

Package ‘command’

October 17, 2026

Type Package

Title Process Command Line Arguments

Version 0.3.0

Description Process command line arguments. Processing includes checking, naming, and coercing to numeric, date, or logical. By capturing assumptions about inputs and outputs, the 'command' package can provide scripts with well defined interfaces, leading to safer, more modular data analysis pipelines.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.5.0)

Imports cli, methods, tools

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

Config/testthat/edition 3

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

Config/Needs/website quarto, rmarkdown, fs

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

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author John Bryant [aut, cre],
Bayesian Demography Limited [cph]

Maintainer John Bryant <john@bayesiandemography.com>

Repository CRAN

Date/Publication 2026-10-17 06:10:02 UTC

Contents

command-package	2
cmd_assign	3
cmd_assign_collect	6
extract_make	8
extract_shell	11
makefile	13
shell_script	15
use_renv	17
Index	18

command-package	<i>command: Process command line arguments</i>
-----------------	--

Description

Process command line arguments, allowing scripts to behave like functions, with well-defined inputs and outputs. Helps make data analysis workflows more modular, and therefore more transparent, flexible, and reliable.

Details

- [cmd_assign\(\)](#) Process command line arguments
- [cmd_assign_collect\(\)](#) Collect unnamed command line arguments into one character vector
- [use_renv\(\)](#) Activate a renv project before loading packages
- [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
- [Using command with renv](#) Running pipeline scripts with renv

Author(s)

Maintainer: John Bryant <john@bayesiandemography.com>

Authors:

- John Bryant <john@bayesiandemography.com>

Other contributors:

- Bayesian Demography Limited [copyright holder]

See Also

Useful links:

- <https://bayesiandemography.github.io/command/>
- <https://github.com/bayesiandemography/command>
- Report bugs at <https://github.com/bayesiandemography/command/issues>

cmd_assign

Assign Values Passed at the Command Line or Interactively

Description

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

cmd_assign_quiet() is identical to cmd_assign() except that it does not print progress messages to the console.

Usage

```
cmd_assign(...)
```

```
cmd_assign_quiet(...)
```

Arguments

... Name-value pairs.

Details

cmd_assign() is a user-friendly alternative to `commandArgs()`, designed specifically for data analysis workflows.

cmd_assign() can be used to record and enforce assumptions about inputs and outputs. This provides scripts with better-defined interfaces, making them clearer and safer.

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 whether it is called interactively, or from within an R script being 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 from within a script that has been invoked using 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 is in production.

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()` assigns `cmd_assign()` names to command line arguments in the following order:

order	cmd_assign() name	command line argument
1	<code>max_age</code>	<code>-max_age=100</code>
2	<code>.data</code>	<code>raw_data.csv</code>
3	<code>.out</code>	<code>person.rds</code>

The values specified in the call to `cmd_assign()` are not defaults

The values specified in the call to `cmd_assign()` are not defaults in the sense of values to fall back on when no value is supplied. For instance, if we take a script containing the call

```
cmd_assign(
  n_draw = 10,
  .out = "simulation.rds"
)
```

and run it from the command line without passing a value for `n_draw`, then `cmd_assign()` will throw an error. It will *not* set `n_draw` to 10.

Defaults can hide bugs and make code less predictable.

Coercing values passed at the command line

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

```
cmd_assign(
  .data = "data/individuals.rds",
  impute = TRUE,
  date = as.Date("2025-01-01"),
  .out = "output/fitted.rds"
)
```

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

```
Rscript src/fit.R \
  data/individuals.rds \
  output/fitted.rds \
  --impute=TRUE \
  --date=2025-01-01
```

then `cmd_assign()` will create the following length-1 vectors

name	class	value
<code>.data</code>	"character"	"data/individuals.rds"
<code>impute</code>	"logical"	TRUE
<code>date</code>	"Date"	"2025-01-01"
<code>.out</code>	"character"	"output/fitted.rds"

The dot-name convention

When naming arguments to `cmd_assign()`, it is typically good practice to use names starting with dots for file paths, and names without dots for ordinary objects, as in

```
cmd_assign(
  .households = "data/households.csv",
  n_record = 500,
  sex = "Female",
  .out = "out/subsample_female.rds"
)
```

Function `cmd_assign()` does not in fact distinguish between names with dots and names without dots. But the dot-name convention is still useful because of the way it calls attention to the distinction between file paths and ordinary objects. For instance, a dot-name is a reminder that the object being named is a reference to an input or output, rather than the input or output itself.

In contrast, functions `extract_make()`, `extract_shell()`, `makefile()`, and `shell_script()` do treat names with dots differently from names without dots. See the help for `extract_make()`, `extract_shell()`, `makefile()`, and `shell_script()` for details.

References

- [Command-Line Programs](#) Introduction to Rscript

See Also

- `cmd_assign_collect()` Collect unnamed command line arguments into one character vector
- `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
- `use_renv()` Activate a renv project before loading packages
- [Quick Start](#) How to use `cmd_assign()`
- [Modular Workflows for Data Analysis](#) Safe, flexible data analysis workflows.
- [Using command with renv](#) Running pipeline scripts with renv
- 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")
}
```

<code>cmd_assign_collect</code>	<i>Assign Varying Set of Values Passed at the Command Line or Interactively</i>
---------------------------------	---

Description

Assign names to values in the global environment, like `cmd_assign()`, but without specifying the number of values in advance. Values for unnamed command line arguments are collected into a single character vector.

`cmd_assign_collect_quiet()` is identical to `cmd_assign_collect()` except that it does not print progress messages to the console.

Usage

```
cmd_assign_collect(...)

cmd_assign_collect_quiet(...)
```

Arguments

... Name-value pairs.

Value

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

Matching names and values

cmd_assign_collect(), like [cmd_assign\(\)](#), starts by processing named command line arguments (i.e. arguments of the form --n_draw=100). Unlike cmd_assign(), however, cmd_assign_collect() puts all the values that are left over into a single character vector. This vector can be any length, including zero. cmd_assign_collect() must contain a single name-value pair that is not used for named command line arguments. cmd_assign_collect() assigns the name from this name-value to the character vector.

If, for instance, the script combine.R contains

```
cmd_assign_collect(
  .files = c("data/a.csv", "data/b.csv", "out/combined.rds"),
  use_log = TRUE
)
```

and if combine.R is run from the command line using

```
Rscript combine.R \
  data/a.csv \
  data/c.csv \
  out/combined.rds \
  --use_log=TRUE
```

then use_log will be assigned the value TRUE and .files will be assigned the value c("data/a.csv", "data/c.csv", "out/combined.rds").

Processing the vector of unnamed arguments

The R script in which cmd_assign_collect() is called has responsibility for processing the character vector of values from unnamed command line arguments. The call to cmd_assign_collect() above, for instance, might be followed by statements such as

```
.inputs <- head(.files, -1)
.out <- tail(.files, 1)
inputs <- lapply(.inputs, read.csv)
```

Use `cmd_assign()` where possible

`cmd_assign_collect()` is more flexible than `cmd_assign()`, in that it does not require the number, name, or type of values to be specified in advance. But if these things are in fact known in advance, then it is good practice to document and test for them, which is what `cmd_assign()` does.

See Also

- `cmd_assign()` Process command line arguments with one value per argument
- `extract_shell()` Turn a `cmd_assign()` or `cmd_assign_collect()` call into a shell command
- `extract_make()` Turn a `cmd_assign()` or `cmd_assign_collect()` call into a Makefile rule
- `shell_script()` Create a shell script
- `makefile()` Create a Makefile
- **Quick Start** How to use `cmd_assign()`

Examples

```
if (interactive()) {
  cmd_assign_collect(
    files = c("data/x.csv", "data/y.csv"),
    use_log = TRUE
  )
}
```

extract_make	<i>Turn a 'cmd_assign' Call Into a Makefile Rule</i>
--------------	--

Description

Extract a call to `cmd_assign()` or `cmd_assign_collect()` 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 <code>cmd_assign()</code> or <code>cmd_assign_collect()</code> .
dir_make	The directory that contains the Makefile. The default is the current working directory.

Details

The rule may need editing before it is added to a Makefile.

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()` or `cmd_assign_collect()`, and will appear in the first line of the Makefile rule. When writing and testing the file, use `cmd_assign()` or `cmd_assign_collect()` 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.

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 identify arguments that refer to file names, `extract_make()` uses the following rules:

- 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
- `cmd_assign_collect()` Process command line arguments when the number of arguments is not known in advance
- [Quick Start](#) How to use `cmd_assign()` and `cmd_assign_collect()`
- [Modular Workflows for Data Analysis](#) Safe, flexible data analysis workflows
- [littler](#) Alternative to Rscript

Examples

```
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	<i>Turn a 'cmd_assign' Call Into a Shell Command</i>
---------------	--

Description

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

Usage

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

Arguments

<code>path_file</code>	Path to the R script containing the call to <code>cmd_assign()</code> or <code>cmd_assign_collect()</code> . The path starts at <code>dir_shell</code> .
<code>dir_shell</code>	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()` or `cmd_assign_collect()`, and will be the first argument passed to `Rscript` in the shell command. When writing and testing the script, use `cmd_assign()` or `cmd_assign_collect()` 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 identify arguments that refer to file names, `extract_shell()` uses the following rules:

- 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
- [cmd_assign_collect\(\)](#) Process command line arguments when the number of arguments is not known in advance
- [Quick Start](#) How to use `cmd_assign()` and `cmd_assign_collect()`
- [Modular Workflows for Data Analysis](#) Safe, flexible data analysis workflows

Examples

```
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

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

Details

The Makefile is a draft and needs editing before use. `makefile()` cannot infer every rule, and a file path that contains a space or a character special to Make must be corrected in the dependency line.

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(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
  list.files()

  ## 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() or cmd_assign_collect() .
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

shell_script() tries to put the commands in the order they need to run, by matching each script's output path with the input paths of the other scripts. It can get this order wrong, so the shell script may still need editing.

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(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(.results = 'out/results.rds', .out = 'out/more_results.rds')",
               "results <- readRDS(.results)",
               "more_results <- results + 2",
               "saveRDS(more_results, file = .out)",
               con = "src/more_results.R")

  ## the shell script runs 'results.R' first, because
  ## 'more_results.R' reads the file that 'results.R' writes

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

  ## shell script has been created
  list.files()

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

})
```

use_renv	<i>Activate an renv Project for the Current Session</i>
----------	---

Description

Make sure renv is activated, even if a script is run from Rscript or littler.

Usage

```
use_renv(project = NULL, quiet = TRUE)
```

Arguments

project	Optional path to a project root. If NULL (the default), search upward from the script's directory when known, otherwise from getwd(). The script directory is taken from Rscript's --file= argument when present, or from source references under littler.
quiet	If TRUE (the default), suppress messages.

Details

Package **renv** makes workflows reproducible by locking down the versions of R packages. Normal renv mechanisms can fail, however, if a script is run from Rscript or littler. Adding

```
command::use_renv()
```

to the top of R scripts fixes the problem.

For more details, see the article [Using command with renv](#).

Value

The project root path (invisibly) if renv was activated or was already active; otherwise NULL.

See Also

- `cmd_assign()` Process command line arguments
- `cmd_assign_collect()` Process command line arguments when the number of arguments is not known in advance
- [Using command with renv](#)

Examples

```
command::use_renv() ## if renv not used here, no effect

## see article for full example
```

Index

cmd_assign, 3
cmd_assign(), 2, 6–17
cmd_assign_collect, 6
cmd_assign_collect(), 2, 6, 8–13, 15, 17
cmd_assign_collect_quiet
 (cmd_assign_collect), 6
cmd_assign_quiet (cmd_assign), 3
command (command-package), 2
command-package, 2
commandArgs(), 3, 6

extract_make, 8
extract_make(), 2, 6, 8, 12, 14
extract_shell, 11
extract_shell(), 2, 6, 8, 10, 16

file.exists(), 10, 12

makefile, 13
makefile(), 2, 6, 8, 10, 15, 16

shell_script, 15
shell_script(), 2, 6, 8, 12–14

use_renv, 17
use_renv(), 2, 6
utils::Rscript(), 9, 11