): recurse into all subdirectories. • `FALSE`: only list files at the top level of each directory in `path`. • A positive integer: limit recursion to that many levels deep.
all	Whether to list hidden files and directories. Default is `FALSE`.
use_git	Should Git be used to restrict file discovery inside Git repositories? If `TRUE`, `list_files()` keeps only files that were first discovered according to `path`, `recurse`, and `all`, and are also returned by `git ls-files --cached --others --exclude-standard`. `use_git = TRUE` looks for the Git root by walking upward from each supplied `path`, but it does not recursively search downward for Git repositories in subdirectories. Git must be installed and available on `PATH`.
extension	Optional character vector of file extensions to keep. Either: • `NULL` (default): no filtering by extension; all extensions are kept. • A character vector of extensions to keep, with or without a leading dot (e.g. `c("R", ".Rmd", "qmd")`).

Extensions are normalized before matching: leading dots are stripped, matching is case-insensitive, and duplicates are ignored. Only the last component of compound extensions is used (e.g. `"tar.gz"` uses `"gz"`), with a warning.

`path_pattern` Optional pattern applied to filter normalized file paths (see `as_seekr_path()`). Either:

- `NULL` (default): no filtering by path.
- A string, interpreted as a regular expression via `stringr::regex()`.
- A `stringr_pattern` object such as `stringr::regex()` or `stringr::fixed()`.

`max_file_size` Maximum file size in bytes. Files larger than this value are excluded. Default is `Inf`, meaning no files are excluded by size. Zero and negative values are treated as `Inf`.

`exclude` Named list of functions used to exclude unwanted files during filtering. Either:

- `NULL`, to disable additional exclude functions.
- A named list of functions, each taking a character vector of normalized file paths and returning a logical vector of the same length, where `TRUE` means the file should be excluded.

Defaults to exclude_functions, which excludes common non-text or irrelevant files.

`context` Number of surrounding lines to capture around each match. Either:

- A single non-negative integer (default: `5L`): captures the same number of lines before and after each match.
- A pair of non-negative integers `c(before, after)`: captures `before` lines before and `after` lines after each match.

`encoding` Encoding used to decode file content during the matching step. Either:

- A single string (default: `"UTF-8"`), applied to all files.
- `NULL`: encoding is guessed for each file individually using `stringi::stri_enc_detect()`, falling back to `"UTF-8"` when detection fails.

Note: `replace_files()` always writes files in UTF-8. A warning is issued once per session when any file is read with a non-UTF-8 encoding. By default, `replace_files()` refuses to write those matches unless `allow_encoding_change = TRUE` is set.

`.progress` Whether to display progress messages. Default is `TRUE` in interactive sessions and `FALSE` otherwise (see `rlang::is_interactive()`). Can be set globally with `options(seekr.progress = FALSE)`.

Value

A `seekr_match` vector. Each element represents one match and carries the file path, match position, matched text, optional replacement, context lines, encoding, and a hash of the searched text used for replacement safety. The vector is always returned, even when empty.

An attribute `"exclusions"` is attached to the result after filtering, containing a data frame with one row per input file and one column per exclusion function, detailing which files were excluded and why. Retrieve it with `exclusions()`.

If the result is empty, use `empty_stage()` to see whether the pipeline became empty during input, listing, filtering, or matching.

See Also

- seekr_match for the match object structure and available methods.
- `print.seekr_match()` and `summary.seekr_match()` to inspect results.
- `filter_match()` to subset matches.
- `replace_files()` to apply replacements.

Examples

```
# Create a small temporary project to search in
example_dir <- tempfile("seekr-example")
dir.create(example_dir)
dir.create(file.path(example_dir, "R"))
dir.create(file.path(example_dir, "tests"))
dir.create(file.path(example_dir, "data"))

writeLines(
  c(
    "old_fn <- function(x) {",
    "  # TODO: rename foo",
    "  foo + x",
    "}"
  ),
  file.path(example_dir, "R", "code.R")
)

writeLines(
  c(
    "test_that('foo works', {",
    "  # TODO: update test",
    "  expect_equal(foo, 1)",
    "})"
  ),
  file.path(example_dir, "tests", "test-code.R")
)

writeLines(
  c(
    "name,value",
    "foo,1",
    "bar,2"
  ),
  file.path(example_dir, "data", "values.csv")
)

# seek() is built from three lower-level functions
files <- list_files(example_dir)
filtered <- filter_files(files, extension = "R", path_pattern = "/R/")
x <- match_files(filtered, "foo", "bar")

# These functions can be piped
y <-
```

```
  example_dir |>
  list_files() |>
  filter_files(extension = "R", path_pattern = "/R/") |>
  match_files("foo", "bar")

identical(x, y)

# This is equivalent to the seek() call below
z <- seek("foo", "bar", path = example_dir, extension = "R", path_pattern = "/R/")
identical(y, z)

# Search for a pattern in all text files
x <- seek("TODO", path = example_dir)
print(x)

# Search only in R files
seek("TODO", path = example_dir, extension = "R")

# Search only in a specific subfolder
seek("TODO", path = example_dir, path_pattern = "/R/")

# seekr() is a shortcut for searching R, R Markdown, and Quarto files
seekr("old_fn", path = example_dir)

# Stage a plain string replacement
x <- seek("old_fn", "new_fn", path = example_dir)
x

# Stage replacements with a function
x <- seek(
  "foo|bar",
  replacement = function(x) ifelse(x == "foo", "bar", "foo"),
  path = example_dir
)
x

# Stage replacements after searching
x <- seekr("foo|bar", path = example_dir)
field(x, "replacement") <- ifelse(field(x, "match") == "foo", "bar", "foo")
x

# Create a temporary backup directory
backup_dir <- tempfile("seekr-backup")
dir.create(backup_dir)

# Apply replacements after inspection
replace_files(x, backup_dir = backup_dir)

# Restore files from the latest backup
bck <- last_backup(backup_dir = backup_dir)
restore_files(from = bck$backup, to = bck$original, backup_dir = backup_dir)

# See which files were excluded
```

```
exclusions(x)

# empty_stage() explains where the pipeline became empty
dir.create(file.path(example_dir, "empty"))

empty_stage(seek("foo", path = character()))
empty_stage(seek("foo", path = file.path(example_dir, "empty")))
empty_stage(seek("foo", path = example_dir, extension = "dummy"))
empty_stage(seek("missing_pattern", path = example_dir))

# Remove the two temporary directories
unlink(backup_dir, recursive = TRUE)
unlink(example_dir, recursive = TRUE)
```

---

seekr_match *Create* seekr_match *vectors*

Description

A seekr_match is an S3 vector built on vctrs::new_rcrd() that represents the matches found by seek(), seekr(), match_files(), or match_text().

Each element corresponds to a single match and stores its source path, position, matched text, optional replacement, surrounding context lines, encoding, and a hash of the searched text used for replacement safety.

Usage

```
new_seekr_match(
  path = character(),
  start_line = integer(),
  end_line = integer(),
  start = integer(),
  end = integer(),
  start_col = integer(),
  end_col = integer(),
  match = character(),
  replacement = character(),
  before = character(),
  line = character(),
  after = character(),
  encoding = character(),
  hash = character()
)
```

Arguments

path	A character vector of source identifiers. For file workflows, these are normalized absolute file paths. For match_text() workflows, they may also be non-existing identifiers.
start_line, end_line	Integer vectors. 1-based line numbers where each match begins and ends.
start, end	Integer vectors. 1-based absolute character positions where each match begins and ends.
start_col, end_col	Integer vectors. 1-based column positions of the match start and end within their respective lines.
match	A character vector. The exact text matched.
replacement	A character vector. The staged replacement for each match. NA indicates no replacement is staged.
before	A character vector. Context lines preceding each match.
line	A character vector. The complete line(s) containing each match.
after	A character vector. Context lines following each match.
encoding	A character vector. The encoding used to read each file.
hash	A character vector. Hash of the searched text, used to check that it has not changed before replacement.

Value

new_seekr_match() returns an empty or populated seekr_match vector. new_seekr_match() is a low-level constructor intended primarily for internal use and advanced extensions. In normal usage, create seekr_match vectors with seek(), seekr(), match_files(), or match_text().

Fields

Access any field with vctrs::field() which is re-exported by seekr:

```
field(x, "path")
field(x, "match")
```

Methods and functions

seekr_match objects support the following S3 methods:

- print(): displays matches in a formatted view with optional context and replacement preview.
- summary(): summarizes matches by file, matched text, replacement, extension, and encoding.
- str(): shows the internal field structure with types and sample values.
- tibble::as_tibble(): converts to a tibble for advanced manipulation.

The following functions are also commonly used with seekr_match vectors:

- as_match(): converts a tibble back to a seekr_match vector.
- filter_match(): subsets matches using dplyr-style expressions.
- sort_within_files(): reorders matches within each file while preserving file order.

Attributes

In addition to its fields, a `seekr_match` vector returned by `seek()` or `seekr()` may carry two attributes:

- `empty_stage`: when the vector is empty, indicates which pipeline step produced no output. Retrieve with `empty_stage()`.
- `exclusions`: when files were removed during filtering, a data frame detailing which files were excluded and by which function. Retrieve with `exclusions()`.

These attributes are dropped when combining `seekr_match` vectors.

See Also

- `seek()` and `seekr()` to search for matches and produce `seekr_match` vectors.
- `print.seekr_match()` and `summary.seekr_match()` to inspect results.
- `filter_match()` to subset matches before replacing.
- `replace_files()` to apply staged replacements.
- `empty_stage()` and `exclusions()` to diagnose empty results.

Examples

```
# Produce a seekr_match vector
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("function", toupper, path = ext_path)

# Access a field
field(x, "path")
field(x, "match")

# Subset
head(x, 3)

# Combine two seekr_match vectors
y <- seekr("Hello", path = ext_path)
c(x, y)
```

`seekr_option` *Retrieve a seekr option*

Description

`seekr_option()` returns the resolved value of a single seekr option.

It first checks whether the user has set the option with `base::options()`. If not, it returns the package default. The returned value is validated before being returned.

This function is mostly useful for seekr's own default arguments, such as:

```
.progress = seekr_option("seekr.progress")
```

In most cases, users should configure seekr with `base::options()` and inspect available options with `seekr_options()`.

Usage

```
seekr_option(name)
```

Arguments

`name`	Name of the seekr option to retrieve, as a single string. See `seekr_options()` for the list of valid names.

Value

The resolved option value. The returned type depends on the option:

- `logical` for `seekr.progress`.
- `character` for all other options (paths, print symbols, and ANSI style codes).

See Also

`seekr_options()` to list all available options and their defaults.

Examples

```
seekr_option("seekr.progress")
seekr_option("seekr.print.mode")

options(seekr.print.mode = "plain")
seekr_option("seekr.print.mode")

options(seekr.print.mode = NULL)
seekr_option("seekr.print.mode")
```

---

seekr_options *List seekr options*

Description

`seekr_options()` returns the options that control seekr's global behavior, along with their current user-defined value and their package default.

These options can be changed with `base::options()`. For example:

```
options(seekr.progress = FALSE)
options(seekr.print.mode = "plain")
```

Option values:

The `current` column reports the value currently set with `base::options()`. If an option has not been set by the user, `current` is `NA` and seekr falls back to the value shown in `default`.

Available options:

The main options are:

- `seekr.progress`: whether seekr displays progress messages by default.
- `seekr.backup_dir`: directory where backups are stored.
- `seekr.print.mode`: print mode, either `"rich"`, `"color"`, or `"plain"`.
- `seekr.print.tab`: symbol used to display tab characters.
- `seekr.print.newline`: symbol used to display newline characters when printing deleted newlines.
- `seekr.style.*`: ANSI style codes used internally by rich printing.

The `seekr.style.*` options are intentionally low-level. They accept ANSI SGR codes as strings, such as `"31"`, `"1;31"`, or `"38;5;243"`.

Usage

```
seekr_options()
```

Value

A tibble with one row per seekr option and the following columns:

- `name`: option name.
- `current`: value currently set by the user, or `NA` if unset.
- `default`: default value used by seekr when the option is unset.

See Also

`seekr_option()` to retrieve the resolved value of a single option.

Examples

```
seekr_options()

# Disable progress messages globally
options(seekr.progress = FALSE)
seekr_options()

# Reset the option
options(seekr.progress = NULL)
```

sort_within_files *Sort matches within each file*

Description

`sort_within_files()` sorts matches by position within each file, while preserving the order in which files appear in `x`.

This differs from `sort()`, which sorts globally and reorders files alphabetically. Use `sort_within_files()` when you want positions within each file to be consistent — for example, after combining two `seekr_match` vectors with `vctrs::vec_c()` — without changing the order in which files are displayed or processed.

Usage

```
sort_within_files(x)
```

Arguments

x	A `seekr_match` vector.

Value

A `seekr_match` vector with the same matches in potentially different order: files appear in their original order, and matches within each file are sorted by `start`, then `end`, then `match`, then `replacement`.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("TODO", path = ext_path)
y <- seekr("FIXME", path = ext_path)

# Combine and reorder within files without changing file order
z <- vctrs::vec_c(x, y)
sort_within_files(z)
```

str.seekr_match *Inspect the structure of a* `seekr_match` *vector*

Description

`str()` displays the internal structure of a `seekr_match` vector: the name, type, and sample values of each field, formatted for the console width.

This is useful for a quick overview of what a `seekr_match` vector contains without printing the full formatted output produced by `print.seekr_match()`.

Usage

```
## S3 method for class 'seekr_match'
str(object, ...)
```

Arguments

object	A seekr_match vector.
...	Not used. Present for compatibility with the str() generic.

Value

Invisibly returns the seekr_match vector.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("TODO", path = ext_path)
str(x)
```

---

summary.seekr_match *Summarize matches and planned replacements*

Description

summary() produces a compact overview of a seekr_match vector: the most frequent files, matched texts, file extensions, and encodings.

When replacements are staged, matched texts are displayed together with their replacement preview, giving a high-level picture of what replace_files() would change.

Usage

```
## S3 method for class 'seekr_match'
summary(object, ...)

## S3 method for class 'summary_seekr_match'
print(x, ..., n = NULL)
```

Arguments

object	A seekr_match vector.
...	Not used. Present for compatibility with the summary() generic.
x	A summary_seekr_match object, as returned by summary.seekr_match().
n	Maximum number of rows to print in each summary table. If NULL, a compact default is used. This limit is applied separately to each section of the summary, such as top files, top matches/replacements, top extensions, and top encodings.

Value

An object of class `summary_seekr_match`, containing summary tables for files, matches/replacements, extensions, and encodings. Print it with `print()` to display a formatted summary in the console.

See Also

- `print.seekr_match()` for a full match-level display with context lines.
- `str.seekr_match()` for the internal field structure.

Examples

```
ext_path <- system.file("extdata", package = "seekr")
x <- seekr("TODO", path = ext_path)
y <- seekr("TODO", "DONE", path = ext_path)
summary(x)
summary(y)
summary(c(x, y))
```

---

with_capture_groups_matrix

Use capture groups in function-based replacements

Description

This helper wraps a function to indicate that it should receive the full capture group matrix as input, instead of the default vector of full matches. This allows the replacement logic to use individual capture groups.

Usage

```
with_capture_groups_matrix(fn)
```

Arguments

`fn`	A function taking a single argument: a character matrix of capture groups, where each row corresponds to a match, the first column is the full match and subsequent columns are capture groups.

Details

The capture matrix is the result of `stringr::str_match_all()`: the first column contains the full match, and subsequent columns contain capture groups.

Value

A function identical to `fn`, but marked with an internal attribute used by `compute_replacement()` to dispatch on replacement logic.

Examples

```
text <- "lorem ipsum foo_bar lorem ipsum bar_foo lorem ipsum"
fn_repl <- function(M) paste0(tolower(M[, 3L]), ".", toupper(M[, 2L]))
fn_repl <- with_capture_groups_matrix(fn_repl)
match_text(text, path = "example", pattern = "(\\w+)_(\\w+)", replacement = fn_repl)
```

Index

as.data.frame.seekr_match
(as_tibble.seekr_match), 3
as_match (as_tibble.seekr_match), 3
as_match(), *14*, *36*
as_seekr_path, 2
as_seekr_path(), *12*, *13*, *15*, *32*
as_tibble.seekr_match, 3
as_tibble.seekr_match(), *14*

backups, 5
base::options(), *37–39*

compute_replacement(), *42*

delete_backups (backups), 5
delete_backups(), *28*
diffobj::diffFile(), *26*, *27*
dplyr::filter(), *3*, *14*
dplyr::mutate(), *3*

empty_stage, 8
empty_stage(), *32*, *37*
exclude_functions, 9, *10–12*, *32*
exclusions, 10
exclusions(), *11*, *13*, *32*, *37*

filter_files, 11
filter_files(), *2*, *3*, *9–11*, *15–18*, *30*
filter_match, 13
filter_match(), *4*, *21*, *23*, *26*, *33*, *36*, *37*

is_dependency_dir (exclude_functions), 9
is_git_dir (exclude_functions), 9
is_minified_file (exclude_functions), 9
is_not_text_mime (exclude_functions), 9

last_backup (backups), 5
last_backup(), *23*, *26–28*
list_backups (backups), 5
list_backups(), *23*, *26–28*
list_files, 15
list_files(), *3*, *8*, *11*, *13*, *15–17*, *30*, *31*

match_files, 17
match_files(), *2*, *3*, *11*, *13*, *19*, *20*, *22*, *30*,
35, *36*
match_text, 19
match_text(), *18*, *22*, *26*, *35*, *36*

new_seekr_match (seekr_match), 35

open_backup_dir (backups), 5

print(), *36*
print.seekr_match, 20
print.seekr_match(), *33*, *37*, *40*, *42*
print.summary_seekr_match
(summary.seekr_match), 41

rappdirs::user_data_dir(), *6*
replace_files, 22
replace_files(), *3–7*, *14*, *17–20*, *22*, *25*, *26*,
28, *30–33*, *37*, *41*
replace_text, 25
replace_text(), *20*, *23*
restore_files, 26
restore_files(), *5–7*, *23*
restore_files_interactive
(restore_files), 26
restore_files_interactive(), *7*, *23*
rlang::is_interactive(), *6*, *12*, *16*, *18*, *22*,
27, *32*

seek, 29
seek(), *8–11*, *13*, *15–17*, *20–23*, *30*, *35–37*
seekr (seek), 29
seekr(), *8–11*, *20*, *22*, *23*, *35–37*
seekr_match, *3*, *4*, *8*, *9*, *14*, *18–23*, *25*, *30*, *32*,
33, 35, *40*, *41*
seekr_option, 37
seekr_option(), *39*
seekr_options, 38

seekr_options(), *38*
sort_within_files, 40
sort_within_files(), *36*
str(), *36*
str.seekr_match, 40
str.seekr_match(), *42*
stringi::stri_enc_detect(), *18*, *32*
stringr, *17*, *19*, *31*
stringr::coll(), *17*, *19*, *31*
stringr::fixed(), *12*, *17*, *19*, *31*, *32*
stringr::regex(), *12*, *17*, *19*, *31*, *32*
stringr::str_match_all(), *42*
summary(), *21*, *36*
summary.seekr_match, 41
summary.seekr_match(), *33*, *37*, *41*

tibble::as_tibble(), *36*
toupper, *18*, *20*, *31*

vctrs::field(), *4*, *14*, *36*
vctrs::new_rcrd(), *35*
vctrs::vec_c(), *40*

with_capture_groups_matrix, 42
with_capture_groups_matrix(), *18*, *20*, *31*