Package: command (via r-universe)

June 1, 2026

Type Package

Title Process Command Line Arguments

Version 0.1.4

Description Process command line arguments, as part of a data analysis workflow. 'command' makes it easier to construct a workflow consisting of lots of small, self-contained scripts, all run from a Makefile or shell script. The aim is a workflow that is modular, transparent, and reliable.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports cli, fs, methods, tools

Suggests covr, dplyr, ggplot2, knitr, littler, quarto, readr, rmarkdown, testthat (>= 3.0.0), tidyr, withr

Config/testthat/edition 3

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

URL https://bayesiandemography.github.io/command/, https://github.com/bayesiandemography/command

Config/Needs/website quarto, rmarkdown

BugReports https://github.com/bayesiandemography/command/issues

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-01 03:43:40 UTC

RemoteUrl https://github.com/bayesiandemography/command

RemoteRef HEAD

RemoteSha 21a4a24215e88274e8173661d7fad35967b723fc

Contents

`cmd_assign` *Assign Values Passed at the Command Line or Interactively*

Description

Assign values to names in the working environment. The values are typically supplied through the command line, but can be supplied interactively.

Specifying the inputs and outputs of scripts through the command line can contribute to safter, more modular workflows.

`cmd_assign_quiet()` is identical to `cmd_assign()`, but does not print progress messages to the console.

Usage

```
cmd_assign(...)

cmd_assign_quiet(...)
```

Arguments

...	Name-value pairs.

Value

`cmd_assign()` is called for its side effect, which is to create objects in the global environment. However, `cmd_assign()` also invisibly returns a named list of objects.

Types of session

`cmd_assign()` behaves differently depending on how it whether it is called

1. interactively, or
2. inside an R script that is run from the command line.

For instance, if the code

`cmd_assign(use_log = TRUE)`

is run interactively, it creates an object called `use_log` with value `TRUE`.

But if the same code is run inside a script via the command
`Rscript tidy_data.R --use_log=FALSE`

it creates an object called `use_log` with value `FALSE`.

`cmd_assign()` is typically called interactively when a workflow is being developed, and through the command line when the workflow has matured.

Matching names and values

When used in a script called from the command line, `cmd_assign()` first matches named command line arguments, and then matches unnamed command line arguments, in the order in which they are supplied.

If, for instance, the script `person.R` contains the lines

```
cmd_assign(.data = "raw_data.csv",
           max_age = 85,
           .out = "person.rds")
```

and if `person.R` is run from the command line using

```
Rscript person.R raw_data.csv person.rds --max_age=100
```

then `cmd_assign()` first matches named command line argument `--max_age=100` to `cmd_assign() argument max_age, ar`

Coercing values passed at the command line

Values passed at the command line start out as text strings. `cmd_assign()` coerces these text strings to have the same class as the corresponding values in the call to `cmd_assign()`. For instance, if a script called `fit.R` contains the lines

```
cmd_assign(.data = "cleaned.rds",
           impute = TRUE,
           date = as.Date("2026-01-01"),
           .out = "fit.rds")
```

and if `fitted.R` is run from the command line using

```
Rscript fitted.R cleaned.rds fit.rds --impute=TRUE --date=2025-01-01
```

then `cmd_assign()` will create

- a character vector called `.data` with value "cleaned.rds",
- a logical vector called `impute` with value `TRUE`,
- a date vector called `date` with value `"2025-01-01"`, and
- a character vector called `.out` with value ‘"fit.rds".

References

- Command-Line Programs Introduction to Rscript

See Also

- extract_shell() Turn a cmd_assign() call into a shell command
- extract_make() Turn a cmd_assign() call into a Makefile rule
- shell_script() Create a shell script
- makefile() Create a Makefile
- Quick Start How to use cmd_assign()
- Modular Workflows for Data Analysis Safe, flexible data analysis workflows.
- Base R function commandArgs() uses a more general, lower-level approach to processing command line arguments. (commandArgs() is called internally by cmd_assign().)
- littler Alternative to Rscript

Examples

```
if (interactive()) {
  cmd_assign(.data = "mydata.csv",
             n_iter = 2000,
             .out = "results.rds")
}
```

---

extract_make *Turn a 'cmd_assign' Call Into a Makefile Rule*

Description

Extract a call to cmd_assign() from an R script, and turn it into a Makefile rule.

Usage

```
extract_make(path_file, dir_make = NULL)
```

Arguments

path_file	A path from dir_make to the R scripe containing the call to cmd_assign().
dir_make	The directory that contains the Makefile. The default is the current working directory.

Value

extract_make() is typically called for its side effect, which is to print a Makefile rule. However, extract_make() invisibly returns a text string with the rule.

The components of a Makefile rule

A Makefile rule produced by `extract_make()` normally looks something like this:

```
out/model.rds: src/model.R \
  data/cleaned.rds
      Rscript $^ $@ --use_log=TRUE
```

In this rule

- `out/model.rds` is the "target", i.e. the file that the rule creates;
- `src/model.R` and `data/timeseries.rds` are "prerequisites", i.e. files that are used to create the target;
- `\` is a "line continuation character";
- ` ` at the start of the third line is a tab, telling `make` that the recipe for creating the target from the starts here;
- `Rscript` is a call to `utils::Rscript()`;
- `$^` is an automatic variable meaning "all the prerequisites" and `$@` is an automatic variable meaning "the target", so that `Rscript $^ $@` expands to `Rscript src/model.R data/cleaned.rds out/model.rds`; and
- `--use_log=TRUE` is a named argument that Rscript passes to `src/model.R`

Using `extract_make()` to build a data analysis workflow

- Step 1. Write the R file that carries out the step in analysis (eg tidying data, fitting a model, making a graph.) This file will contain a call to `cmd_assign()`, and will appear in the first line of the Makefile rule. When writing and testing the file, use `cmd_assign()` interactively.
- Step 2. Once the R file is working correctly, call `extract_make()`, and add the rule to your Makefile.

When using `extract_make()`, it is a good idea to set the current working directory to the project directory (something that will happen automatically if you are using RStudio projects.)

Location of the Makefile

The Makefile normally sets at the top of the project, so that the project folder looks something like this:

```
Makefile
- data/
- src/
- out/
report.qmd
```

Identifying file arguments

To construct the Makefile rule, `extract_make()` needs to be able to pick out arguments that refer to file names. To do so, it uses the following heuristic:

- if the call includes arguments whose names start with a dot, then these arguments are assumed to refer to file names;
- otherwise, find arguments whose values actually are file names (as determined by `file.exists()`), or that look like they could be.

References

- Project Management with Make Makefiles in data analysis workflows
- GNU make Definitive guide
- Command-Line Programs Introduction to Rscript

See Also

- `extract_shell()` Shell script equivalent of `extract_make()`
- `makefile()` Create a Makefile from calls to `cmd_assign()`
- `cmd_assign()` Process command line arguments
- Quick Start How to use `cmd_assign()`
- Modular Workflows for Data Analysis Safe, flexible data analysis workflows
- littler Alternative to Rscript

Examples

```
library(fs)
library(withr)
with_tempdir({

  ## create 'src'  directory
  dir_create("src")

  ## put an R script containing a call to
  ## 'cmd_assign' in the 'src' directory
  writeLines(c("cmd_assign(x = 1, .out = 'out/results.rds')",
               "results <- x + 1",
               "saveRDS(results, file = .out)"),
             con = "src/results.R")

  ## call 'extract_make()'
  extract_make(path_file = "src/results.R",
               dir_make = ".")

})
```

`extract_shell` *Turn a 'cmd_assign' Call Into a Shell Command*

Description

Extract a call to `cmd_assign()` from an R script, and turn it into a shell command.

Usage

```
extract_shell(path_file, dir_shell = NULL)
```

Arguments

`path_file`	Path to the R script containing the call to `cmd_assign()`. The path starts at `dir_shell`.
`dir_shell`	The directory that contains the shell script. The default is the current working directory.

Value

`extract_shell()` is typically called for its side effect, which is to print a shell command. However, `extract_shell()` invisibly returns a text string with the command.

The components of a shell command

The shell command produced by `extract_shell()` normally looks something like this:

```
Rscript src/model.R \
  data/cleaned.rds \
  out/model.rds \
  --use_log=TRUE
```

In this command

- `Rscript` is a call to `utils::Rscript()`;
- `\` is a "line continuation character";
- `data/cleaned.rds` and `out/model.rds` are unnamed arguments that Rscript passes to `src/model.R`; and
- `--use_log=TRUE` is a named argument that Rscript passes to `src/model.R`

Using `extract_shell()` to build a data analysis workflow

- Step 1. Write an R script that carries out a step in analysis (eg tidying data, fitting a model, making a graph.) This script will contain a call to `cmd_assign()`, and will be the first argument passed to Rscript in the shell command. When writing and testing the script, use `cmd_assign()` interactively.
- Step 2. Once the R script is working correctly, call `extract_shell()`, and add the command to your shell script.

Location of the shell script

The shell script normally sits at the top level of the project, so that the project folder looks something like this:

```
workflow.sh
- data/
- src/
- out/
report.qmd
```

Identifying file arguments

To construct the rule, `extract_shell()` needs to be able to identify arguments that refer to a file name. To do so, it uses the following heuristic:

- if the call includes arguments whose names start with a dot, then these arguments are assumed to refer to file names;
- otherwise, find arguments whose values actually are file names (as determined by `file.exists()`) or that look like they could be.

References

- Episodes 1–3 of The Unix Shell Introduction to the command line
- Command-Line Programs Introduction to Rscript
- littler Alternative to Rscript

See Also

- `extract_make()` Makefile equivalent of `extract_shell()`
- `shell_script()` Create a shell script from calls to `cmd_assign()`
- `cmd_assign()` Process command line arguments
- Quick Start How to use `cmd_assign()`
- Modular Workflows for Data Analysis Safe, flexible data analysis workflows

Examples

```
library(fs)
library(withr)

with_tempdir({

  ## create 'src' directory
  dir_create("src")

  ## add an R script containing a call to 'cmd_assign'
  writeLines(c("cmd_assign(x = 1, .out = 'out/results.rds')",
               "results <- x + 1",
               "saveRDS(results, file = .out)"),
```

```
            con = "src/results.R")

  ## call 'extract_shell()'
  extract_shell(path_file = "src/results.R",
                dir_shell = ".")

})
```

`makefile` *Create a Makefile*

Description

Create a Makefile for a data analysis workflow. The Makefile can include rules extracted from existing R files.

Usage

```
makefile(
  path_files = NULL,
  dir_make = NULL,
  name_make = "Makefile",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`path_files`	A path from `dir_make` to a directory with R scripts containing calls to `cmd_assign()`. Optional.
`dir_make`	The directory where `makefile()` will create the Makefile. If no value is supplied, then 'makefile(); creates the Makefile the current working directory.
`name_make`	The name of the Makefile. The default is `"Makefile"`.
`overwrite`	Whether to overwrite an existing Makefile. Default is `FALSE`.
`quiet`	Whether to suppress progress messages. Default is `FALSE`.

Details

To create a Makefile in the `files` directory, set `files` to `"."`.

To obtain the contents of the Makefile without creating a file on disk, creating the file on disk, set `name_make` to `NULL`.

Supplying a value for `files` is optional for `makefile()`, but compulsory for `shell_script()`. The output from `makefile()` includes some general-purpose Makefile commands, while the output from `shell_script()` is generated entirely from `files`.

Value

`makefile()` is called for its side effect, which is to create a file. However, `makefile()` also returns a string with the contents of the Makefile.

References

- Project Management with Make Makefiles in data analysis workflows
- GNU make Definitive guide
- Command-Line Programs Introduction to Rscript

See Also

- Creating a Makefile More on `makefile()`
- `extract_make()` Turn a `cmd_assign()` call into a Makefile rule
- `shell_script()` Shell script equivalent of `makefile()`
- `cmd_assign()` Process command line arguments
- Modular Workflows for Data Analysis Safe, flexible data analysis workflows
- littler Alternative to Rscript

Examples

```
library(fs)
library(withr)

with_tempdir({

  ## create 'src'  directory
  dir_create("src")

  ## put R scripts containing calls to
  ## 'cmd_assign' in the 'src' directory
  writeLines(c("cmd_assign(x = 1, .out = 'out/results.rds')",
               "results <- x + 1",
               "saveRDS(results, file = .out)"),
             con = "src/results.R")
  writeLines(c("cmd_assign(x = 1, .out = 'out/more_results.rds')",
               "more_results <- x + 2",
               "saveRDS(more_results, file = .out)"),
             con = "src/more_results.R")

  ## call 'makefile()'
  makefile(path_files = "src",
           dir_make = ".")

  ## Makefile has been created
  dir_tree()

  ## print contents of Makefile
  cat(readLines("Makefile"), sep = "\n")
```

```
})
```

`shell_script` *Create a Shell Script*

Description

Create a shell script for a data analysis workflow consisting of commands extracted from existing R files.

Usage

```
shell_script(
  path_files,
  dir_shell = NULL,
  name_shell = "workflow.sh",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`path_files`	A path from `dir_shell` to a directory with R scripts containing calls to `cmd_assign()`.
`dir_shell`	The directory where `shell_script()` will create the shell script. If no value is supplied, then `shell_script()` creates the shell script in the current working directory.
`name_shell`	The name of the shell script. The default is `"workflow.sh"`.
`overwrite`	Whether to overwrite an existing shell script. Default is `FALSE`.
`quiet`	Whether to suppress progress messages. Default is `FALSE`.

Details

To create a shell script in the `files` directory, set `files` to `"."`.

To obtain the contents of the shell script without creating a file on disk, creating the file on disk, set `name_shell` to `NULL`.

Supplying a value for `files` is compulsory for `shell_script()`, but optional for `makefile()`. The output from `shell_script()` is generated entirely from `files` while the output from `makefile()` also includes some general-purpose Makefile commands.

Value

`shell_script()` is called for its side effect, which is to create a file. However, `shell_script()` also returns a string with the contents of the shell script.

References

- Episodes 1–3 of The Unix Shell Introduction to the command line
- Command-Line Programs Introduction to Rscript

See Also

- Creating a Shell Script More on `shell_script()`
- `extract_shell()` Turn a `cmd_assign()` call into a shell command
- `makefile()` Makefile equivalent of `shell_script()`
- `cmd_assign()` Process command line arguments
- Modular Workflows for Data Analysis Safe, flexible data analysis workflows
- littler Alternative to Rscript

Examples

```
library(fs)
library(withr)

with_tempdir({

  ## create 'src'  directory
  dir_create("src")

  ## put R scripts containing calls to
  ## 'cmd_assign' in the 'src' directory
  writeLines(c("cmd_assign(x = 1, .out = 'out/results.rds')",
               "results <- x + 1",
               "saveRDS(results, file = .out)"),
             con = "src/results.R")
  writeLines(c("cmd_assign(x = 1, .out = 'out/more_results.rds')",
               "more_results <- x + 2",
               "saveRDS(more_results, file = .out)"),
             con = "src/more_results.R")

  ## call 'shell_script()'
  shell_script(path_files = "src",
               dir_shell = ".")

  ## shell script has been created
  dir_tree()

  ## print contents of shell script
  cat(readLines("workflow.sh"), sep = "\n")

})
```

Index