Package ‘makepipe’

January 7, 2025

Title Pipeline Tools Inspired by 'GNU Make'

Version 0.2.2

Description A suite of tools for transforming an existing workflow into a self-documenting pipeline with very minimal upfront costs. Segments of the pipeline are specified in much the same way a 'Make' rule is, by declaring an executable recipe (which might be an R script), along with the corresponding targets and dependencies. When the entire pipeline is run through, only those recipes that need to be executed will be. Meanwhile, execution metadata is captured behind the scenes for later inspection.

License GPL (>= 3)

URL `https://kinto-b.github.io/makepipe/`,
`https://github.com/kinto-b/makepipe`

BugReports `https://github.com/kinto-b/makepipe/issues`

Imports cli, nomnoml, R6, utils, roxygen2

Suggests knitr, covr, testthat (>= 3.0.0), withr, rmarkdown, webshot2, visNetwork,

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

NeedsCompilation no

Author Kinto Behr [aut, cre, cph]

Maintainer Kinto Behr <`kinto.behr@gmail.com`>

Repository CRAN

Date/Publication 2025-01-07 10:30:02 UTC

Contents

---

make_register — *Register objects to be returned from* `make_with_source`

Description

It is sometimes useful to have access to certain objects which are generated as side-products in a source script which yields as a main-product one or more targets. Typically these objects are used for checking that the targets were produced as expected.

Usage

```
make_register(value, name, quiet = FALSE)
```

Arguments

`value`	A value to be registered in a source script and returned as part of the `Segment`
`name`	A variable name, given as a character string. No coercion is done, and the first element of a character vector of length greater than one will be used, with a warning.
`quiet`	A logical determining whether or not warnings are signaled when `make_register()` is called outside of a 'makepipe' pipeline

Value

`value` invisibly

Examples

```
## Not run:
  # Imagine this is part of your source script:
  x <- readRDS("input.Rds")
  x <- do_stuff(x)
  chk <- do_check(x)
  make_register(chk, "x_check")
  saveRDS(x, "output.Rds")

  # You will have access to `chk` in your pipeline script:
  step_one <- make_with_source(
    "source.R",
    "output.Rds",
    "input.Rds",
  )
  step_one$result$chk

## End(Not run)
```

---

make_with_dir *Create a pipeline using roxygen tags*

Description

Instead of maintaining a separate pipeline script containing calls to `make_with_source()`, you can add roxygen-like headers to the .R files in your pipeline containing the `@makepipe` tag along with `@targets`, `@dependencies`, and so on. These tags will be parsed by `make_with_dir()` and used to construct a pipeline. You can call a specific part of the pipeline that has been documented in this way using `make_with_roxy()`.

Usage

```
make_with_dir(
  dir = ".",
  recursive = FALSE,
  build = TRUE,
  envir = new.env(parent = parent.frame()),
  quiet = getOption("makepipe.quiet")
)

make_with_roxy(
  source,
  envir = new.env(parent = parent.frame()),
  quiet = getOption("makepipe.quiet"),
  build = TRUE
)
```

Arguments

`dir`	A character vector of full path names; the default corresponds to the working directory
`recursive`	A logical determining whether or not to recurse into subdirectories
`build`	A logical determining whether or not the pipeline/segment will be built immediately or simply returned to the user
`envir`	The environment in which to execute the `source` or `recipe`. By default, execution will take place in a fresh environment whose parent is the calling environment.

`quiet`	A logical determining whether or not messages are signaled
`source`	The path to an R script which makes the `targets`

Details

Other than `@makepipe`, which is used to tell whether a given script should be included in the pipeline, the tags recognised mirror the arguments to `make_with_source()`. In particular,

- `@targets` and `@dependencies` are for declaring inputs and outputs, the expected format is a comma separated list of strings like `@targets "out1.Rds", "out2.Rds"` but R code like `@targets file.path(DIR, "out.Rds")` (evaluated in `envir`) works too
- `@packages` is for declaring the packages that the targets depend on, the expected format is `@packages pkg1 pkg2 etc`
- `@force` is for declaring whether or not execution should be forced, the expected format is a logical like `TRUE` or `FALSE`

See the getting started vignette for more information.

Value

A `Pipeline` object

See Also

Other make: `make_with_recipe()`, `make_with_source()`

Examples

```
## Not run:
# Create a pipeline from scripts in the working dir without executing it
p <- make_with_dir(build = FALSE)
p$build() # Then execute it yourself

## End(Not run)
```

`make_with_recipe` *Make targets out of dependencies using a recipe*

Description

Make targets out of dependencies using a recipe

Usage

```
make_with_recipe(
  recipe,
  targets,
  dependencies = NULL,
  packages = NULL,
  envir = new.env(parent = parent.frame()),
  quiet = getOption("makepipe.quiet"),
  force = FALSE,
  label = NULL,
  note = NULL,
  build = TRUE,
  ...
)
```

Arguments

`recipe`	A chunk of R code which makes the `targets`
`targets`	A character vector of paths to files
`dependencies`	A character vector of paths to files which the `targets` depend on
`packages`	A character vector of names of packages which `targets` depend on
`envir`	The environment in which to execute the `source` or `recipe`. By default, execution will take place in a fresh environment whose parent is the calling environment.
`quiet`	A logical determining whether or not messages are signaled
`force`	A logical determining whether or not execution of the `source` or `recipe` will be forced (i.e. happen whether or not the targets are out-of-date)
`label`	A short label for the `source` or `recipe`, displayed in pipeline visualisations. If `NULL`, the `basename(source)` or 'Recipe' will be used.
`note`	A description of what the `recipe` does, displayed in pipeline visualisations. If `NULL`, the `recipe` code is used.
`build`	A logical determining whether or not the pipeline/segment will be built immediately or simply returned to the user
`...`	Additional parameters to pass to `base::eval()`

Value

A `Segment` object containing execution metadata.

See Also

Other make: `make_with_dir()`, `make_with_source()`

Examples

```
## Not run:
# Merge files in fresh environment if raw data has been updated since last
# merged
make_with_recipe(
  recipe = {
    dat <- readRDS("data/raw_data.Rds")
    pop <- readRDS("data/pop_data.Rds")
    merged_dat <- merge(dat, pop, by = "id")
    saveRDS(merged_dat, "data/merged_data.Rds")
  },
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds")
)

# Merge files in current environment if raw data has been updated since last
# merged. (If recipe executed, all objects bound in source will be available
# in current env).
make_with_recipe(
  recipe = {
    dat <- readRDS("data/raw_data.Rds")
    pop <- readRDS("data/pop_data.Rds")
    merged_dat <- merge(dat, pop, by = "id")
    saveRDS(merged_dat, "data/merged_data.Rds")
  },
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds"),
  envir = environment()
)

# Merge files in global environment if raw data has been updated since last
# merged. (If source executed, all objects bound in source will be available
# in global env).
make_with_recipe(
  recipe = {
    dat <- readRDS("data/raw_data.Rds")
    pop <- readRDS("data/pop_data.Rds")
    merged_dat <- merge(dat, pop, by = "id")
    saveRDS(merged_dat, "data/merged_data.Rds")
  },
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds"),
  envir = globalenv()
)

## End(Not run)
```

`make_with_source` *Make targets out of dependencies using a source file*

Description

Make targets out of dependencies using a source file

Usage

```
make_with_source(
  source,
  targets,
  dependencies = NULL,
  packages = NULL,
  envir = new.env(parent = parent.frame()),
  quiet = getOption("makepipe.quiet"),
  force = FALSE,
  label = NULL,
  note = NULL,
  build = TRUE,
  ...
)
```

Arguments

`source`	The path to an R script which makes the `targets`
`targets`	A character vector of paths to files
`dependencies`	A character vector of paths to files which the `targets` depend on
`packages`	A character vector of names of packages which `targets` depend on
`envir`	The environment in which to execute the `source` or `recipe`. By default, execution will take place in a fresh environment whose parent is the calling environment.
`quiet`	A logical determining whether or not messages are signaled
`force`	A logical determining whether or not execution of the `source` or `recipe` will be forced (i.e. happen whether or not the targets are out-of-date)
`label`	A short label for the `source` or `recipe`, displayed in pipeline visualisations. If `NULL`, the `basename(source)` or 'Recipe' will be used.
`note`	A description of what the `source` does, displayed in pipeline visualisations
`build`	A logical determining whether or not the pipeline/segment will be built immediately or simply returned to the user
`...`	Additional parameters to pass to `base::source()`

Value

A `Segment` object containing execution metadata.

See Also

Other make: `make_with_dir()`, `make_with_recipe()`

Examples

```
## Not run:
# Merge files in fresh environment if raw data has been updated since last
# merged
make_with_source(
  source = "merge_data.R",
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds")
)


# Merge files in current environment if raw data has been updated since last
# merged. (If source executed, all objects bound in source will be available
# in current env).
make_with_source(
  source = "merge_data.R",
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds"),
  envir = environment()
)


# Merge files in global environment if raw data has been updated since last
# merged. (If source executed, all objects bound in source will be available
# in global env).
make_with_source(
  source = "merge_data.R",
  targets = "data/merged_data.Rds",
  dependencies = c("data/raw_data.Rds", "data/raw_pop.Rds"),
  envir = globalenv()
)

## End(Not run)
```

out_of_date *Check if targets are out-of-date vis-a-vis their dependencies*

Description

Check if targets are out-of-date vis-a-vis their dependencies

Usage

```
out_of_date(targets, dependencies, packages = NULL)
```

Arguments

`targets`	A character vector of paths to files

`dependencies`	A character vector of paths to files which the `targets` depend on
`packages`	A character vector of names of packages which `targets` depend on

Value

`TRUE` if any of `targets` are older than any of `dependencies` or if any of `targets` do not exist; `FALSE` otherwise

Examples

```
## Not run:
out_of_date("data/processed_data.Rds", "data/raw_data.Rds")

## End(Not run)
```

`pipeline-accessors` *Access and interface with Pipeline.*

Description

`get_pipeline()`, `set_pipeline()` and `reset_pipeline()` access and modify the current *active* pipeline, while all other helper functions do not affect the active pipeline

Usage

```
is_pipeline(pipeline)

set_pipeline(pipeline)

get_pipeline()

reset_pipeline()
```

Arguments

`pipeline`	A pipeline. See Pipeline for more details.

See Also

Other pipeline: `Pipeline`, `pipeline-vis`

Examples

```
## Not run:
# Build up a pipeline from scratch and save it out
reset_pipeline()
# A series of `make_with_*()` blocks go here...
saveRDS(get_pipeline(), "data/my_pipeline.Rds")

# ... Later on we can read in and set the pipeline
p <- readRDS("data/my_pipeline.Rds")
set_pipeline(p)

## End(Not run)
```

`pipeline-vis` *Visualise the Pipeline.*

Description

Produce a flowchart visualisation of the pipeline. Out-of-date targets will be coloured red, up-to-date targets will be coloured green, and everything else will be blue.

Usage

```
show_pipeline(
  pipeline = get_pipeline(),
  as = c("nomnoml", "visnetwork", "text"),
  labels = NULL,
  notes = NULL,
  ...
)

save_pipeline(
  file,
  pipeline = get_pipeline(),
  as = c("nomnoml", "visnetwork", "text"),
  labels = NULL,
  notes = NULL,
  ...
)
```

Arguments

`pipeline`	A pipeline. See Pipeline for more details.
`as`	A string determining whether to use `nomnoml` or `visNetwork`
`labels`	A named character vector mapping nodes in the `pipeline` onto labels to display beside them.

`notes`	A named character vector mapping nodes in the `Pipeline` onto notes to display on beside the labels (nomnoml) or as tooltips (visNetwork).
`...`	Arguments passed onto `Pipeline$nomnoml()` or `Pipeline$visnetwork`
`file`	File to save png (nomnoml) or html (visnetwork) into

Details

Labels and notes must be supplied as named character vector where the names correspond to the filepaths of nodes (i.e. `targets`, `dependencies`, or `source` scripts)

See Also

Other pipeline: `Pipeline`, `pipeline-accessors`

Examples

```
## Not run:
# Run pipeline
make_with_source(
  "recode.R",
  "data/0 raw_data.R",
  "data/1 data.R"
)
make_with_source(
  "merge.R",
  c("data/1 data.R", "data/0 raw_pop.R"),
  "data/2 data.R"
)

# Visualise pipeline with custom notes
show_pipeline(notes = c(
  "data/0 raw_data.R" = "Raw survey data",
  "data/0 raw_pop.R" = "Raw population data",
  "data/1 data.R" = "Survey data with recodes applied",
  "data/2 data.R" = "Survey data with demographic variables merged in"
))

## End(Not run)
```

Index