Package 'IssueTrackeR'

April 25, 2025

Type Package

Title List Things to Do

Version 1.1.1

Description Manage a 'GitHub' problem using R: wrangle issues, labels and milestones. It includes functions for storing, prioritizing (sorting), displaying, adding, deleting, and selecting (filtering) issues based on qualitative and quantitative information. Issues (labels and milestones) are written in lists and categorized into the S3 class to be easily manipulated as datasets in R.

License MIT + file LICENSE

URL https://github.com/TanguyBarthelemy/IssueTrackeR, https://tanguybarthelemy.github.io/IssueTrackeR/

BugReports https://github.com/TanguyBarthelemy/IssueTrackeR/issues

Depends R (>= 4.1)

Imports crayon, gh, tools, yaml

Suggests cli, rlang, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.2

NeedsCompilation no

Author Tanguy Barthelemy [aut, cre]

Maintainer Tanguy Barthelemy <tanguy.barthelemy@insee.fr>

Repository CRAN

Date/Publication 2025-04-25 12:30:14 UTC

Contents

append 2
contains 3
filter_issues 6
format_issues 7
format_labels 8
format_milestones 9
get_issues 10
new_issue 12
new_issues 13
print.IssueTB 14
sort.IssuesTB 15
update_database 17
write_issues_to_dataset 18

Index **21**

---

append *Vector Merging*

Description

Add elements to a vector.

Usage

```
append(x, values, after = length(x))

## S3 method for class 'IssuesTB'
append(x, values, after)
```

Arguments

x	the vector the values are to be appended to.
values	a IssueTB or a IssuesTB object.
after	a subscript, after which the values are to be appended.

Value

A vector containing the values in x with the elements of values appended after the specified element of x.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Examples

```
append(1:5, 0:1, after = 3)
```

`contains` *Does the issue(s) contains this text?*

Description

Check if the issues contains text, values in its title, labels, body and milestone.

Usage

```
contains(x, ...)

## S3 method for class 'IssueTB'
contains(
  x,
  values,
  fields = c("body", "title", "labels", "milestone"),
  values_logic_gate = c("AND", "OR"),
  fields_logic_gate = c("OR", "AND"),
  negate = FALSE,
  ...
)

## S3 method for class 'IssuesTB'
contains(x, values, ...)

## Default S3 method:
contains(x, ...)
```

Arguments

`x`	a `IssueTB` or `IssuesTB` object.
`...`	Arguments passed on to `vgrepl` and therefore to `grepl`
`values`	a vector `string`. Patterns to look for in the outcome.
`fields`	a vector `string`. The different fields of the issue in which to search for the pattern (among `"title"`, `"body"`, `"labels"` and `"milestone"`).
`values_logic_gate`	the logic operator which will aggregate the different assertion related to values: `"OR"` or `"AND"` (by default).
`fields_logic_gate`	the logic operator which will aggregate the different assertion related to fields: `"OR"` (by default) or `"AND"`.
`negate`	a boolean indicate the negation of the assertion.

Details

The contains function in R is designed to check if specific fields of GitHub issues contain certain values, offering a flexible mechanism for constructing complex assertions. The function operates with two main logical gates: `fields_logic_gate` and `values_logic_gate`.

The `fields_logic_gate` determines how conditions on multiple fields are combined (either "OR" or "AND"). This means that the call `contains(x = issue_1, fields = c("body", "title"), values = "README", fields_logic_gate = "OR")` will say whether the issue `issue_1` contains the string `"README"` in its title OR in its body.

The `values_logic_gate` specifies how conditions on multiple values are combined within each field (either "OR" or "AND"). For example the call

```
contains(x = issue_1,
         fields = "body",
         values = c("README", "package"),
         values_logic_gate = "OR")
```

will say whether the issue `issue_1` contains the string `"README"` OR `"package"` in its body. Whereas the call

```
contains(x = issue_1,
         fields = "title",
         values = c("README", "package"),
         values_logic_gate = "AND")
```

will say whether the issue `issue_1` contains the string `"README"` AND `"package"` in its body.

The function can also negate the condition using the `negate` argument, effectively allowing users to negate an assertion.

The following example:

```
contains(
    x = all_issues,
    fields = "labels",
    values = c("unknown", "medium"),
    values_logic_gate = "OR",
    negate = TRUE,
    fields_logic_gate = "AND"
)
```

designates issues that contain neither "unknown" nor "medium" in their label.

Note that in the last example, the `fields_logic_gate` argument has no importance and is not taken into account because there is only one field on which to filter. In the same way, if the `values` argument contains only one element, the `values_logic_gate` argument has no importance and is not taken into account.

This function is not case-sensitive.

Value

a boolean (of length equals 1 if the class of `x` is `IssueTB` and length superior to 1 if `x` if of class `IssuesTB`) specifying if the `pattern` is contained in the field `field` of the issue.

How assertions with multiple values and multiple fields are built

For the order of logical assertions, as it is easy to add assertions linked by an AND (by piping a new filter_issues), it has been decided that assertions containing AND gates will be distributed and assertions containing OR gates will be factorised. The assertions used by `filter_issues` will therefore have the following format: $(P1ANDQ1)OR(P2ANDQ2)$

Thus the following call to filter_issue:

```
filter_issues(
    ...,
    values = c("v1", "v2"), fields = c("f1", "f2"),
    values_logic_gate = "AND", fields_logic_gate = "OR",
    ...
)
```

will be represented by the following logical proposition: $(v1inf1ANDv2inf1)OR(v1inf2ANDv2inf2)$.

This makes it possible to create more complex logical forms by combining AND gates and OR gates.

Short names

- `fields` = `"b"` for `"body"`;
- `fields` = `"t"` for `"title"`;
- `fields` = `"l"` for `"labels"`;
- `fields` = `"m"` for `"milestone"`.

Examples

```
all_issues <- get_issues(source = "online", verbose = FALSE)
issue_1 <- all_issues[[1L]]
# This will return TRUE if the issue contains either "README" or "package"
# in its body.
contains(x = issue_1,
         fields = "body",
         values = c("README", "package"),
         values_logic_gate = "OR")
# This will return TRUE if the issue contains "README" in its body AND its
# title.
contains(x = issue_1,
         values = "README",
         fields = c("body", "title"),
         fields_logic_gate = "AND")
```

`filter_issues` *Filter issue or issues*

Description

Filtering issues with some constraint on the labels, the title and the body.

Usage

```
filter_issues(x, ...)

## S3 method for class 'IssuesTB'
filter_issues(x, ...)

## Default S3 method:
filter_issues(x, ...)
```

Arguments

`x`	a `IssuesTB` object.
`...`	Other options used to control filtering behaviour with different fields and values. Passed on to `contains` as: • `values`: a vector `string`. Patterns to look for in the outcome. • `fields`: a vector `string`. The different fields of the issue in which to search for the pattern (among `"title"`, `"body"`, `"labels"` and `"milestone"`) • `fields_logic_gate`: the logic operator which will aggregate the different assertion related to fields: `"OR"` (by default) or `"AND"`. • `values_logic_gate`: the logic operator which will aggregate the different assertion related to values: `"OR"` or `"AND"` (by default). • `negate`: a boolean indicate the negation of the assertion.

Details

This function relies on the function `contains`. More informations on the filtering in the documentation of the function `contains`.

Value

a `IssuesTB` object filtered

Examples

```
all_issues <- get_issues(source = "online", verbose = FALSE)
# Condition: issues containing "README" in its body OR title
filtered_issues <- filter_issues(
    x = all_issues,
```

```
    fields = c("body", "title"),
    values = "README",
    fields_logic_gate = "OR"
)

# Condition: issues containing neither "unknown" nor "medium" in their label
filtered_issues <- filter_issues(
    x = all_issues,
    fields = "labels",
    values = c("unknown", "medium"),
    values_logic_gate = "OR",
    negate = TRUE,
    fields_logic_gate = "AND"
)
```

`format_issues` *Format the issue in a simpler format*

Description

Format the issue in a simpler format

Usage

```
format_issues(
  raw_issues,
  raw_comments,
  repo = getOption("IssueTrackeR.repo"),
  owner = getOption("IssueTrackeR.owner"),
  verbose = TRUE
)
```

Arguments

`raw_issues`	a `gh_response` object output from the function `gh` which contains all the data and metadata for GitHub issues.
`raw_comments`	a `gh_response` object output from the function `gh` which contains all the data and metadata for GitHub comments.
`repo`	A character string specifying the GitHub repository name (only taken into account if `source` is set to `"online"`). Defaults to the package option `IssueTrackeR.repo`.
`owner`	A character string specifying the GitHub owner (only taken into account if `source` is set to `"online"`). Defaults to the package option `IssueTrackeR.owner`.
`verbose`	A logical value indicating whether to print additional information. Default is `TRUE`.

Value

a list representing an issue with simpler structure (with number, title, body and labels) of all issues.

Examples

```
raw_issues <- gh::gh(
    repo = "rjdemetra",
    owner = "rjdverse",
    endpoint = "/repos/:owner/:repo/issues",
    .limit = Inf
)
raw_comments <- gh::gh(
    repo = "rjdemetra",
    owner = "rjdverse",
    endpoint = "/repos/:owner/:repo/issues/comments",
    .limit = Inf
)
all_issues <- format_issues(raw_issues = raw_issues,
                            raw_comments = raw_comments,
                            verbose = FALSE)
```

---

format_labels *Format the label in a simpler format*

Description

Format the label in a simpler format

Usage

```
format_labels(raw_labels, verbose = TRUE)
```

Arguments

raw_labels	a gh_response object output from the function gh which contains all the data and metadata for GitHub labels.
verbose	A logical value indicating whether to print additional information. Default is TRUE.

Value

a list representing labels with simpler structure (with name, description, colour)

Examples

```
# With labels
raw_labels <- gh::gh(
   repo = "rjdemetra",
   owner = "rjdverse",
   endpoint = "/repos/:owner/:repo/labels",
   .limit = Inf
)
format_labels(raw_labels)
```

`format_milestones` *Format the milestones in a simpler format*

Description

Format the milestones in a simpler format

Usage

```
format_milestones(raw_milestones, verbose = TRUE)
```

Arguments

`raw_milestones`	a `gh_response` object output from the function `gh` which contains all the data and metadata for GitHub milestones.
`verbose`	A logical value indicating whether to print additional information. Default is `TRUE`.

Value

a list representing milestones with simpler structure (with title, description and due_on)

Examples

```
# With milestones
milestones_jdplus_main <- gh::gh(
    repo = "jdplus-main",
    owner = "jdemetra",
    endpoint = "/repos/:owner/:repo/milestones",
    state = "all",
    .limit = Inf
 )
format_milestones(milestones_jdplus_main)
```

get_issues *Retrieve information from the issues of GitHub*

Description

use gh to ask the API of GitHub and et a list of issues with their labels and milestones.

Usage

```
get_issues(
  source = c("local", "online"),
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
  dataset_name = "open_issues.yaml",
  repo = getOption("IssueTrackeR.repo"),
  owner = getOption("IssueTrackeR.owner"),
  state = c("open", "closed", "all"),
  verbose = TRUE
)

get_labels(
  source = c("local", "online"),
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
  dataset_name = "list_labels.yaml",
  repo = getOption("IssueTrackeR.repo"),
  owner = getOption("IssueTrackeR.owner"),
  verbose = TRUE
)

get_milestones(
  source = c("local", "online"),
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
  dataset_name = "list_milestones.yaml",
  repo = getOption("IssueTrackeR.repo"),
  owner = getOption("IssueTrackeR.owner"),
  verbose = TRUE
)
```

Arguments

source	a character string that is either "online" if you want to fetch information from GitHub or "local" (by default) if you want to fetch information locally.
dataset_dir	A character string specifying the path which contains the datasets (only taken into account if source is set to "local"). Defaults to the package option IssueTrackeR.dataset.dir.
dataset_name	A character string specifying the name of the datasets which will be written (only taken into account if source is set to "local"). Defaults to "open_issues.yaml".

repo	A character string specifying the GitHub repository name (only taken into account if source is set to "online"). Defaults to the package option IssueTrackeR.repo.
owner	A character string specifying the GitHub owner (only taken into account if source is set to "online"). Defaults to the package option IssueTrackeR.owner.
state	a character string that is either "open" (by default) if you want to fetch only open issues from GitHub, "closed" if you want to fetch only closed issues from GitHub or "all" if you want to fetch all issues from GitHub (closed and open). Only taken into account if source is set to "online".
verbose	A logical value indicating whether to print additional information. Default is TRUE.

Details

The functions of get type are useful to retrieve object related to issues from GitHub. So it's possible to retrieve issues, labels and milestones.

The defaults value for the argument dataset_name depends on the function:

- defaults is "list_issues.yaml" for get_issues()
- defaults is "list_milestones.yaml" for get_milestones()
- defaults is "list_labels.yaml" for get_labels()

Value

The function get_issues returns an object of class IssuesTB. It is a list composed by object of class IssueTB. An object of class IssueTB represents an issue with simpler structure (with number, title, body and labels).

The function get_labels returns a list representing labels with simpler structure (with name, description, colour).

The function get_milestones returns a list representing milestones with simpler structure (with title, description and due_on).

Examples

```
# From online

issues <- get_issues(source = "online")
print(issues)

labels <- get_labels(source = "online")
print(labels)

milestones <- get_milestones(source = "online")
print(milestones)


# From local
```

```
# First update the local database
update_database(verbose = TRUE)

issues <- get_issues(source = "local",
                     dataset_name = "open_issues.yaml",
                     state = "open")
labels <- get_labels(source = "local")
milestones <- get_milestones(source = "local")
```

`new_issue` *Create a new* `IssueTB` *object*

Description

Create a new `IssueTB` object

Usage

```
new_issue(
  title,
  body,
  number,
  state = c("open", "closed"),
  created_at = Sys.Date(),
  labels = NULL,
  milestone = NULL,
  issue = list(),
  repo = NULL,
  owner = NULL,
  ...
)
```

Arguments

`title`	a string. The title of the issue.
`body`	a string. The title of the issue.
`number`	a string. The title of the issue.
`state`	a character string that is either `"open"` (by default) if the issue is still open or `"closed"` if the issue is now closed.
`created_at`	a date. The title of the issue.
`labels`	a vector string (or missing). The labels of the issue.
`milestone`	a string (or missing). The milestone of the issue.
`issue`	a list representing the object.

repo	A character string specifying the GitHub repository name (only taken into account if source is set to "online"). Defaults to the package option IssueTrackeR.repo.
owner	A character string specifying the GitHub owner (only taken into account if source is set to "online"). Defaults to the package option IssueTrackeR.owner.
...	Other information we would like to add to the issue.

Value

a IssueTB object.

Examples

```
# Empty issue
issue1 <- new_issue()

# Custom issue
issue2 <- new_issue(
    title = "Nouvelle issue",
    body = "Un nouveau bug pour la fonction...",
    number = 47,
    created_at = Sys.Date()
)

issue3 <- new_issue(issue = issue2)
```

---

new_issues *Create a new* IssuesTB *object*

Description

Create a new IssuesTB object

Usage

```
new_issues(x = list())

## S3 method for class 'IssueTB'
new_issues(x)

## S3 method for class 'IssuesTB'
new_issues(x)

## Default S3 method:
new_issues(x = list())
```

Arguments

x	a list containing IssueTB objects

Value

a IssuesTB object.

Examples

```
# Empty issue
issues1 <- new_issues()

# Custom issue
issues2 <- new_issues(
    x = new_issue(
        title = "Une autre issue",
        body = "J'ai une question au sujet de...",
        number = 2,
        created_at = Sys.Date()
    )
)

issues3 <- new_issues(x = list(
    new_issue(
        title = "Nouvelle issue",
        body = "Un nouveau bug pour la fonction...",
        state = "open",
        number = 1,
        created_at = Sys.Date()
    ),
    new_issue(
        title = "Une autre issue",
        body = "J'ai une question au sujet de...",
        state = "closed",
        number = 2,
        created_at = Sys.Date()
    )
))
```

---

print.IssueTB *Display IssueTB and IssuesTB object*

Description

Display IssueTB and IssuesTB with formatted output in the console

Usage

```
## S3 method for class 'IssueTB'
print(x, ...)

## S3 method for class 'IssuesTB'
print(x, ...)
```

Arguments

x	a IssueTB or IssuesTB object.
...	Unused argument

Details

This function displays an issue (IssueTB object) or a list of issues (IssuesTB object) with a formatted output.

Value

invisibly (with invisible()) NULL.

Examples

```
all_issues <- get_issues(source = "online", verbose = FALSE)

# Display one issue
print(all_issues[[1]])

# Display several issues
print(all_issues[1:10])
```

---

sort.IssuesTB *Sort issues*

Description

Sorting issues with some constraint and order on the labels, the title, the milestones and/or the body.

Usage

```
## S3 method for class 'IssuesTB'
sort(
  x,
  decreasing = FALSE,
  sorting_variables = list(),
  filtering_factors = list(),
  ...
)
```

Arguments

x	a IssuesTB object.
decreasing	logical. Should the sort be increasing or decreasing?
sorting_variables	a list containing the quantitative variables to sort the issues. The filters are applied in the order of the variables supplied.
filtering_factors	a list containing constraints for sorting issues by sub-group in order of priority
...	Additional arguments related to milestones for the function simple_sort.

Details

In the order of the constraints imposed by the filtering_factors argument, the function will first filter by constraint. For each constraint, the function will then sort according to the quantitative variables supplied in sorting_variables.

For example, the following call:

```
sort(
    x = issues,
    sorting_variables = list(c(object = "milestones", field = "due_on"),
                             c(object = "issues", field = "created_at")),
    filtering_factors = list(list(values = "bug",
                                  fields = "labels",
                                  values_logic_gate = "OR"),
                             list(values = "package", fields = "title")),
    decreasing = TRUE
)
```

will behave as follows:

- 1. It will select all the issues that have "bug" as a label, then sort them according to the chronological order of milestones (according to deadlines) and the chronological order of issue creation dates
- 1. Among the remaining issues, it will filter the issues that have "package" in the title and apply the same sorting.
- 1. Finally, among all the remaining issues (not sorted until now), the function will apply the same sorting.
- 1. The function returns the global list of sorted issues.

The argument filtering_factors is a list of constraint following the same naming convention as the filter_issues. So the constraints are represented by named lists with the various arguments (apart from x) to the filter_issues (values, fields, fields_logic_gate, values_logic_gate and negate).

Value

a IssuesTB object sorted.

Examples

```
# Get the milestones of the prject
milestones <- get_milestones("online")
write_milestones_to_dataset(milestones)

all_issues <- get_issues(source = "online", verbose = FALSE)
sorted_issues <- sort(
    x = all_issues,
    sorting_variables = list(list(object = "milestones", field = "due_on"),
                             list(object = "issues", field = "created_at")),
    filtering_factors = list(list(values = "bug",
                                  fields = "labels",
                                  values_logic_gate = "OR"),
                             list(values = "package", fields = "title")),
    milestones = milestones
)
```

---

update_database *Update database*

Description

Update the different local database (issues, labels and milestones) with the online reference.

Usage

```
update_database(
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
 datasets_name = c(open = "open_issues.yaml", closed = "closed_issues.yaml", labels =
    "list_labels.yaml", milestones = "list_milestones.yaml"),
  verbose = TRUE,
  ...
)
```

Arguments

dataset_dir	A character string specifying the path which contains the datasets (only taken into account if source is set to "local"). Defaults to the package option IssueTrackeR.dataset.dir.
datasets_name	A named character string of length 4, specifying the names of the different datasets which will be written. The names datasets_name have to be "open", "closed", "labels" and "milestones". Defaults to c(open = "open_issues.yaml", closed = "closed_issues.yaml", labels = "list_labels.yaml", milestones = "list_milestones.yaml") .

verbose — A logical value indicating whether to print additional information. Default is TRUE.

... — Additional arguments for connecting to the GitHub repository:

- repo A character string specifying the GitHub repository name. Defaults to the package option IssueTrackeR.repo.
- owner A character string specifying the GitHub owner. Defaults to the package option IssueTrackeR.owner. (See the documentation of get to have more information on theses parameters):

Value

invisibly (with invisible()) TRUE.

Examples

```
update_database()
```

---

write_issues_to_dataset

Save datasets in a yaml file

Description

Save datasets in a yaml file

Usage

```
write_issues_to_dataset(issues, ...)

## S3 method for class 'IssuesTB'
write_issues_to_dataset(
  issues,
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
  dataset_name = "list_issues.yaml",
  verbose = TRUE,
  ...
)

## Default S3 method:
write_issues_to_dataset(issues, ...)

write_labels_to_dataset(
  labels,
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
```

```
  dataset_name = "list_labels.yaml",
  verbose = TRUE
)

write_milestones_to_dataset(
  milestones,
  dataset_dir = getOption("IssueTrackeR.dataset.dir"),
  dataset_name = "list_milestones.yaml",
  verbose = TRUE
)
```

Arguments

issues	a IssuesTB object.
...	Unused parameter.
dataset_dir	A character string specifying the path which contains the datasets (only taken into account if source is set to "local"). Defaults to the package option IssueTrackeR.dataset.dir.
dataset_name	A character string specifying the name of the datasets which will be written (only taken into account if source is set to "local"). Defaults to "open_issues.yaml".
verbose	A logical value indicating whether to print additional information. Default is TRUE.
labels	a list representing all labels with simpler structure (with name, description, colour)
milestones	a list representing milestones with simpler structure (with title, description and due_on).

Details

Depending on the object, the defaults value of the argument dataset_name is:

- "list_issues.yaml" for issues;
- "list_labels.yaml" for labels;
- "list_milestones.yaml" for milestones.

Value

invisibly (with invisible()) TRUE if the export was successful and an error otherwise.

Examples

```
all_issues <- get_issues(source = "online", verbose = FALSE)
write_issues_to_dataset(all_issues)

labels <- get_labels(source = "online")
write_labels_to_dataset(labels)

milestones <- get_milestones(source = "online")
```

```
write_milestones_to_dataset(milestones)
```

Index

append, 2

contains, 3, *6*

filter_issues, 6, *16*
format_issues, 7
format_labels, 8
format_milestones, 9

get, *18*
get_issues, 10
get_labels *(*get_issues*)*, 10
get_milestones *(*get_issues*)*, 10
gh, *7–10*
grepl, *3*

new_issue, 12
new_issues, 13

print.IssuesTB *(*print.IssueTB*)*, 14
print.IssueTB, 14

sort.IssuesTB, 15

update_database, 17

vgrepl, *3*

write_issues_to_dataset, 18
write_labels_to_dataset
        *(*write_issues_to_dataset*)*, 18
write_milestones_to_dataset
        *(*write_issues_to_dataset*)*, 18