

Package: pippy (via r-universe)

October 14, 2026

Type Package

Title Orchestrate Functional Workflows

Version 0.0.0.1

Description Make it easy for users to adopt a functional approach to their analysis. Pipelines are defined using a reduced subset of R and, when run, use an intelligent approach to work planning and object caching. Whilst the goal is to be ``good enough" for many pieces of analysis, by encouraging a functional approach, it remains easy to move to more advanced alternatives such as targets (<[doi:10.32614/CRAN.package.targets](https://doi.org/10.32614/CRAN.package.targets)>).

URL <https://timtaylor.codeberg.page/pippy/>

BugReports <https://codeberg.org/TimTaylor/pippy/issues>

License GPL-2 | GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports cli, datasets, dplyr, globals, igraph, rlang (>= 1.3.0), stats, tools, utils, vctrs

Suggests compiler, fs, knitr, quarto, rmarkdown, testthat (>= 3.0.0)

Depends R (>= 4.5.0)

Config/testthat/edition 3

Config/testthat/load-all list(export_all = FALSE, helpers = FALSE)

Config/testthat/parallel true

Config/roxygen2/version 8.1.1

VignetteBuilder quarto

Config/pak/sysreqs libgmp-dev libxml2-dev

Repository <https://timtaylor.r-universe.dev>

Date/Publication 2026-10-14 13:57:28 UTC

RemoteUrl <https://codeberg.org/TimTaylor/pippy>

RemoteRef HEAD

RemoteSha b9d048bcb965f48cbdf841cc9619ec700178e06a
RemoteSubdir pkg

Contents

create_skeleton_pipeline	2
load_pipeline_object	3
run_pipeline	4
Index	8

create_skeleton_pipeline
<i>Create a demo pipeline</i>

Description

Sets up an example pipeline which can be used as the basis for your own.

Usage

```
create_skeleton_pipeline(dir = "DemoPipeline")
```

Arguments

dir Directory you wish to create the pipeline in.

Value

Absolute path of the created pipeline directory (invisibly).

Examples

```
dir <- create_skeleton_pipeline(tempfile())

# Note the configuration file and the folder of R functions
list.files(dir, all.files = TRUE, recursive = TRUE, no.. = TRUE)

unlink(dir)
```

load_pipeline_object *Load an object from the cache*

Description

Load an object from the cache

Usage

```
load_pipeline_object(x, ..., scenario = NULL, pipeline_dir = ".")
```

Arguments

x	[character] Object(s) to load. Corresponds to the target(s) of an already run pipeline.
...	Not currently used.
scenario	[character] Which scenarios to consider. If NULL, objects from all scenarios present in the cache are returned.
pipeline_dir	[character] The main pipeline directory (where the .pipeline_cache is located).

Value

A nested list of requested scenarios and objects.

Examples

```
# generate a demo pipeline with a single scenario
dir <- create_skeleton_pipeline(tempfile())

# run with the 'default' and 'production' configuration but don't return
out <- run_pipeline(
  {
    raw  <- load_dat(CONFIG$in_csv)
    clean <- wrangle_dat(raw, CONFIG$rows)
    plot <- plot_dat(clean, CONFIG$out_plot)
  },
  pipeline_dir = dir,
  scenario = c("default", "production"),
  return = FALSE
)

str(out)

# Loads both default and production
load_pipeline_object("clean", pipeline_dir = dir)
```

```
# Loads just the production object
load_pipeline_object("clean", scenario = "production", pipeline_dir = dir)

unlink(dir)
```

run_pipeline

Run a pipeline

Description

run_pipeline() manages the running of a user defined pipeline across multiple scenarios. Different stages of analysis are defined as functions which are then brought together as a pipeline using a syntax that is a reduced subset of R. This pipeline is then represented as a directed acyclic graph allowing us to employ a robust approach to work planning and caching.

Usage

```
run_pipeline(
  x,
  ...,
  scenario = getOption("pipeline.scenario_default", "default"),
  scenario_default = getOption("pipeline.scenario_default", "default"),
  pipeline_dir = ".",
  relative_config_file = getOption("pipeline.config_file", "config.R"),
  relative_function_dir = getOption("pipeline.function_dir", "R"),
  relative_output_dir = getOption("pipeline.output_dir", "output"),
  force = FALSE,
  saveRDS_args = list(),
  readRDS_args = list(),
  return = TRUE,
  gc = FALSE,
  optimise_for_memory = FALSE
)
```

Arguments

x	[expression] R Expression of pipeline assignments. See details for information on the supported syntax.
...	Not currently used.
scenario	[character] Scenario(s) to run on top of scenario_default. These represent the file configurations you wish to loop over. In this case the configuration file will first be parsed looking for a named-list entry corresponding to the value of the scenario_default argument. The chosen scenarios are

then layered on top of this default using `utils::modifyList()`. The default scenario is always run first.

scenario_default	[character] The default scenario to consider. That is, the scenario on which the specified scenarios are layered. If no configuration file exists this will represent the folder under the <code>output_dir</code> where file output and cached objects are stored as well as the element of the resulting list output.
pipeline_dir	[character] The directory you wish to run the the pipeline relative to.
relative_config_file	[character] The configuration file. Must be a none-nested and given relative to <code>pipeline_dir</code> . See details for information on the configuration file format.
relative_function_dir	[character] The directory to look for user defined pipeline functions. Must be a none-nested and given relative to <code>pipeline_dir</code> .
relative_output_dir	[character] The directory for output. Must be a none-nested and given relative to <code>pipeline_dir</code> .
force	[bool] Do you want to force a run of the pipeline. If TRUE, then cached objects are removed and the pipeline is (re)run.
saveRDS_args	[list] List of additional arguments passed to <code>saveRDS()</code> . This argument allows you to pass additional arguments to that function (e.g. <code>list(compress = "zstd")</code>)
readRDS_args	[list] List of additional arguments passed to <code>readRDS()</code> .
return	[bool] Should the output be returned. Defaults to TRUE, but when running across multiple scenarios and with outputs that use a large amount of memory it can be useful to set FALSE. If FALSE, a successful run will return NULL invisibly.
gc	[bool] Should we force calls to <code>gc()</code> whilst running the pipeline. For pipeline's creating large objects, setting this to TRUE <i>may</i> help reduce memory consumption.
optimise_for_memory	[bool]

If TRUE then when a target is no longer needed in the calculation of subsequent targets they are removed from memory and a forced gc is called.

When the objects are needed by subsequent scenarios they will be reloaded from disk.

This can only be TRUE when return = FALSE.

Details

Normally a pipeline expression will involve multiple assignments need embracing to represent a single expression. The syntax has a few additional requirements:

- The left-hand-side (target) of each assignment must be uniquely named.
- Function calls on the right-hand-side (rhs) must be defined in the function directory (see `relative_function_dir`) or contained in the base R or stats namespaces.
- The target names cannot match the names of functions in base R, or those contained in the function directory.
- The order of assignments is not important as it is determined at runtime.
- Values from the configuration file (see `relative_config_file`) are accessed by the special CONFIG value (a list which can be subset using the \$ symbol).
- For serialising objects to disk we utilise R's rds format (with the ability to pass arguments through via `saveRDS_args`). This means that functions seen on the RHS of the expressions should only return objects that are unserialisable by `readRDS()`.

Configuration files are simply R scripts where scenarios are defined by named lists. They are evaluated in a dedicated environment where we enable two special functions:

- `INFILE()` marks a file as an input. This ensures that `run_pipeline()` knows to keep a hash of the object and to monitor it for future changes when the pipeline is rerun.
- `OUTFILE()` marks a file name as an output. This ensures that outputs from different scenarios (see later) do not overwrite one another (output files get saved in a scenario-dependent directory). It also ensures that output files are recreated if they are deleted.

Value

A named list of outputs for each configuration.

Caveats

- When working out which stages of a pipeline need rerunning we take in to account functions defined by the user in the function directory. We do not account for functions of called packages or base R and stats. This means that updating packages (or R itself) can potentially invalidate the cache so it is the users responsibility to maintain a consistent environment, or force a rerun of the pipeline where necessary. Future versions of pippy will warn when underlying package versions (or R itself) change between runs.
- The only global object we account for is `.Random.seed`. If you wish to take advantage of object caching then it is important to set a seed before running a pipeline. This seed will then be utilised at the start of each scenario run.

Examples

```
# generate a demo pipeline with a single scenario
dir <- create_skeleton_pipeline(tempfile())

# Note the configuration file and the folder of R functions
list.files(dir, all.files = TRUE, recursive = TRUE, no.. = TRUE)

# Run the pipeline with the default configuration (see config.R)
out <- run_pipeline(
  {
    raw  <- load_dat(CONFIG$in_csv)
    clean <- wrangle_dat(raw, CONFIG$rows)
    plot <- plot_dat(clean, CONFIG$out_plot)
  },
  pipeline_dir = dir,
  scenario = "default"
)

# output in a list
str(out$default$raw)
str(out$default$clean)
out$default$plot

# run with the 'production' configuration as well (see config.R)
out <- run_pipeline(
  {
    raw  <- load_dat(CONFIG$in_csv)
    clean <- wrangle_dat(raw, CONFIG$rows)
    plot <- plot_dat(clean, CONFIG$out_plot)
  },
  pipeline_dir = dir,
  scenario = c("default", "production")
)

unlink(dir)
```

Index

`create_skeleton_pipeline`, [2](#)

`load_pipeline_object`, [3](#)

`run_pipeline`, [4](#)