Package 'mortar'

April 28, 2026

Type Package

Title Standardize Data Science Workflows

Version 0.4.0

Maintainer Anthony Martinez <ajmartinez@usgs.gov>

Description Helper functions to standardizes common workflows in the USGS Data Science Community of Practice to produce more robust, reproducible pipelines. It contains helper functions to standardize (1) the organization of project repositories and (2) the creation ofpipelines from the 'targets' R Package (Landau et al. (2026) <doi:10.5281/zenodo.18555866>), using the DS CoP best practices. We draw upon community developed best practices as well as certain USGS-specific requirements. See Shrycock et al. (2023) <doi:10.3133/tm7B2> for examples of these USGS requirements.

License CC0

BugReports https://github.com/DOI-USGS/mortar/issues

URL https://doi-usgs.github.io/mortar/, https://github.com/DOI-USGS/mortar, https://code.usgs.gov/water/computational-tools/mortar

Encoding UTF-8

Imports cli, gert, glue, purrr, R.utils, rlang, usethis

RoxygenNote 7.3.2

Suggests knitr, renv, rmarkdown, stringr, tarchetypes, targets, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Joseph Zemmels [aut] (ORCID: <https://orcid.org/0009-0008-1463-6313>), Anthony Martinez [aut, cre] (ORCID: <https://orcid.org/0000-0002-4295-0261>), Jesse Ross [ctb] (ORCID: <https://orcid.org/0000-0002-5422-8284>)

Repository CRAN

Date/Publication 2026-04-28 19:00:15 UTC

Contents

`file_edit` *Edit a text file programmatically*

Description

Insert or transform (`txt` argument) one or more lines in a file specified either by line number or a boolean function (`match` argument). Any file that can be read with the `readLines()` function should be compatible.

Usage

```
file_edit(file, txt, match, append = TRUE)
```

Arguments

`file`	chr, path to a file
`txt`	chr or chr fun, either a single character string to insert into the file or a function that transforms a character string. If supplying a function, the first argument of the function must accept a single string and it must return a single string. The function will be applied independently to each line in the file that matches the match argument. Supports anonymous functions. See examples.
`match`	num vector or lgl fun, either a numeric vector indicating line indices in the file or a function that returns a logical indicating lines to change. If supplying a function, the first argument of the function must accept a single string and it must return a logical vector. The function will be applied independently to each line in the file. Supports anonymous functions. See examples.
`append`	lgl, if TRUE the text will be added as a new line after the line(s) specified in the match argument. If FALSE, then the text will replace the lines specified in the match argument

Value

(invisibly) FALSE if no lines in the specified file match the match argument and TRUE otherwise

Examples

```
tmpfile <- tempfile()

writeLines(text =
             c("hello I am joe",
               "this is multiple lines of text",
               "12345",
               "",
               "^ line 4 is an empty line"),
           con = tmpfile)
cat(readLines(tmpfile),sep = "\n")

# The simplest use is to append a line in a file with new text. Note that txt
# must be a single character string.
file_edit(file = tmpfile,
          txt = "this line is appended after the second line",
          match = 2,
          append = TRUE)
cat(readLines(tmpfile),sep = "\n")

# We can instead replace a line using append = FALSE. Here, we replace the empty
# line in the file. After adding the line above, the empty line now occupies
# index 5.
file_edit(file = tmpfile,
          txt = "this line is no longer empty",
          match = 5,
          append = FALSE)
cat(readLines(tmpfile),sep = "\n")

# The txt argument can also be a function, anonymous or named. It just needs to
# accept a single string and return a single string. Here, we replace "is" with
# "was" in line 6.
file_edit(file = tmpfile,
          txt = ~ stringr::str_replace(.x, "is", "was"),
          match = 6,
          append = FALSE)
cat(readLines(tmpfile),sep = "\n")

# The match argument can also be a  function. It just needs to accept a single
# string and return a single boolean. You can use it with the txt argument
# for some interesting interactions. Here, we change every line containing
# the word "is" to uppercase
file_edit(file = tmpfile,
          txt = toupper,
          match = ~ stringr::str_detect(.x, " is "),
          append = FALSE)
cat(readLines(tmpfile),sep = "\n")

# strings containing carriage returns are preserved. Here, we use the readLines
# and length functions to ensure we append the string to the end of the file
multiline <-
"
```

```
Here is a string
that takes up multiple lines"

file_edit(file = tmpfile,
          txt = multiline,
          match = length(readLines(tmpfile)),
          append = TRUE)
cat(readLines(tmpfile),sep = "\n")
```

`get_usgs_gitlab_url` *Get the HTTPS URL for the git remote of a repo at code.usgs.gov*

Description

Get the HTTPS URL for the git remote of a repo at code.usgs.gov

Usage

```
get_usgs_gitlab_url(remote_name = "origin")
```

Arguments

`remote_name`	chr; name of the git remote. The default is "origin".

Value

a character sting of the HTTPS URL to the remote repo

Examples

```
## Not run:
# Will fail if working directory is not in a git repo
get_usgs_gitlab_url("origin")

## End(Not run)
```

`tar_init` *Initialize a targets project*

Description

Run this function in a new targets project directory. It will create R files and directories using the targets project structure often used by the USGS Data Science Community of Practice. For more information, internal USGS employees can look at section 12 of Targets 2 training course.

Usage

```
tar_init(
  home,
  phase_names,
  phase_nums = seq_along(phase_names),
  separate_phase_scripts = TRUE,
  phase_subdirs = c("src", "out"),
  use_leading_zeros = FALSE,
  overwrite = FALSE
)
```

Arguments

home	chr, root directory of targets project. To use the current working directory, use ".".
phase_names	chr vector, names of target phases like "fetch", "process", etc.
phase_nums	int vector, numbers to prepend to phase_names like "1_fetch", "2_process", etc. Defaults to seq_along(phase_names)
separate_phase_scripts	lgl, should a different R script be created for each phase like "1_fetch.R", "2_process.R", etc? If FALSE, then targets lists will be initialized in _targets.R file. Defaults to TRUE
phase_subdirs	chr vector, subdirectories within each phase. Defaults to "src", "out"
use_leading_zeros	lgl, should "0" be used in front of phase numbers in file and directory names?
overwrite	lgl, should the initialization overwrite files and folders that already exist? Defaults to FALSE

Value

NULL invisibly

Examples

```
# temporary directories in which targets project is initialized (you can skip
# this part if creating your own project)
tmp <- tempdir()
unlink(tmp, recursive = TRUE, force = TRUE)
dir.create(tmp)

# creates 1_fetch, 2_process, 3_summarize R scripts and directories
tar_init(home = tmp, phase_names = c("fetch", "process", "summarize"))

list.files(tmp, full.names = FALSE, recursive = TRUE, all.files = TRUE,
           pattern = "\\.(R|empty)$")

# clean out tmp folder
unlink(tmp, recursive = TRUE, force = TRUE)
```

```
dir.create(tmp)

# different structure starting with 0_config and including "in/" dir in each phase
tar_init(home = tmp,
         phase_names = c("config", "pull", "munge", "visualize"),
         phase_nums = 0:3,
         phase_subdirs = c("in", "src", "out"),
         separate_phase_scripts = FALSE)

list.files(tmp, full.names = FALSE, recursive = TRUE, all.files = TRUE,
           pattern = "\\.(R|empty)$")

# clean out tmp folder
unlink(tmp, recursive = TRUE, force = TRUE)
dir.create(tmp)
```

---

use-file-usgs *Add individual USGS project files to a directory*

Description

Creates common project files like .gitignore, README.md. LICENSE.md, etc.#'

Usage

```
use_gitignore_usgs(home, additions = NULL, open = rlang::is_interactive())

use_readme_usgs(home, open = rlang::is_interactive())

use_readme_rmd_usgs(home, open = rlang::is_interactive())

use_license_usgs(home, open = rlang::is_interactive())

use_code_json_usgs(home, open = rlang::is_interactive())

use_disclaimer_provisional_usgs(home, open = rlang::is_interactive())

use_disclaimer_approved_usgs(home, open = rlang::is_interactive())

use_code_of_conduct_usgs(home, open = rlang::is_interactive())

use_contributing_usgs(home, repo_url = NULL, open = rlang::is_interactive())

use_changelog_usgs(home, open = rlang::is_interactive())

use_gitlab_mr_template_usgs(home, open = FALSE)
```

Arguments

home	chr, root directory of targets project. To use the current working directory, use ".".
additions	chr vector, other files/directories to be added to .gitignore
open	lgl; whether to open the file for interactive editing
repo_url	chr, URL to Git repository. If NULL (default) a generic URL will be provided as a link to the repository in CONTRIBUTING.md. Otherwise, the issue page for repo_url will be used as the link.

Value

NULL invisibly

Examples

```
tmp <- tempdir()

use_gitignore_usgs(home = tmp,
                   additions = c("excluded_file.R",
                                 "excluded_dir",
                                 "*excluded_pattern*"))
# here are the contents of the .gitignore:
cat(readLines(file.path(tmp,".gitignore")), sep = "\n")

unlink(tmp, recursive = TRUE, force = TRUE)
```

---

use_project_usgs *Set up a USGS project directory*

Description

Wraps the other use_*_usgs functions to initialize multiple files in one directory.

Usage

```
use_project_usgs(
  home,
  gitignore_additions = NULL,
  readme_type = c("md", "rmd"),
  disclaimer_type = c("provisional", "approved"),
  repo_url = get_usgs_gitlab_url("origin"),
  use_mr_template = TRUE,
  open = rlang::is_interactive()
)
```

Arguments

home	chr, root directory of targets project. To use the current working directory, use ".".
gitignore_additions	chr vector, other files/directories to be added to .gitignore or NULL (default) for no additions.
readme_type	chr, either "md" to create README as a markdown file (README.md; default) or "rmd" to create it as an R markdown file (README.Rmd). Using an Rmd file will allow you to include R code and output in your README.md.
disclaimer_type	chr, either "provisional" (default) to include the provisional software disclaimer statement or "approved" for the approved disclaimer statement. It is important to only include the approved disclaimer statement if your software is an approved software release.
repo_url	chr, URL to Git repository. If NULL (default) a generic URL will be provided as a link to the repository in CONTRIBUTING.md. Otherwise, the issue page for repo_url will be used as the link.
use_mr_template	lgl, Should GitLab Merge Request template be included in project repository. Default is TRUE.
open	lgl; whether to open the files for interactive editing.

Value

NULL invisibly

Examples

```
tmp <- tempdir()
unlink(tmp, recursive = TRUE, force = TRUE)
dir.create(tmp)

use_project_usgs(
  home = tmp,
  repo_url = "https://code.usgs.gov/water/computational-tools/mortar"
)
list.files(tmp)

# start over
unlink(tmp, recursive = TRUE, force = TRUE)
dir.create(tmp)

# creates README.Rmd and DISCLAIMER_APPROVED instead
use_project_usgs(
  home = tmp,
  readme_type = "rmd",
  disclaimer_type = "approved",
  repo_url = "https://code.usgs.gov/water/computational-tools/mortar"
)
```

```
list.files(tmp)

# Clean up temp files
unlink(tmp, recursive = TRUE, force = TRUE)
```

Index