Package: rmake (via r-universe)

September 6, 2024

Type Package

Title Makefile Generator for R Analytical Projects

Version 1.2.0

Author Michal Burda

Maintainer Michal Burda <michal.burda@osu.cz>

Description Creates and maintains a build process for complex analytic tasks in R. Package allows to easily generate Makefile for the (GNU) 'make' tool, which drives the build process by (in parallel) executing build commands in order to update results accordingly to given dependencies on changed data or updated source files.

License GPL (>= 3.0)

Encoding UTF-8

LazyData true

Imports tools, pryr, assertthat, rmarkdown, visNetwork, knitr

Suggests testthat

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

VignetteBuilder knitr

Repository https://beerda.r-universe.dev

RemoteUrl https://github.com/beerda/rmake

RemoteRef HEAD

RemoteSha 61b615a9dd993501cb54a0f39f4bcaf598b14ff1

Contents

`rmake-package` *Makefile generator for R analytical projects*

Description

rmake creates and maintains a build process for complex analytic tasks in R. Package allows to easily generate Makefile for the (GNU) 'make' tool, which drives the build process by (in parallel) executing build commands in order to update results accordingly to given dependencies on changed data or updated source files.

Details

Note: The package requires the `R_HOME` environment variable to be properly set.

Basic Usage

Suppose you have a file `dataset.csv`. You want to pre-process it and store the results into `dataset.rds` within the `preprocess.R` R script. After that, `dataset.rds` is then an input file for `report.Rmd` and `details.Rmd`, which are R-Markdown scripts that generate `report.pdf` and `details.pdf`. The whole project can be initialized with **rmake** as follows:

1. Let us assume that you have **rmake** package as well as the `make` tool properly installed.
2. Create a new directory (or an R studio project) and copy your `dataset.csv` into it.
3. Load **rmake** package and create skeleton files for it:
   ```
   library(rmake)
   rmakeSkeleton('.')
   ```

   `Makefile.R` and `Makefile` will be created in current directory (`'.'`).

4. Create your file `preprocess.R`, `report.Rmd` and `details.Rmd`.
5. Edit `Makefile.R` as follows:
```
library(rmake)
job <- list(
rRule('dataset.rds', 'preprocess.R', 'dataset.csv'),
markdownRule('report.pdf', 'report.Rmd', 'dataset.rds'),
markdownRule('details.pdf', 'details.Rmd', 'dataset.rds')
)
makefile(job, "Makefile")
```
   This will create three build rules: processing of `preprocess.R` and execution of `report.Rmd` and `details.Rmd` in order to generate resulting PDF files.
6. Run `make` or build your project in R Studio (Build/Build all). This will automatically regenerate `Makefile` and execute `preprocess.R` and the generation of `report.Rmd` and `details.Rmd` accordingly to the changes made to source files.

`defaultVars` *Variables used within Makefile generating process*

Description

`defaultVars` is a reserved variable, a named vector that defines Makefile variables, i.e. shell variables that will exist during the execution of Makefile rules. The content of this variable is written into the resulting Makefile within the execution of the `makefile()` function.

Usage

```
defaultVars
```

Format

An object of class `character` of length 4.

Author(s)

Michal Burda

See Also

`makefile()`

`expandTemplate` *Expand template rules into a list of rules by replacing* `rmake` *variables with their values*

Description

The functionality of `expandTemplate()` differs accordingly to the type of the first argument. If the first argument is a template job (i.e., a list of template rules), or a template rule, then a job is created from templates by replacing `rmake` variables in templates with values of these variables, as specified in the second argument. The `rmake` variable is a part of a string in the format of `$[VARIABLE_NAME]`.

Usage

```
expandTemplate(template, vars)
```

Arguments

`template`	An instance of the S3 `rmake.rule` class, or a list of such objects, or a character vector.
`vars`	A named character vector, matrix, or data frame with variable definitions. For character vector, names are variable names, values are variable values. For matrix or data frame, colnames are variable names and column values are variable values.

Details

If `vars` is a character vector then all variables in `vars` are replaced in `template` so that the result will contain `length(template)` rules. If `vars` is a data frame or a character matrix then the replacement of variables is performed row-wisely. That is, a new sequence of rules is created from `template` for each row of variables in `vars` so that the result will contain `nrow(vars) * length(template)` rules.

If the first argument of `expandTemplate()` is a character vector then the result is a character vector created by row-wise replacements of `rmake` variables, similarly as in the case of template jobs. See examples.

Value

If `template` is an instance of the S3 `rmake.rule` class, or a list of such objects, a list of rules created from `template` by replacing `rmake` variables is returned. If `template` is a character vector then a character vector with all variants of `rmake` values is returned.

Author(s)

Michal Burda

See Also

`replaceVariables()`, `rule()`

Examples

```
# Examples with template jobs and rules:

tmpl <- rRule('data-$[VERSION].csv', 'process-$[TYPE].R', 'output-$[VERSION]-$[TYPE].csv')

job <- expandTemplate(tmpl, c(VERSION='small', TYPE='a'))
# is equivalent to
job <- list(rRule('data-small.csv', 'process-a.R', 'output-small-a.csv'))

job <- expandTemplate(tmpl, expand.grid(VERSION=c('small', 'big'), TYPE=c('a', 'b', 'c')))
# is equivalent to
job <- list(rRule('data-small.csv', 'process-a.R', 'output-small-a.csv'),
            rRule('data-big.csv', 'process-a.R', 'output-big-a.csv'),
            rRule('data-small.csv', 'process-b.R', 'output-small-b.csv'),
            rRule('data-big.csv', 'process-b.R', 'output-big-b.csv'),
            rRule('data-small.csv', 'process-c.R', 'output-small-c.csv'),
            rRule('data-big.csv', 'process-c.R', 'output-big-c.csv'))


# Examples with template character vectors:
expandTemplate('data-$[MAJOR].$[MINOR].csv',
               c(MAJOR=3, MINOR=1))
# returns: c('data-3.1.csv')

expandTemplate('data-$[MAJOR].$[MINOR].csv',
               expand.grid(MAJOR=c(3:4), MINOR=c(0:2)))
# returns: c('data-3.0.csv', 'data-4.0.csv',
#            'data-3.1.csv', 'data-4.1.csv',
#            'data-3.2.csv', 'data-4.2.csv')
```

---

getParam *Wrapper around the* params *global variable*

Description

Returns an element of the global params variable that is normally used to send parameters to a script from the Makefile generated by rmake. Script parameters may be defined with the params argument of the rRule() or markdownRule() functions.

Usage

```
getParam(name, default = NA)
```

Arguments

name	Name of the parameter
default	Default value to be returned if the params global variable does not exist, which typically occurs if the script is executed not from Makefile.

Value

Function returns an element of given `name` from the `params` variable that is created inside of the `Makefile` recipe. If the `params` global variable does not exist (the script is likely to be executed directly, i.e., not from the Makefile generated by `rmake`), the `default` value is returned and a warning is generated. If the `params` global variable exists but it is not a list or the `name` element does not exist there, an error is thrown.

Author(s)

Michal Burda

See Also

`rRule()`, `markdownRule()`

Examples

```
task <- getParam('task', 'default')
```

---

inShell *Convert R code to the character vector of shell commands evaluating the given R code.*

Description

The function takes R commands, deparses them, substitutes existing variables, and converts everything to character strings, from which a shell command is created that sends the given R code to the R interpreter. Function is used internally to print the commands of R rules into `Makefile`.

Usage

```
inShell(...)
```

Arguments

`...`	R commands to be converted

Value

A character vector of shell commands, which send the given R code by pipe to the R interpreter

Author(s)

Michal Burda

See Also

`rRule()`, `markdownRule()`

Examples

```
inShell({
  x <- 1
  y <- 2
  print(x+y)
})
```

`is.rule` *Check if the argument is a valid rule object.*

Description

Function tests whether `x` is a valid rule object, i.e., whether it is list a list and inherits from the `rmake.rule` S3 class. Instances of `rule` represent an atomic building unit, i.e. a command that has to be executed, which optionally depends on some files or other rules – see `rule()` for more details.

Usage

```
is.rule(x)
```

Arguments

`x`	An argument to be tested

Value

`TRUE` if `x` is a valid rule object and `FALSE` otherwise.

Author(s)

Michal Burda

See Also

`rule()`, `makefile()`, `rRule()`, `markdownRule()`, `offlineRule()`

knitrRule *Rule for building text documents by using the knitr package*

Description

This rule is for execution of knitr in order to create the text file, as described in knitr:knit().

Usage

```
knitrRule(target, script, depends = NULL, params = list(), task = "all")
```

Arguments

target	Name of the output file to be created
script	Name of the RNW file to be rendered
depends	A vector of file names that the markdown script depends on, or NULL.
params	A list of R values that become available within the script in a params variable.
task	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from 'all' will cause creation of a new task depending on the given rule. Executing make taskname will then force building of this rule.

Details

This rule executes the following command in a separate R process: library(knitr); params <- params; knitr::knit(sc

That is, parameters given in the params argument are stored into the global variable and then the script is processed with knitr. That is, the re-generation of the Makefile with any change to params will not cause the re-execution of the recipe unless any other script dependencies change.

Issuing make clean from the shell causes removal of all files specified in target parameter.

Value

Instance of S3 class rmake.rule

Author(s)

Michal Burda

See Also

markdownRule(), rule(), makefile(), rRule()

Examples

```
r <- knitrRule(target='report.tex',
               script='report.Rnw',
               depends=c('data1.csv', 'data2.csv'))

# generate the content of a makefile (as character vector)
makefile(list(r))

# generate to file
tmp <- tempdir()
makefile(list(r), file.path(tmp, "Makefile"))
```

---

make *Run 'make" in the system*

Description

This function executes the make command in order to re-build all dependencies, accordingly to Makefile generated by makefile().

Usage

```
make(...)
```

Arguments

...	Command-line arguments passed to the make command

Value

Exit status of the command, see base::system2() for details.

Author(s)

Michal Burda

See Also

makefile(), rmakeSkeleton()

Examples

```
## Not run:
  make()        # make all
  make('clean') # make the 'clean' task
  make('-j', 4) # make with 4 processes in parallell

## End(Not run)
```

makefile *Generate Makefile from given list of rules (*job*).*

Description

In the (GNU) make jargon, *rule* is a sequence of commands to build a result. In this package, rule should be understood similarly: It is a command or a sequence of command that optionally produces some files and depends on some other files (such as data files, scripts) or other rules. Moreover, a rule contain a command for cleanup, i.e. for removal of generated files.

Usage

```
makefile(
  job = list(),
  fileName = NULL,
  makeScript = "Makefile.R",
  vars = NULL,
  all = TRUE,
  tasks = TRUE,
  clean = TRUE,
  makefile = TRUE,
  depends = NULL
)
```

Arguments

job	A list of rules (i.e. of instances of the S3 class rmake.rule - see rule())
fileName	A file to write to. If NULL, the result is returned as a character vector instead of writing to a file.
makeScript	A name of the file that calls this function (in order to generate the makefile rule)
vars	A named character vector of shell variables that will be declared in the resulting Makefile (additionally to [defaultVars])
all	TRUE if the all rule should be automatically created and added: created all rule has dependencies to all the other rules, which causes that everything is built if make all is executed in shell's command line.
tasks	TRUE if "task" rules should be automatically created and added – see rule() for more details.
clean	TRUE if the clean rule should be automatically created and added
makefile	TRUE if the Makefile rule should be automatically created and added: this rule causes that any change in the R script - that generates the Makefile (i.e. that calls makefile()) - issues the re-generation of the Makefile in the beginning of any build.
depends	a character vector of file names that the makefile generating script depends on

Details

The makefile() function takes a list of rules (see rule()) and generates a Makefile from them. Additionally, all and clean rules are optionally generated too, which can be executed from shell by issuing make all or make clean command, respectively, in order to build everything or erase all generated files.

If there is a need to group some rules into a group, it can be done either via dependencies or by using the task mechanism. Each rule may get assigned one or more tasks (see task in rule()). Each task is then created as a standalone rule depending on assigned rules. That way, executing make task_name will build all rules with assigned task task_name. By default, all rules are assigned to task all, which allows make all to build everything.

Value

If fileName is NULL, the function returns a character vector with the contents of the Makefile. Instead, the content is written to the given fileName.

Author(s)

Michal Burda

See Also

rule(), rmakeSkeleton()

Examples

```
# create some jobs
job <- list(
    rRule('dataset.rds', 'preprocess.R', 'dataset.csv'),
    markdownRule('report.pdf', 'report.Rmd', 'dataset.rds'),
    markdownRule('details.pdf', 'details.Rmd', 'dataset.rds'))

# generate Makefile (output as a character vector)
makefile(job)

# generate to file
tmp <- tempdir()
makefile(job, file.path(tmp, "Makefile"))
```

---

markdownRule *Rule for building text documents from Markdown files*

Description

This rule is for execution of Markdown rendering in order to create text file of various supported formats such as (PDF, DOCX, etc.).

Usage

```
markdownRule(target, script, depends = NULL, params = list(), task = "all")
```

Arguments

target	Name of the output file to be created
script	Name of the markdown file to be rendered
depends	A vector of file names that the markdown script depends on, or NULL.
params	A list of R values that become available within the script in a params variable.
task	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from 'all' will cause creation of a new task depending on the given rule. Executing make taskname will then force building of this rule.

Details

This rule executes the following command in a separate R process: params <- params; rmarkdown::render(script, outp

That is, parameters given in the params argument are stored into the global variable and then the script is rendered with rmarkdown. That is, the re-generation of the Makefile with any change to params will not cause the re-execution of the recipe unless any other script dependencies change.

Issuing make clean from the shell causes removal of all files specified in target parameter.

Value

Instance of S3 class rmake.rule

Author(s)

Michal Burda

See Also

rule(), makefile(), rRule()

Examples

```
r <- markdownRule(target='report.pdf',
                  script='report.Rmd',
                  depends=c('data1.csv', 'data2.csv'))

# generate the content of a makefile (as character vector)
makefile(list(r))

# generate to file
tmp <- tempdir()
makefile(list(r), file.path(tmp, "Makefile"))
```

offlineRule *Rule for requesting manual user action*

Description

Instead of building the target, this rule simply issues the given error message. This rule is useful for cases, where the target `target` depends on `depends`, but has to be updated by some manual process. So if `target` is older than any of its dependencies, `make` will throw an error until the user manually updates the target.

Usage

```
offlineRule(target, message, depends = NULL, task = "all")
```

Arguments

`target`	A character vector of target file names of the manual (offline) build command
`message`	An error message to be issued if targets are older than dependencies from `depends`
`depends`	A character vector of file names the targets depend on
`task`	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from `'all'` will cause creation of a new task depending on the given rule. Executing `make taskname` will then force building of this rule.

Value

Instance of S3 class `rmake.rule`

Author(s)

Michal Burda

See Also

`rule()`, `makefile()`

Examples

```
r <- offlineRule(target='offlinedata.csv',
                 message='Please re-generate manually offlinedata.csv',
                 depends=c('source1.csv', 'source2.csv'))

# generate the content of a makefile (as character vector)
makefile(list(r))

# generate to file
tmp <- tempdir()
makefile(list(r), file.path(tmp, "Makefile"))
```

prerequisites *Return given set of properties of all rules in a list*

Description

`targets()` returns a character vector of all unique values of `target` properties, `prerequisites()` returns `depends` and `script` properties, and `tasks()` returns `task` properties of the given `rule()` or list of rules.

Usage

```
prerequisites(x)

targets(x)

tasks(x)

terminals(x)
```

Arguments

`x`	An instance of the `rmake.rule` class or a list of such instances

Details

`terminals()` returns only such targets that are not prerequisites to any other rule.

Value

A character vector of unique values of the selected property obtained from all rules in `x`

Author(s)

Michal Burda

See Also

`rule()`

Examples

```
job <- 'data.csv' %>>%
  rRule('process.R', task='basic') %>>%
  'data.rds' %>>%
  markdownRule('report.Rnw', task='basic') %>>%
  'report.pdf'

prerequisites(job)    # returns c('process.R', data.csv', 'report.Rnw', 'data.rds')
targets(job)          # returns c('data.rds', 'report.pdf')
```

```
tasks(job)           # returns 'basic'
```

`replaceSuffix` *Replace suffix of the given file name with a new extension (suffix)*

Description

This helper function takes a file name `fileName`, removes an extension (a suffix) from it and adds a new extension `newSuffix`.

Usage

```
replaceSuffix(fileName, newSuffix)
```

Arguments

`fileName`	A character vector with original filenames
`newSuffix`	A new extension to replace old extensions in file names `fileName`

Value

A character vector with new file names with old extensions replaced with `newSuffix`

Author(s)

Michal Burda

Examples

```
replaceSuffix('filename.Rmd', '.pdf')          # 'filename.pdf'
replaceSuffix(c('a.x', 'b.y', 'c.z'), '.csv')  # 'a.csv', 'b.csv', 'c.csv'
```

`replaceVariables` *Replace* `rmake` *variables in a character vector*

Description

This function searches for all `rmake` variables in given vector `x` and replaces them with their values that are defined in the `vars` argument. The `rmake` variable is a identified by the `$[VARIABLE_NAME]` string.

Usage

```
replaceVariables(x, vars)
```

Arguments

x	A character vector where to replace the rmake variables
vars	A named character vector with variable definitions (names are variable names, values are variable values)

Value

A character vector with rmake variables replaced with their values

Author(s)

Michal Burda

See Also

expandTemplate()

Examples

```
vars <- c(SIZE='small', METHOD='abc')
replaceVariables('result-$[SIZE]-$[METHOD].csv', vars)  # returns 'result-small-abc.csv'
```

---

rmakeSkeleton *Prepare existing project for building with* rmake.

Description

This function creates a Makefile.R with an empty *rmake* project and generates a basic Makefile from it.

Usage

```
rmakeSkeleton(path)
```

Arguments

path	Path to the target directory where to create files. Use "." for the current directory.

Author(s)

Michal Burda

See Also

makefile(), rule()

Examples

```
# creates/overrides Makefile.R and Makefile in a temporary directory
rmakeSkeleton(path=tempdir())
```

`rRule` *Rule for running R scripts*

Description

This rule is for execution of R scripts in order to create various file outputs.

Usage

```
rRule(
  target,
  script,
  depends = NULL,
  params = list(),
  task = "all",
  preBuild = NULL,
  postBuild = NULL
)
```

Arguments

`target`	Name of output files to be created
`script`	Name of the R script to be executed
`depends`	A vector of file names that the R script depends on, or `NULL`.
`params`	A list of R values that become available within the `script` in a `params` variable.
`task`	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from `'all'` will cause creation of a new task depending on the given rule. Executing `make taskname` will then force building of this rule.
`preBuild`	a character vector of shell commands to be executed before building the target
`postBuild`	a character vector of shell commands to be executed after building the target

Details

In detail, this rule executes the following command in a separate R process: `params <- params; source(script)`

That is, parameters given in the `params` argument are stored into the global variable and then the `script` is sourced. That is, the re-generation of the `Makefile` with any change to `params` will not cause the re-execution of the recipe unless any other script dependencies change.

Issuing `make clean` from the shell causes removal of all files specified in `target` parameter.

Value

Instance of S3 class `rmake.rule`

Author(s)

Michal Burda

See Also

`rule()`, `makefile()`, `markdownRule()`

Examples

```
r <- rRule(target='cleandata.csv',
           script='clean.R',
           depends=c('data1.csv', 'data2.csv'))

# generate the content of a makefile (as character vector)
makefile(list(r))

# generate to file
tmp <- tempdir()
makefile(list(r), file.path(tmp, "Makefile"))
```

---

rule *General creator of an instance of the S3* `rmake.rule` *class*

Description

Rule is an atomic element of the build process. It defines a set of `target` file names, which are to be built with a given `build` command from a given set `depends` of files that targets depend on, and which can be removed by a given `clean` command.

Usage

```
rule(
  target,
  depends = NULL,
  build = NULL,
  clean = NULL,
  task = "all",
  phony = FALSE,
  type = ""
)
```

Arguments

target	A character vector of target file names that are created by the given build command
depends	A character vector of file names the build command depends on
build	A shell command that runs the build of the given target
clean	A shell command that erases all files produced by the build command
task	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from 'all' will cause creation of a new task depending on the given rule. Executing make taskname will then force building of this rule.
phony	Whether the rule has a PHONY (i.e. non-file) target. A rule should be marked with phony if the target is not a file name that would be generated by the build commands. E.g. all or clean are phony targets. Also all targets representing tasks (see task above) are phony.
type	A string representing a type of a rule used e.g. while printing a rule in easily readable format. For instance, rRule() uses R, markdownRule() uses markdown etc.

Details

If there is a need to group some rules together, one can assign them the same task identifier in the task argument. Each rule may get assigned one or more tasks. Tasks may be then built by executing make task_name on the command line, which forces to rebuild all rules assigned to the task 'task_name'. By default, all rules are assigned to task all, which causes make all command to build everything.

Value

Instance of S3 class rmake.rule

Author(s)

Michal Burda

See Also

makefile(), inShell()

Examples

```
r <- rule(target='something.abc',
          depends=c('file.a', 'file.b', 'file.c'),
          build='myCompiler file.a file.b file.c -o something.abc',
          clean='$(RM) something.abc')

# generate the content of a makefile (as character vector)
makefile(list(r))
```

```
# generate to file
tmp <- tempdir()
makefile(list(r), file.path(tmp, "Makefile"))
```

`subdirRule` *Rule for running the make process on a subdirectory*

Description

The subdirectory in the `target` argument is assumed to contain its own `Makefile`. This rule causes the execution of `make <targetTask>` in this subdirectory (where `<targetTask>` is the value of the `targetTask` argument).

Usage

```
subdirRule(target, depends = NULL, task = "all", targetTask = "all")
```

Arguments

`target`	Name of the subdirectory
`depends`	Must be `NULL`
`task`	A character vector of parent task names. The mechanism of tasks allows to group rules. Anything different from `'all'` will cause creation of a new task depending on the given rule. Executing `make taskname` will then force building of this rule.
`targetTask`	What task to execute in the subdirectory.

Value

An instance of S2 classs `rmake.rule`

Author(s)

Michal Burda

See Also

`rule()`, `makefile()`

visualizeRules *Visualize dependencies defined by a rule or a list of rules*

Description

Visualize dependencies defined by a rule or a list of rules

Usage

```
visualizeRules(x, legend = TRUE)
```

Arguments

x	An instance of the S3 rmake.rule class or a list of such objects
legend	Whether to draw a legend

Author(s)

Michal Burda

See Also

makefile(), rule()

Examples

```
job <- c('data1.csv', 'data2.csv') %>>%
  rRule('process.R') %>>%
  'data.rds' %>>%
  markdownRule('report.Rmd') %>>%
  'report.pdf'

## Not run:
visualizeRules(job)

## End(Not run)
```

%>>% *A pipe operator for rmake rules*

Description

This pipe operator simplifies the definition of multiple rmake rules that constitute a chain, that is, if a first rule depends on the results of a second rule, which depends on the results of a third rule and so on.

Usage

```
lhs %>>% rhs
```

Arguments

lhs	A dependency file name or a call to a function that creates a rmake.rule.
rhs	A target file or a call to a function that creates a rmake.rule.
envir	The environment in which to evaluate the arguments of the operator.

Details

The format of proper usage is as follows: 'inFile' %>>% rule() %>>% 'outFile', which is equivalent to the call rule(depends='inFile', target='outFile'). rule must be a function that accepts the named parameters depends and target and creates the rmake.rule object (see rule(), rRule(), markdownRule() etc.). inFile and outFile are file names.

Multiple rules may be pipe-lined as follows: 'inFile' %>>% rRule('script1.R') %>>% 'medFile' %>>% rRule('script2.R') %>>% 'outFile', which is equivalent to a job of two rules created with: rRule(script='script1.R', depends='inFile', target='medFile') and rRule(script='script2.R', depends='medFile', target='outFile').

Value

A list of instances of the rmake.rule class.

Author(s)

Michal Burda (%>>% operator is derived from the code of the magrittr package by Stefan Milton Bache and Hadley Wickham)

See Also

rule(), makefile()

Examples

```
job1 <- 'data.csv' %>>%
  rRule('preprocess.R') %>>%
  'data.rds' %>>%
  markdownRule('report.rnw') %>>%
  'report.pdf'

# is equivalent to

job2 <- list(rRule(target='data.rds', script='preprocess.R', depends='data.csv'),
             markdownRule(target='report.pdf', script='report.rnw', depends='data.rds'))
```

Index