

# Package ‘grepreaper’

January 23, 2026

**Title** Efficient Data Filtering and Aggregation Using Grep

**Version** 0.1.0

**Description** Provides an interface to the system-level 'grep' utility for efficiently reading, filtering, and aggregating data from multiple flat files. By pre-filtering data at the command line before it enters the R environment, the package reduces memory overhead and improves ingestion speed. Includes functions for counting records across large file systems and supports recursive directory searching.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Suggests** ggplot2, knitr, rmarkdown

**VignetteBuilder** knitr

**Imports** data.table, methods

**NeedsCompilation** no

**Author** David Shilane [aut],  
Atharv Raskar [aut],  
Akshat Maurya [aut, cre]

**Maintainer** Akshat Maurya <codingmaster902@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-01-23 21:10:02 UTC

## Contents

build_grep_cmd . . . . .	2
grep_count . . . . .	2
grep_read . . . . .	3
split_columns . . . . .	5

<b>Index</b>	<b>7</b>
--------------	----------

---

build_grep_cmd	<i>Build grep command string</i>
----------------	----------------------------------

---

### Description

Constructs a safe and properly formatted grep command string for system execution. This function handles input sanitization by utilizing R's internal shell quoting mechanism, ensuring compatibility across different operating systems.

### Usage

```
build_grep_cmd(pattern, files, options = "", fixed = FALSE)
```

### Arguments

pattern	Character vector of patterns to search for.
files	Character vector of file paths to search in.
options	Character string containing grep flags (e.g., "-i", "-v").
fixed	Logical; if TRUE, grep is told to treat patterns as fixed strings.

### Value

A properly formatted command string ready for system execution.

---

grep_count	<i>grep_count: Efficiently count the number of relevant records from one or more files using grep</i>
------------	---

---

### Description

grep\_count: Efficiently count the number of relevant records from one or more files using grep

### Usage

```
grep_count(
  files = NULL,
  path = NULL,
  file_pattern = NULL,
  pattern = "",
  invert = FALSE,
  ignore_case = FALSE,
  fixed = FALSE,
  recursive = FALSE,
  word_match = FALSE,
  only_matching = FALSE,
```

```

    skip = 0,
    header = TRUE,
    include_filename = FALSE,
    show_cmd = FALSE,
    show_progress = FALSE,
    ...
  )

```

### Arguments

files	Character vector of file paths to read.
path	Optional. Directory path to search for files.
file_pattern	Optional. A pattern to filter filenames when using the path argument. Passed to <code>list.files</code> .
pattern	Pattern to search for within files (passed to <code>grep</code> ).
invert	Logical; if TRUE, return non-matching lines.
ignore_case	Logical; if TRUE, perform case-insensitive matching (default: TRUE).
fixed	Logical; if TRUE, pattern is a fixed string, not a regular expression.
recursive	Logical; if TRUE, search recursively through directories.
word_match	Logical; if TRUE, match only whole words.
only_matching	Logical; if TRUE, return only the matching part of the lines.
skip	Integer; number of rows to skip.
header	Logical; if TRUE, treat first row as header.
include_filename	Logical; if TRUE, include source filename as a column.
show_cmd	Logical; if TRUE, return the grep command string instead of executing it.
show_progress	Logical; if TRUE, show progress indicators.
...	Additional arguments passed to <code>fread</code> .

### Value

A `data.table` containing file names and counts.

---

grep_read	<i>grep_read: Efficiently read and filter lines from one or more files using grep, returning a data.table.</i>
-----------	--

---

### Description

`grep_read`: Efficiently read and filter lines from one or more files using `grep`, returning a `data.table`.

**Usage**

```

grep_read(
  files = NULL,
  path = NULL,
  file_pattern = NULL,
  pattern = "",
  invert = FALSE,
  ignore_case = FALSE,
  fixed = FALSE,
  show_cmd = FALSE,
  recursive = FALSE,
  word_match = FALSE,
  show_line_numbers = FALSE,
  only_matching = FALSE,
  nrows = Inf,
  skip = 0,
  header = TRUE,
  col.names = NULL,
  include_filename = FALSE,
  show_progress = FALSE,
  ...
)

```

**Arguments**

<code>files</code>	Character vector of file paths to read.
<code>path</code>	Optional. Directory path to search for files.
<code>file_pattern</code>	Optional. A pattern to filter filenames when using the path argument. Passed to <code>list.files</code> .
<code>pattern</code>	Pattern to search for within files (passed to <code>grep</code> ).
<code>invert</code>	Logical; if TRUE, return non-matching lines.
<code>ignore_case</code>	Logical; if TRUE, perform case-insensitive matching (default: TRUE).
<code>fixed</code>	Logical; if TRUE, pattern is a fixed string, not a regular expression.
<code>show_cmd</code>	Logical; if TRUE, return the grep command string instead of executing it.
<code>recursive</code>	Logical; if TRUE, search recursively through directories.
<code>word_match</code>	Logical; if TRUE, match only whole words.
<code>show_line_numbers</code>	Logical; if TRUE, include line numbers from source files. Headers are automatically removed and lines renumbered.
<code>only_matching</code>	Logical; if TRUE, return only the matching part of the lines.
<code>nrows</code>	Integer; maximum number of rows to read.
<code>skip</code>	Integer; number of rows to skip.
<code>header</code>	Logical; if TRUE, treat first row as header. Note that using FALSE means that the first row will be included as a row of data in the reading process.

col.names	Character vector of column names.
include_filename	Logical; if TRUE, include source filename as a column.
show_progress	Logical; if TRUE, show progress indicators.
...	Additional arguments passed to fread.

**Value**

A data.table with different structures based on the options:

- Default: Data columns with original types preserved
- show\_line\_numbers=TRUE: Additional 'line\_number' column (integer) with source file line numbers
- include\_filename=TRUE: Additional 'source\_file' column (character)
- only\_matching=TRUE: Single 'match' column with matched substrings
- show\_cmd=TRUE: Character string containing the grep command

**Note**

When searching for literal strings (not regex patterns), set fixed = TRUE to avoid regex interpretation. For example, searching for "3.94" with fixed = FALSE will match "3894" because "." is a regex metacharacter.

Header rows are automatically handled:

- With show\_line\_numbers=TRUE: Headers (line\_number=1) are removed and lines renumbered
- Without line numbers: Headers matching column names are removed
- Empty rows and all-NA rows are automatically filtered out

---

split_columns	<i>Split columns based on a delimiter</i>
---------------	---

---

**Description**

Efficiently splits character vectors into multiple columns based on a specified delimiter. This function is optimized for performance and handles common use cases like parsing grep output or other delimited text data.

**Usage**

```
split_columns(
  x,
  column.names = NA,
  split = ":",
  resulting.columns = 3,
  fixed = TRUE
)
```

**Arguments**

<code>x</code>	Character vector to split
<code>column.names</code>	Names for the resulting columns (optional)
<code>split</code>	Delimiter to split on (default: ":")
<code>resulting.columns</code>	Number of columns to create (default: 3)
<code>fixed</code>	Whether to use fixed string matching (default: TRUE)

**Value**

A data.table with split columns. Column names are automatically assigned as V1, V2, V3, etc. unless custom names are provided via `column.names`.

**Examples**

```
# Split grep-like output with colon delimiter
data <- c("file.txt:15:error message", "file.txt:23:warning message")
result <- split_columns(data, resulting.columns = 3)
print(result)

# With custom column names
result_named <- split_columns(data,
                              column.names = c("filename", "line", "message"),
                              resulting.columns = 3)

print(result_named)

# Split into 2 columns (combining remaining elements)
result_2col <- split_columns(data, resulting.columns = 2)
print(result_2col)
```

# Index

`build_grep_cmd`, [2](#)

`grep_count`, [2](#)

`grep_read`, [3](#)

`split_columns`, [5](#)