

Package ‘funspotr’

January 29, 2026

Title Spot R Functions & Packages

Version 0.0.5

Description Helpers for parsing out the R functions
and packages used in R scripts and notebooks.

License MIT + file LICENSE

BugReports <https://github.com/brshallo/funspotr/issues>

Imports dplyr (>= 0.8.3), tidyr, purrr, stringr, glue, knitr, httr,
callr, readr, here, formatR, fs, tibble, utils, import (>=
1.3.0), lifecycle

Suggests remotes, visNetwork, igraph

Enhances blogdown

Encoding UTF-8

RoxygenNote 7.3.3

URL <https://brshallo.github.io/funspotr/>

Language en-US

NeedsCompilation no

Author Bryan Shalloway [aut, cre]

Maintainer Bryan Shalloway <brshallo.dev@gmail.com>

Repository CRAN

Date/Publication 2026-01-29 06:00:01 UTC

Contents

check_pkgs_availability	2
list_files_github_repo	3
list_files_wd	4
network_plot	5
spot_funs	6
spot_funs_custom	7
spot_funs_files	9

spot_pkgs	11
spot_pkgs_used	12
spot_tags	13
unnest_results	15
Index	16

check_pkgs_availability
Check Packages Availability

Description

Check whether packages are available in current library.

Usage

```
check_pkgs_availability(pkgs, quietly = TRUE)
```

Arguments

pkgs	Character vector of package names. (Typically the output from <code>spot_pkgs()</code>).
quietly	logical: should progress and error messages be suppressed?

Value

Named logical vector indicating whether each package is available on the machine.

Examples

```
library(funspotr)
library(dplyr)

file_lines <- "
library(dplyr)
require(tidyr)
library(madeUpPkg)

as_tibble(mpg) %>%
  group_by(class) %>%
  nest() %>%
  mutate(stats = purrr::map(data,
                            ~lm(cty ~ hwy, data = .x)))

made_up_fun()
"

file_output <- tempfile(fileext = ".R")
writeLines(file_lines, file_output)
```

```
spot_pkgs(file_output) %>%  
  check_pkgs_availability()
```

list_files_github_repo

List Files in Github Repo

Description

Return a dataframe containing the paths of files in a github repository. Generally used prior to `spot_{fun/pkgs}_files()`.

Usage

```
list_files_github_repo(  
  repo,  
  branch = NULL,  
  pattern = stringr::regex("(r|rmd|rmarkdown|qmd)$", ignore_case = TRUE),  
  rmv_index = TRUE  
)
```

Arguments

repo	Github repository, e.g. "brshallo/feat-eng-lags-presentation"
branch	Branch of github repository, default is "main".
pattern	Regex pattern to keep only matching files. Default is <code>stringr::regex("(r rmd rmarkdown qmd)\$", ignore_case = TRUE)</code> which will keep only R, Rmarkdown and Quarto documents. To keep all files use " . ".
rmv_index	Logical, most repos containing blogdown sites will have an index.R file at the root. Change to FALSE if you don't want this file removed.

Value

Dataframe with columns of `relative_paths` and `absolute_paths` for file path locations. `absolute_paths` will be urls to raw files.

See Also

`list_files_wd`, `list_files_github_gists`

Examples

```
## Not run:
library(dplyr)
library(funspotr)

# pulling and analyzing my R file github gists
gh_urls <- list_files_github_repo("brshallo/feat-eng-lags-presentation", branch = "main")

# Will just parse the first 2 files/gists
contents <- spot_funcs_files(slice(gh_urls, 1:2))

contents %>%
  unnest_results()

## End(Not run)
```

list_files_wd	<i>List Files in Working Directory</i>
----------------------	----------------------------------------

Description

Return a dataframe containing the paths of files in the working directory. Generally used prior to `spot_{funcs/pkgs}_files()`.

Usage

```
list_files_wd(
  path = ".",
  pattern = stringr::regex("(r|rmd|rmarkdown|qmd)$", ignore_case = TRUE),
  rmv_index = TRUE
)
```

Arguments

<code>path</code>	Character vector or path. Default is <code>."</code> which will set the starting location for <code>relative_paths</code> .
<code>pattern</code>	Regex pattern to keep only matching files. Default is <code>stringr::regex("(r rmd rmarkdown qmd)\$", ignore_case = TRUE)</code> which will keep only R, Rmarkdown and Quarto documents. To keep all files use <code>".."</code> .
<code>rmv_index</code>	Logical, most repos containing blogdown sites will have an <code>index.R</code> file at the root. Change to <code>FALSE</code> if you don't want this file removed.

Details

Can also be used outside of working directory if `path` is specified.

Value

Dataframe with columns `relative_paths` and `absolute_paths`.

See Also

`list_files_github_repo`, `list_files_github_gists`

Examples

```
## Not run:
library(dplyr)
library(funspotr)

# pulling and analyzing my R file github gists
files_local <- list_files_wd()

# Will just parse the first 2 files/gists
contents <- spot_funcs_files(slice(files_local, 2:3))

contents %>%
  unnest_results()

## End(Not run)
```

network_plot

funspotr Network Plot

Description

Output simple network plot using `visNetwork` connecting either `fun`s or `pkgs` to `relative_paths/absolute_paths`.

Usage

```
network_plot(df, to = .data$pkgs, show_each_use = FALSE)
```

Arguments

<code>df</code>	Dataframe containing columns <code>relative_paths</code> , <code>absolute_paths</code> and either <code>fun</code> s or <code>pkgs</code> . Generally the output from running: <code>github_spot_*</code> () %>% <code>unnest_results</code> ()
<code>to</code>	<code>fun</code> s or <code>pkgs</code>
<code>show_each_use</code>	Binary, default is FALSE. If TRUE edge thickness will be based on the number of times a package or function is used.

Value

`visNetwork` plot

Examples

```
## Not run:
library(dplyr)
library(funspotr)

gh_ex_pkgs <- list_files_github_repo(
  repo = "brshallo/feat-eng-lags-presentation",
  branch = "main") %>%
  spot_funcs_files()

gh_ex_pkgs %>%
  unnest_results() %>%
  network_plot(to = pkgs)

## End(Not run)
```

spot_funcs

Spot Functions

Description

Given `file_path` extract all functions and their associated packages from specified file.

Usage

```
spot_funcs(file_path, ...)
```

Arguments

<code>file_path</code>	character vector of path to file.
<code>...</code>	This allows you to pass additional arguments through to <code>spot_funcs_custom()</code> .

Details

`spot_funcs()` uses `spot_funcs_custom()` to run – it is a less verbose version and does not require passing in the packages separately. See README and `?spot_funcs_custom` for details on how the function works and arguments that can be passed through (via `...`).

If code syntax is malformed and cannot be properly parsed, function will error.

Value

Given default arguments and no missing packages, a dataframe with the following columns is returned:

`fun`: specifying functions in file. `pkgs`: the package a function came from. If `fun` is a custom function or if it came from a package not installed on your machine, `pkgs` will return "(unknown)".

Note that any unused loaded packages / `pkgs` are dropped from output. Any functions without an available package are returned with the value "(unknown)".

See README for further documentation.

See Also

[spot_funs_custom\(\)](#), [spot_funs_files\(\)](#)

Examples

```
library(funspotr)

file_lines <- "
library(dplyr)
require(tidyr)
library(madeUpPkg)

as_tibble(mpg) %>%
  group_by(class) %>%
  nest() %>%
  mutate(stats = purrr::map(data,
    ~lm(cty ~ hwy, data = .x)))

made_up_fun()
"

file_output <- tempfile(fileext = ".R")
writeLines(file_lines, file_output)

spot_funs(file_output)
```

Description

Engine that runs `spot_funs()`. `spot_funs_custom()` has options for changing returned output and for producing print statements and errors. It also requires you to provide a character vector for `pkgs` rather than identifying these automatically via `spot_pkgs()`.

Usage

```
spot_funs_custom(
  pkgs,
  file_path,
  show_each_use = FALSE,
  keep_search_list = FALSE,
  copy_local = TRUE,
  print_pkgs_load_status = FALSE,
  error_if_missing_pkg = FALSE,
  keep_in_multiple_pkgs = FALSE
)
```

Arguments

<code>pkgs</code>	Character vector of packages that are added to search space via <code>require()</code> or <code>import::from()</code> so can be found by <code>utils::find()</code> . Generally will be the returned value from <code>spot_pkgs(file_path, show_explicit_funs = TRUE)</code> .
<code>file_path</code>	character vector of path to file.
<code>show_each_use</code>	Logical, default is FALSE. If changed to TRUE will return individual rows for each time a function is used (rather than just once for the entire file).
<code>keep_search_list</code>	Logical, default is FALSE. If changed to TRUE will include entire search list for function. May be helpful for debugging in cases where funspotr may not be doing a good job of recreating the search list for identifying which packages function(s) came from. This will print all packages in the search list for each function.
<code>copy_local</code>	Logical, if changed to FALSE will not copy to a local temporary folder prior to doing analysis. Many functions require file to already be an .R file and for the file to exist locally. This should generally not be set to TRUE unless these hold.
<code>print_pkgs_load_status</code>	Logical, default is FALSE. If set to TRUE will print a named vector of logicals showing whether packages are on machine along with any warning messages that come when running <code>require()</code> . Will continue on to produce output of function.
<code>error_if_missing_pkg</code>	Logical, default is FALSE. If set to TRUE then <code>print_pkgs_load_status = TRUE</code> automatically. If a package is not installed on the machine then will print load status of individual pkgs and result in an error.
<code>keep_in_multiple_pkgs</code>	Logical, default is FALSE. If set to TRUE will include in the outputted dataframe a column <code>in_multiple_pkgs</code> : logical, whether a function exists in multiple packages loaded (i.e. on the search space of <code>utils::find()</code>).

Details

`spot_funs_custom()` is also what you should use in cases where you don't trust `spot_pkgs()` to properly identify package dependencies from within the same file and instead want to pass in your own character vector of packages.

See README for a description of how the function works.

If a package is not included in `pkgs`, any functions called that should come from that package will be assigned a value of "(unknown)" in the `pkgs` column of the returned output. You can also use the `print_pkgs_load_status` and `error_if_missing_pkg` arguments to alter how output works in cases when not all packages are on the machine.

Explicit calls to unexported functions i.e. `pkg:::fun()` will have `pkgs = "(unknown)"` in the returned dataframe.

Value

Given default arguments and no missing packages, a dataframe with the following columns is returned:

fun: specifying functions in file. pkgs: the package a function came from. If funs is a custom function or if it came from a package not installed on your machine, pkgs will return "(unknown)".

Note that any unused loaded packages / pkgs are dropped from output. Any functions without an available package are returned with the value "(unknown)".

See README for further documentation.

See Also

[spot_funs\(\)](#)

Examples

```
library(funspotr)

file_lines <- "
library(dplyr)
require(tidyr)
library(madeUpPkg)

as_tibble(mpg) %>%
  group_by(class) %>%
  nest() %>%
  mutate(stats = purrr::map(data,
    ~lm(cty ~ hwy, data = .x)))

made_up_fun()
"

file_output <- tempfile(fileext = ".R")
writeLines(file_lines, file_output)

pkgs <- spot_pkgs(file_output)

spot_funs_custom(pkgs, file_output)

# If you'd rather it error when a pkg doesn't exist e.g. for {madeUpPkg}
# set`error_if_missing_pkg = TRUE`
```

Description

`spot_pkgs_files()` : Spot all packages that show-up in R or Rmarkdown or quarto documents in a dataframe of filepaths.

`spot_funs_files()` : Spot all functions and their corresponding packages that show-up in R or Rmarkdown or quarto documents in a dataframe of filepaths.

Usage

```
spot_funs_files(df, ..., .progress = TRUE)

spot_pkgs_files(df, ..., .progress = TRUE)
```

Arguments

df	Dataframe containing a column of <code>absolute_paths</code> .
...	Arguments passed onto <code>spot_{pkgs funs}()</code> .
<code>.progress</code>	Whether to show a progress bar. Use <code>TRUE</code> to turn on a basic progress bar, use a string to give it a name, or see progress_bars for more details.

Details

A `purrr::safely()` wrapper for mapping `spot_pkgs()` or `spot_funs()` across multiple filepaths. I.e. even if some files fail to parse the function will continue on.

Default settings are meant for files where package libraries are referenced *within* the files themselves. See README for more details.

Value

Dataframe with `relative_paths` and `absolute_paths` of file paths along with a list-column `spotted` containing `purrr::safely()` named list of "result" and "error" for each file parsed. Use `unnest_results()` to unnest only the "result" values.

See Also

[spot_pkgs\(\)](#), [spot_funs\(\)](#), [unnest_results\(\)](#)

Examples

```
## Not run:
library(funspotr)
library(dplyr)

list_files_github_repo("brshallo/feat-eng-lags-presentation", branch = "main") %>%
  spot_funs_files()

## End(Not run)
```

spot_pkgs*Spot Packages*

Description

Extract all pkg called in either `library(pkg)`, `require(pkg)` `requireNamespace("pkg")` or `pkg::fun()`. Will not identify packages loaded in other ways not typically done in interactive R scripts (e.g. relying on a `DESCRIPTION` file for a pkg or something like `source("lib-calls.R")`). Inspiration: [blogdown#647](#).

Usage

```
spot_pkgs(
  file_path,
  show_explicit_funcs = FALSE,
  copy_local = TRUE,
  as_yaml_tags = FALSE
)
```

Arguments

<code>file_path</code>	String of path to file of interest.
<code>show_explicit_funcs</code>	In cases where a function is called explicitly, show both the package dependency and the function together. For example a script containing <code>dplyr::select()</code> (as opposed to <code>library(dplyr); select()</code>) would have <code>spot_pkgs(show_explicit_funcs = TRUE)</code> return the item as "dplyr::select" rather than just "dplyr".
<code>copy_local</code>	Logical, default is TRUE. If changed to FALSE will not copy to a local temporary folder prior to doing analysis. Many processes require file to already be a .R file and for the file to exist locally, hence this should usually be set to TRUE.
<code>as_yaml_tags</code>	Logical, default is FALSE. If set to TRUE flattens and puts into a format convenient for pasting in "tags" section of YAML header of a Rmd document for a blogdown post.

Details

In cases where `show_explicit_funcs = TRUE` and there are explicit calls in the package, "pkg:fun" is returned instead.

Packages are extracted solely based on text – not whether the package actually exists or not. Hence even packages that you do not have installed on your machine but show-up in the script will be returned in the character vector.

Value

Character vector of all packages loaded in file.

See Also

[spot_pkgs_used\(\)](#), [spot_pkgs_from_description\(\)](#), [spot_pkgs_files\(\)](#), [renv::dependencies\(\)](#)

Examples

```
library(funspotr)

file_lines <- "
library(dplyr)
require(tidyr)
library(madeUpPkg)

as_tibble(mpg) %>%
  group_by(class) %>%
  nest() %>%
  mutate(stats = purrr::map(data,
    ~lm(cty ~ hwy, data = .x)))

made_up_fun()
"

file_output <- tempfile(fileext = ".R")
writeLines(file_lines, file_output)

spot_pkgs(file_output)

# To view `purrr::map` as an explicit call
spot_pkgs(file_output, show_explicit_funcs = TRUE)

# To output for blogdown post YAML header tags
cat(spot_pkgs(file_output, as_yaml_tags = TRUE))
```

spot_pkgs_used	<i>Spot Packages Used</i>
----------------	---------------------------

Description

Primarily used for cases where you load metapackages like `tidyverse` or `tidymodels` but only want to return those packages that have functions from the package that are actually called. E.g. say you have a `library(tidyverse)` call but only end-up using functions that are in `dplyr` – in that case `spot_pkgs()` would return "tidyverse" whereas `spot_pkgs_used()` would return "dplyr".

Usage

```
spot_pkgs_used(file_path, as_yaml_tags = FALSE)
```

Arguments

file_path	String of path to file of interest.
as_yaml_tags	Logical, default is FALSE. If set to TRUE flattens and puts into a format convenient for pasting in "tags" section of a blogdown post Rmd document.

Details

Also does not return uninstalled packages or those loaded when R starts up.

Is essentially just calling `spot_funs() |> with(unique(pkgs))` in the background. Does not have as many options as `spot_pkgs()` though.

Value

Character vector of all packages with functions used in the file.

spot_tags

Spot Tags

Description

Put quoted inline R function in your blogdown or quarto post's YAML header to have the packages be the packages used in your post (wrapper around `funspotr::spot_pkgs()`).

Usage

```
spot_tags(
  file_path = knitr::current_input(),
  used = FALSE,
  drop_knit = FALSE,
  yaml_bullet = NULL,
  ...
)
```

Arguments

file_path	Default is the file being knitted but can change to some other file (e.g. in cases where the code for the post may reside in a different file).
used	Default is FALSE. If TRUE will pass to <code>show_pkgs_used()</code> rather than <code>show_pkgs()</code> . (Mainly useful for showing actual packages used rather than meta-packages being called like <code>tidyverse</code> or <code>tidymodels</code> . Also uses a more strict parsing method.
drop_knit	Many blogdown posts have <code>knitr::opts_chunk\$set()</code> in them and you may not want this tag showing-up. Default is to keep this, but set to FALSE to drop "knitr" from being tagged.

yaml_bullet	Default is NULL meaning that <code>file_path</code> is read-in and correct format is guessed based on "spot_tags" appearance with either a hyphen or bracket (corresponding with bulleted or array format in the YAML header). If it's first occurrence happens on a line that contains a bracket the value becomes FALSE else it becomes TRUE. If set to NULL and "spot_tags" is not detected at all in <code>file_path</code> it will default to FALSE. <code>yaml_bullet</code> can also be specified directly with either TRUE or FALSE. TRUE entails that <code>spot_tags()</code> is set in a YAML bullet, FALSE indicates the user is inputting it in an array (see examples below). See examples for how to hard-code.
...	Any additional arguments to pass to <code>spot_pkgs*()</code> .

Details

```
tags:
  - "`r funspotr::spot_tags()`"
```

OR

```
tags: ["`r funspotr::spot_tags()`"]
```

OR

```
categories: ["`r funspotr::spot_tags()`"]
```

Thanks Yihui for the suggestions and for getting this working [blogdown#647](#), [blogdown#693](#).)

Value

Character vector in a format meant to be read while evaluating the YAML header when rendering.

See Also

[spot_pkgs\(\)](#), [spot_pkgs_used\(\)](#)

Examples

```
# To review input interactively from within rstudio you might also try:
## Not run:
funspotr::spot_tags(rstudioapi::getSourceEditorContext()$path)

## End(Not run)
```

unnest_results	<i>Unnest Results</i>
----------------	-----------------------

Description

Run after running `list_files_*`() |> `spot_{fun|pkgs}_files()` to unnest the spotted list-column.

Usage

```
unnest_results(df)
```

Arguments

`df` Dataframe outputted by `spot_{fun|pkgs}_files()` that contains a spotted list-column.

Value

An unnested dataframe with what was in spotted moved to the front.

See Also

[spot_fun_files\(\)](#), [spot_pkgs_files\(\)](#)

Examples

```
## Not run:
library(funspotr)
library(dplyr)

list_files_github_repo("brshallo/feat-eng-lags-presentation", branch = "main") %>%
  spot_fun_files() %>%
  unnest_results()

## End(Not run)
```

Index

check_pkgs_availability, 2
list_files_github_repo, 3
list_files_wd, 4
network_plot, 5
progress_bars, 10
spot_funcs, 6
spot_funcs(), 9, 10
spot_funcs_custom, 7
spot_funcs_custom(), 7
spot_funcs_files, 9
spot_funcs_files(), 7, 15
spot_pkgs, 11
spot_pkgs(), 10, 14
spot_pkgs_files(spot_funcs_files), 9
spot_pkgs_files(), 12, 15
spot_pkgs_from_description(), 12
spot_pkgs_used, 12
spot_pkgs_used(), 12, 14
spot_tags, 13
unnest_results, 15
unnest_results(), 10