Package: projr (via r-universe)

June 4, 2026

Title Facilitate Reproducible and Archived Projects

Version 0.5.1

Maintainer Miguel Rodo <miguel.rodo@uct.ac.za>

Description Facilitate reproducible and archived projects.

License file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports renv, jsonlite, yaml, rprojroot, desc, fs, digest, cli, glue

Suggests remotes, roxygen2, usethis, gert, gh, credentials, gitcreds, codemeta, cffr, pkgbuild, rmarkdown, bookdown, knitr, quarto, testthat (>= 3.0.0), devtools, httr, covr, htmltools, DT, curl, rlang, withr, BiocManager, styler

Config/testthat/edition 3

URL https://satvilab.github.io/projr/

VignetteBuilder knitr

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-04 08:58:13 UTC

RemoteUrl https://github.com/SATVILab/projr

RemoteRef v0.5.1

RemoteSha 9dc37d66281448eb552e81f3022588aa39c126cf

Contents

.yml_get *Get active* projr *settings and do no check*

Description

A list of the active projr settings, without doing any error checking. Gets active projr settings, which merges settings and resolves conflicts between local (_projr-local.yml), profile (_projr-<projr>.yml or as - keys in _projr.yml) and default (_projr.yml) settings. Where there are conflicts, local settings has highest precedence (i.e. are always preferred) and default settings have lowest precedence (i.e. are never preferred).

Usage

```
.yml_get(profile)
```

Arguments

profile	character. If supplied, the specific profile file to read in. "default" loads _projr.yml, but another value loads _projr-<profile>.yml. If NULL, then the active profile is used. It not supplied, then treated as NULL.

Value

A named list.

See Also

.yml_get.yml_check

projr_build *Build project to output*

Description

.build_output‘ Builds project to output, which means recording the input and output data hashes, building the actual bookdown document and saving and archiving selected output.

.build_major, .build_minor and .build_patch are wrappers around .build_output with the version component bumped set automatically, e.g. projr_build_major() is equivalent projr_build(bump_component = "

Usage

```
projr_build(
  bump_component,
  msg = NULL,
  args_engine = list(),
  profile = NULL,
  archive_github = FALSE,
  archive_local = FALSE,
  always_archive = TRUE,
  clear_output = NULL
)

projr_build_major(
  msg = NULL,
  args_engine = list(),
  profile = NULL,
  archive_github = FALSE,
  archive_local = FALSE,
  always_archive = TRUE,
  clear_output = NULL
)

projr_build_minor(
  msg = NULL,
  args_engine = list(),
  profile = NULL,
  archive_github = FALSE,
  archive_local = FALSE,
  always_archive = TRUE,
  clear_output = NULL
)

projr_build_patch(
  msg = NULL,
  args_engine = list(),
  profile = NULL,
  archive_github = FALSE,
  archive_local = FALSE,
  always_archive = TRUE,
  clear_output = NULL
)
```

Arguments

bump_component	"major", "minor", "patch" or missing. Specifies version component to increment. If missing, then is set equal to the lowest version component in used version format. No default (i.e. is missing by default).
msg	character. Message to append to Git commit messages. Default is NULL, in which

	case the user is prompted for a message or, if the session is not interactive, it is left empty. Default is NULL. Note that the Git messages in this case would not be blank - they would simply consist of details as to the version being bumped to and the stage in the build process at which the commit was made.
args_engine	list. Arguments passed to the rendering engine (rmarkdown::render, quarto::render or bookdown::render_book).
profile	character. projr profile to use. Will set the environment variable .PROFILE‘ to this value at the start of the build,
archive_github	TRUE,FALSE orcharacter vector of directory labels. If TRUE, then all directories (raw-data, output, etc) are uploaded to a GitHub release named archive as versioned files (e.g. output-v0.1.2.zip). If FALSE, then no directories are uploaded. If a character vector, then only the directories specified are uploaded. Default is FALSE. Ignored if there is a release named archive already specified as a destination in the projr configuration file.
archive_local	TRUE, FALSE or character vector of directory labels. If TRUE, then all directories (raw-data, output, etc) are archived to a local directory. If FALSE, then no directories are archived locally. If a character vector, then only the directories specified are archived. Default is FALSE.
always_archive	logical. If TRUE, then the directories are uploaded regardless of whether the directory to be uploaded has exactly the same contents as the latest version of the directory on the GitHub release. Default is TRUE. Ignored if there is a release named archive already specified as a destination in the projr configuration file.
clear_output	character. When to clear output directories: "pre" (before build, default), "post" (after build), or "never". Can also be set via PROJR_CLEAR_OUTPUT environment variable.

`projr_build_check_packages`

Check if required packages for build are available

Description

Checks if all packages required for the current project build are installed. Returns structured information about missing packages and installation commands.

This is particularly useful in CI/CD environments where the installation commands need to be captured and executed programmatically.

Usage

```
projr_build_check_packages(profile = NULL)
```

Arguments

profile	character. projr profile to use. Default is NULL (use current profile).

Value

A list with the following elements:

available Logical. TRUE if all required packages are installed, FALSE otherwise.

missing Character vector of missing package names. Empty if all packages are available.

install_cmds Character vector of R commands to install missing packages. Empty if all packages are available.

message Character. Human-readable message about package status.

Examples

```
## Not run:
# Check packages for current project
pkg_status <- projr_build_check_packages()

if (!pkg_status$available) {
  cat("Missing packages:", paste(pkg_status$missing, collapse = ", "), "\n")
  cat("Install commands:\n")
  cat(paste(pkg_status$install_cmds, collapse = "\n"), "\n")

  # In CI/CD, you could execute:
  # for (cmd in pkg_status$install_cmds) {
  #   eval(parse(text = cmd))
  # }
}

## End(Not run)
```

`projr_build_dev` *Build dev project*

Description

Builds project to output, which means recording the input and output data hashes, building the actual bookdown document and saving and archiving selected output.

Usage

```
projr_build_dev(
  file = NULL,
  bump = FALSE,
  old_dev_remove = TRUE,
  args_engine = list(),
  profile = NULL,
  clear_output = "never"
)
```

Arguments

file	character vector. Paths to files to build. Paths may be relative to project root, or absolute. Default is NULL, in which case all files are built.
bump	logical. Whether to increment dev version for build. Default is FALSE.
old_dev_remove	logical. If TRUE, then previous development builds are deleted after a successful run.
args_engine	list. Arguments passed to the rendering engine (rmarkdown::render, quarto::render or bookdown::render_book).
profile	character. projr profile to use. Will set the environment variable .PROFILE‘ to this value at the start of the build,
clear_output	character. When to clear output directories: "pre" (before build), "post" (after build), or "never" (default for dev builds). Can also be set via PROJR_CLEAR_OUTPUT environment variable.

---

projr_cat_changelog *Cat useful information*

Description

Cat useful information. .cat_changelog‘:

Usage

```
projr_cat_changelog(n_row = 10)
```

Arguments

n_row	integer. Number of rows to cat.

---

projr_env_set *Set environment variables from files*

Description

Activate environment variables by reading default values from a set of files. If file is NULL, all existing files are used in order of decreasing priority: first _environment.local (machine-specific overrides), then any _environment-<profile> files (profile-specific), and finally _environment (global defaults).

The profiles activated are those set in the QUARTO_PROFILE and the PROJR_PROFILE environment variables, with QUARTO_PROFILE priorities. The QUARTO_PROFILE variable can specify multiple profiles, separated by commas. The PROJR_PROFILE variable can also specify multiple profiles, separated by either commas or semi-colons. In both cases, the earlier profile takes precedence, e.g. QUARTO_PROFILE=test,basic will activate the test profile first.

Usage

```
projr_env_set(file = NULL)
```

Arguments

file	character vector. Paths to files with environment variables to activate. If provided, only these files will be read; otherwise the default set (_environment.local, _environment-<profile>, _environment) is used.

Value

Invisibly returns TRUE if any files were successfully activated, or FALSE if none existed.

Examples

```
# Activate only the local overrides
## Not run:
projr_env_set("_environment.local")

## End(Not run)
# Activate all available defaults in the standard order
## Not run:
projr_env_set()

## End(Not run)
```

---

projr_ignore *Manually Ignore Files or Directories in* .gitignore *and* .Rbuildignore

---

Description

These functions allow manual addition of files and directories to the .gitignore and .Rbuildignore files, outside of the automatic management provided by the projr package.

- projr_ignore: General function to add both files and directories to both .gitignore and .Rbuildignore. If a path does not exist, it is treated as a file.
- projr_ignore_dir: Specifically adds directories to both .gitignore and .Rbuildignore.
- projr_ignore_file: Specifically adds files to both .gitignore and .Rbuildignore.
- projr_ignore_dir_git and projr_ignore_file_git: Add directories or files explicitly to .gitignore.
- projr_ignore_dir_rbuild and projr_ignore_file_rbuild: Add directories or files explicitly to .Rbuildignore.

Usage

```
projr_ignore(ignore, force_create = TRUE)

projr_ignore_dir(ignore, force_create = TRUE)

projr_ignore_file(ignore)

projr_ignore_file_git(ignore, force_create = TRUE)

projr_ignore_dir_git(ignore, force_create = TRUE)

projr_ignore_file_rbuild(ignore, force_create = TRUE)

projr_ignore_dir_rbuild(ignore, force_create = TRUE)
```

Arguments

ignore	A character vector of file or directory paths to be ignored. Paths must be valid non-empty strings.
force_create	logical. If FALSE, then the function will only add to the corresponding ignore file (.gitignore/.Rbuildignore) if it already exists, OR if it is warranted (i.e. if there is a Git repository or DESCRIPTION file, respectively). If TRUE, then the function will create the ignore file if it does not exist, even if there is no Git repository or DESCRIPTION file, and will add the specified paths to it. Default is TRUE.

Details

These functions provide fine-grained control for cases where users want to manually ignore specific paths permanently. They do not interact with the automated ignore management system of projr.

- Non-existent paths provided to projr_ignore are assumed to be files.
- For .gitignore, directories are automatically appended with /** if missing, ensuring proper Git ignore syntax.
- For .Rbuildignore, paths are converted to regular expressions using glob2rx for compatibility with R's build tools.

Value

Invisibly returns TRUE if the operation succeeds, or FALSE if the input contains invalid (empty) paths.

See Also

projr_ignore_auto for dynamically managed ignore entries, and projr_unignore_manual for forcing certain paths to not be ignored.

Examples

```
# Manually ignore files and directories
projr_ignore(c("output", "tempfile.log"))

# Specifically ignore directories
projr_ignore_dir("data")

# Specifically ignore files
projr_ignore_file("README.md")
```

`projr_ignore_auto` *Update* `.gitignore` *and* `.Rbuildignore` *with projr-managed ignores*

Description

The projr_ignore_auto()function updates the project's.gitignoreand.Rbuildignore‘ files to reflect directories and files managed by projr, as well as other directories and files that should clearly be ignored. They are kept up-to-date with the project's configuration, and are written within a demarcated section of the file.

Usage

```
projr_ignore_auto()
```

Value

Called primarily for its side effects (modifying `.gitignore` and/or `.Rbuildignore`). Returns `TRUE` invisibly.

See Also

.ignore_add.ignore_add_git.ignore_add_rbuild

Examples

```
## Not run:
projr_ignore_auto()

## End(Not run)
```

projr_init_github *Initialize a projr Project*

Description

This function performs a full initialization of a projr project. It sets up the project structure by creating directories, generating a README (in Markdown or R Markdown format), configuring a renv environment, writing a DESCRIPTION file, applying a license (if provided), setting up citation files, creating a projr configuration YAML file, establishing literate documentation, and configuring both Git and GitHub repositories.

Usage

```
projr_init_github(
  user = NULL,
  org = NULL,
  public = FALSE,
  create_repo = TRUE,
  git_commit = TRUE
)

projr_init(
  git = TRUE,
  git_commit = TRUE,
  github = TRUE,
  github_public = FALSE,
  github_user = NULL,
  github_org = NULL,
  dir = TRUE,
  readme = TRUE,
  readme_rmd = TRUE,
  desc = FALSE,
  license = NULL,
  projr_yml = FALSE,
  lit_doc = NULL
)

projr_init_all(
  github = TRUE,
  github_org = NULL,
  license = NULL,
  lit_doc = NULL
)

projr_init_renv(bioc = TRUE)

projr_init_cite()
```

```
projr_init_git(commit = TRUE)

projr_init_license(license, first_name, last_name)

projr_init_ignore()
```

Arguments

user	Character or \codeNULL (for \codeprojr_init_github). GitHub username to use when creating the remote; if \codeNULL the token owner is used. Defaults to \codeNULL.
org	Character or \codeNULL (for \codeprojr_init_github). GitHub organisation to use when creating the remote. Defaults to \codeNULL.
public	Logical (for `projr_init_github`). If `TRUE`, the GitHub repository will be public. Defaults to `FALSE`.
create_repo	Logical. If `TRUE` and the project does not have a local Git repository, then in interactive mode it offers to create a GitHub repository and in non-interactive mode creates one automatically. Defaults to `TRUE`.
git_commit	Logical. If `TRUE`, commits the initial changes to the Git repository. Defaults to `TRUE`.
git	Logical. If `TRUE`, initializes a Git repository. Defaults to `TRUE`.
github	Logical. If `TRUE`, attempts to create a GitHub repository for the project. Defaults to `TRUE`.
github_public	Logical. If `TRUE`, the GitHub repository will be public. Defaults to `FALSE`.
github_user, github_org	Character or `NULL`. The owner of the GitHub repo to create. If both are NULL, then creates the repository under the current user's account (as implied by the GitHub token). If both are specified, then organisation is preferred. Default is `NULL`.
dir	Logical. If `TRUE`, initializes the projr-specified directories (e.g., raw, cache, output). Defaults to `TRUE`.
readme	Logical. If `TRUE`, creates a README file. Defaults to `TRUE`.
readme_rmd	Logical. If `TRUE`, generates a README in R Markdown format (`README.Rmd`); otherwise, a Markdown file (`README.md`) is created. Defaults to `TRUE`.
desc	Logical. If `TRUE`, creates a DESCRIPTION file for the project. Defaults to `FALSE`.
license	Character or `NULL`. Specifies the license to apply (e.g., `"ccby"`, `"apache"`, `"cc0"`, `"proprietary"`). Defaults to `NULL`.
projr_yml	Logical. If `TRUE`, creates a `projr.yml` configuration file. Defaults to `FALSE`.
lit_doc	Character or `NULL`. Specifies the type of literate documentation to create. Supported values are `"bookdown"`, `"project"`, `"quarto"`, and `"rmd"`. Defaults to `NULL`.

bioc	Logical (for \codeprojr_init_renv). If \codeTRUE, includes Bioconductor packages in the renv setup. Defaults to \codeTRUE.
commit	Logical (for \codeprojr_init_git). If \codeTRUE, commits the initial changes to the Git repository. Defaults to \codeTRUE.
first_name	Character (for projr_init_license). First name for proprietary license. Required when license = "proprietary".
last_name	Character (for projr_init_license). Last name for proprietary license. Required when license = "proprietary".
username	Character or NULL (for projr_init_github). The GitHub username or organization under which to create the repository. Defaults to NULL.

Details

The projr_init function is a wrapper that calls several helper functions to perform the following tasks:

- Prevent working directory errors by ensuring the **usethis** project is set.
- Create project directories.
- Generate a README file (in Markdown or R Markdown format).
- Initialize a renv environment, optionally with Bioconductor support.
- Write a DESCRIPTION file for project metadata.
- Apply a specified license.
- Configure citation files (if a DESCRIPTION file exists).
- Create a projr configuration YAML file.
- Set up literate documentation in the chosen format.
- Initialize Git (and optionally commit initial changes).
- Create a GitHub repository if requested.

Value

Invisibly returns TRUE if initialization is successful, or FALSE if a particular step is skipped.

---

projr_init_prompt *Initialise project*

Description

Initialise project

Usage

```
projr_init_prompt(
  yml_path_from = NULL,
  renv_force = FALSE,
  renv_bioconductor = TRUE,
  public = FALSE
)
```

Arguments

yml_path_from	character. Path to YAML file to use as _projr.yml. If not supplied, then default _projr.yml file is used.
renv_force	Logical. Passed to renv::init(). If FALSE, then renv::init() will not run if it detects that the working directory already is registered with renv. Default is FALSE.
renv_bioconductor	Logical. Whether renv should look for packages on Bioconductor. Default is TRUE.
public	logical. Whether the GitHub repo created (if any) is public or not. Default is FALSE.

See Also

.init_renviron

---

projr_init_renviron *Set environment variables for projr_init*

Description

Set environment variables for projr_init. When set, projr_init will use these variables to provide options to populate the metadata. This function creates the .Renviron file for the user if it does not exist. It then adds the variables to the .Renviron file without any values set.

Usage

```
projr_init_renviron()
```

`projr_instr_auth_github` *Two-minutes or less authorisation instructions*

Description

Print easy-to-follow, step-by-step instructions for authorisation to GitHub.

Usage

```
projr_instr_auth_github()
```

`projr_license_create_manual` *Manually create LICENSE files without YAML configuration*

Description

Creates LICENSE files in directories without adding license configuration to `_projr.yml`. This allows manual editing of licenses without them being overwritten during builds. If a license configuration exists in the YAML for a directory, it will take precedence and overwrite the manual license.

Usage

```
projr_license_create_manual(
  type,
  labels = NULL,
  authors = NULL,
  year = NULL,
  profile = "default"
)
```

Arguments

`type`	character. License type. Supported types: "CC-BY", "CC0", "Apache-2.0", "MIT", "Proprietary".
`labels`	character vector. Directory labels to create licenses for. If NULL, creates for all raw data directories (raw-data, cache).
`authors`	character vector. Authors or copyright holders. If NULL, attempts to get from DESCRIPTION file or uses "Project Authors" as default.
`year`	integer. Copyright year. If NULL, uses current year.
`profile`	character. Profile to use. Default is "default".

Value

Invisible character vector of labels where licenses were created.

Examples

```
## Not run:
# Create MIT license in raw-data directory
projr_license_create_manual("MIT", "raw-data")

# Create CC-BY license in all raw data directories
projr_license_create_manual("CC-BY")

# Create with custom authors
projr_license_create_manual(
  "Apache-2.0",
  "raw-data",
  authors = c("Jane Doe", "John Smith"),
  year = 2024
)

## End(Not run)
```

---

projr_log_clear *Clear build logs*

Description

Delete build logs based on specified criteria.

Usage

```
projr_log_clear(
  build_type = "all",
  history = TRUE,
  output = TRUE,
  before_date = NULL,
  before_version = NULL
)
```

Arguments

build_type	Character. Either "output", "dev", or "all". Default is "all".
history	Logical. Clear history files. Default is TRUE.
output	Logical. Clear output log files. Default is TRUE.
before_date	Character. Clear logs before this date (YYYY-MM-DD). Default is NULL (no date filter).
before_version	Character. Clear logs before this version. Default is NULL (no version filter).

projr_log_view *View build log (last n lines)*

Description

Display the last N lines of a detailed build log file.

Usage

```
projr_log_view(
  log_file = NULL,
  build_type = "auto",
  n_lines = 10,
  show_header = TRUE
)
```

Arguments

log_file	Character. Path to a log file. If NULL, the most recent log file across both output and dev builds will be used. To view a specific build type, use the build_type parameter.
build_type	Character. Either "output", "dev", or "auto" (default). When "auto", selects the most recent log across both types. When "output" or "dev", selects the most recent log of that specific type.
n_lines	Integer. Number of lines to show from the end of the file. Default is 10. Set to NULL or NA to show the entire file.
show_header	Logical. Whether to print a short header including the logfile path and last modification time. Default is TRUE.

projr_manifest_changes *Query Files Changed Between Versions*

Description

Query which files changed between two project versions based on the manifest. Returns files that were added, modified, or removed between the versions.

Query which files changed across a range of versions. For each file, shows the version where it last changed.

Query when files in each directory last changed. Shows the most recent version where any file in the directory was modified.

Usage

```
projr_manifest_changes(version_from = NULL, version_to = NULL, label = NULL)

projr_manifest_range(version_start = NULL, version_end = NULL, label = NULL)

projr_manifest_last_change(version = NULL)
```

Arguments

version_from	character. Starting version (e.g., "0.0.1" or "v0.0.1"). If NULL, uses the earliest version in the manifest.
version_to	character. Ending version (e.g., "0.0.2" or "v0.0.2"). If NULL, uses the current project version.
label	character. Optional directory label to filter by.
version_start	character. Starting version (inclusive). If NULL, uses earliest version.
version_end	character. Ending version (inclusive). If NULL, uses current version.
version	character. Version to query. If NULL, uses current project version.

Value

A data.frame with columns:

label Directory label

fn File path relative to directory

change_type Type of change: "added", "modified", or "removed"

hash_from File hash in version_from (NA for added files)

hash_to File hash in version_to (NA for removed files)

Returns a 0-row data.frame if no changes found.

A data.frame with columns:

label Directory label

fn File path relative to directory

version_first First version where file appeared

version_last_change Last version where file was modified

hash Current file hash

A data.frame with columns:

label Directory label

version_last_change Most recent version with changes

n_files Number of files in directory at this version

Examples

```
## Not run:
# Query changes between v0.0.1 and v0.0.2
projr_manifest_changes("0.0.1", "0.0.2")

# Query changes in output directory only
projr_manifest_changes("0.0.1", "0.0.2", label = "output")

# Query changes from earliest version to current
projr_manifest_changes()

## End(Not run)

## Not run:
# Query all changes from v0.0.1 to current
projr_manifest_range("0.0.1")

# Query changes in a specific range
projr_manifest_range("0.0.1", "0.0.5")

## End(Not run)

## Not run:
# Query last changes for current version
projr_manifest_last_change()

# Query last changes as of v0.0.5
projr_manifest_last_change("0.0.5")

## End(Not run)
```

`projr_manifest_file_last_change`

Query When a Specific File Last Changed

Description

Query when a specific file last changed in the manifest. Returns the most recent version where the file's hash was different from the previous version, or when it first appeared.

Check if a specific file changed between two versions. Returns TRUE if the file's hash is different between the versions, was added, or was removed.

Get all versions where a specific file changed or appeared. Returns a chronological list of all versions in the manifest where the file's hash is different from the previous version.

Get the version when a specific file first appeared in the manifest.

Usage

```
projr_manifest_file_last_change(fn, label = NULL, version_end = NULL)

projr_manifest_file_changed(
  fn,
  label = NULL,
  version_from = NULL,
  version_to = NULL
)

projr_manifest_file_history(fn, label = NULL)

projr_manifest_file_first(fn, label = NULL)
```

Arguments

fn	character. File path relative to the directory (e.g., "data.csv", "subdir/file.txt").
label	character. Directory label (e.g., "output", "raw-data"). If NULL, searches all directories for the file.
version_end	character. End version to search up to. If NULL, uses current project version.
version_from	character. Starting version (e.g., "0.0.1" or "v0.0.1"). If NULL, uses the earliest version in the manifest.
version_to	character. Ending version (e.g., "0.0.2" or "v0.0.2"). If NULL, uses the current project version.

Value

A data.frame with columns:

label Directory label

fn File path

version_last_change Version when file last changed

hash Current file hash at that version

Returns a 0-row data.frame if file not found.

A data.frame with columns:

label Directory label

fn File path

changed Logical - TRUE if file changed

change_type Type of change: "added", "modified", "removed", or "unchanged"

hash_from File hash in version_from (NA for added files)

hash_to File hash in version_to (NA for removed files)

Returns a 0-row data.frame if file not found in either version.

A data.frame with columns:

label Directory label

fn File path

version Version where file changed or appeared

hash File hash at this version

change_type Type of change: "first_appearance", "modified", or "current"

Returns a 0-row data.frame if file not found.

A data.frame with columns:

label Directory label

fn File path

version_first Version when file first appeared

hash File hash at first appearance

Returns a 0-row data.frame if file not found.

Examples

```
## Not run:
# Query when a specific file last changed
projr_manifest_file_last_change("data.csv", label = "output")

# Search all directories for a file
projr_manifest_file_last_change("report.pdf")

## End(Not run)

## Not run:
# Check if a file changed between versions
projr_manifest_file_changed("data.csv", "output", "0.0.1", "0.0.2")

# Check against current version
projr_manifest_file_changed("data.csv", "output", "0.0.1")

## End(Not run)

## Not run:
# Get full history for a file
projr_manifest_file_history("data.csv", label = "output")

# Search all directories
projr_manifest_file_history("config.yml")

## End(Not run)

## Not run:
# Get when a file first appeared
projr_manifest_file_first("data.csv", label = "output")

# Search all directories
```

```
projr_manifest_file_first("README.md")

## End(Not run)
```

`projr_par_get` *Get project parameters*

Description

Get project parameters from `param` key in `projr` configuration.

Usage

```
projr_par_get(..., profile = NULL)

projr_param_get(..., profile = NULL)
```

Arguments

`...`	character. Sequential names to specify path in list. For example, .param_get("a", "b")`returns the value of`projr$param$a$b‘.
`profile`	character. If `NULL`, then the active profile is used. Default is `NULL`.

`projr_path_get` *Return path*

Description

Returns path to `projr` profile-specific directory. Differs from `projr_dir_get` in that it does not assume that the path is to a directory.

Will create the parent directory of the specified path if it does not exist, and ignore it if requested by `_projr.yml`.

Usage

```
projr_path_get(
  label,
  ...,
  create = TRUE,
  relative = FALSE,
  absolute = FALSE,
  safe = TRUE
)
```

Arguments

label	character. One of "raw_data", "cache","output", "archive" and "docs". Class of directory to return. The "docs" option returns the path to the output directory from bookdown::render_book (as specified in "_bookdown.yml"), whereas the others returns paths as specified in "_projr.yml".
...	Specifies sub-path of directory returned.
create	logical. If TRUE, then the parent directory is created if it does not exist and it is ignored (or not) from .gitignore and .Rbuildignore as specified in _projr.yml. Default is TRUE.
relative	logical. If TRUE, then forces that the returned path is relative to the project root. Default is FALSE.
absolute	logical. If TRUE, then forces the returned path to be absolute. Default is FALSE.
safe	logical. If TRUE, then the output directory is set to be "<path_to_cache>.output" instead of <path_to_output> (as specified in _projr.yml). The only time that this should be set to TRUE should be when .build_output is being run, as otherwise "development" c Default is TRUE. Do not change this unless you know what you are doing.

Details

DETAILS

Value

Character. Path to directory requested.

Examples

```
## Not run:
if (interactive()) {
  # EXAMPLE1
}

## End(Not run)
```

---

projr_path_get_cache_build_dir

Get projr *build cache directory*

Description

Get the cache directory for projr builds. It is a sub-directory of the cache directory. For development builds (.build_dev), this is the final directory for outputanddocs items. For output builds (.build_outp this is the staging directory. After the documents are rendered, they are copied to their final directories.

.path_get_cache_build assumes the path is to a file, whereas .path_get_cache_build_dir assumes the path is to a directory. This distinctiion is only relevant when create = TRUE, as it determines what directory is attempted to be created.

Usage

```
projr_path_get_cache_build_dir(..., create = FALSE, profile)

projr_path_get_cache_build(..., create = FALSE, profile)
```

Arguments

...	comma-separated strings specified initially the label (e.g. "docs" or "output") as well as, optionally, sub-directories (e.g. "img", "). For example, .path_get_cache_build("docs", "img")returns the path to theimgdirectory in thedocs‘ sub-directory of the build cache directory.
create	logical. If TRUE, then the directory is created if it does not exist.
profile	character. The name of the projr profile to use. Default is NULL, which uses the current projr profile.

Value

character. Path to the cache (sub-)directory for projr builds.

See Also

.path_get.path_get_dir

---

projr_path_get_dir *Return path to profile-specific directory*

Description

Returns path to projr profile-specific directory. Also creates the directory if it does not exist, and ignores it if requested by _projr.yml.

Usage

```
projr_path_get_dir(
  label,
  ...,
  create = TRUE,
  relative = FALSE,
  absolute = FALSE,
  safe = TRUE
)
```

Arguments

label	character. One of "raw", "cache","output", "archive" and "docs". Class of directory to return. The "docs" option returns the path to the output directory from bookdown::render_book (as specified in "_bookdown.yml"), whereas the others returns paths as specified in "_projr.yml".
...	Specifies sub-directory of directory returned. Passed to file.path.
create	logical. If TRUE, then the directory is created if it does not exist and it is ignored (or not) from .gitignore and .Rbuildignore as specified in _projr.yml. Default is TRUE.
relative	logical. If TRUE, then forces that the returned path is relative to the project root. Default is FALSE.
absolute	logical. If TRUE, then forces the returned path to be absolute. Default is FALSE.
safe	logical. If TRUE, then the output directory is set to be "<path_to_cache>.output" instead of <path_to_output> (as specified in _projr.yml). The only time that this should be set to TRUE should be when .build_output is being run, as otherwise "development" c Default is TRUE. Do not change this unless you know what you are doing.

Value

Character. Path to directory requested.

---

projr_profile_create *Add projr profile file*

Description

Creates a new projr profile that can override settings in _projr.yml. If the associated file does not exists, it creates a blank file. The file is ignored from the R build process and, if it is the local profile, from Git as well.

Usage

```
projr_profile_create(profile)
```

Arguments

profile	character. Name of the profile. If not supplied, then the profile is named default (and the file _projr.yml is created).

Value

Invisibly returns TRUE if the file was created, and FALSE if the file already exists.

See Also

.profile_create_local.profile_get

projr_profile_create_local *Create a local* projr *profile*

Description

Create a projr profile with highest precedence (i.e. its settings overwrite any others) that is ignored by Git.

Usage

```
projr_profile_create_local()
```

Details

Note that if any setting in _projr-local.yml is empty, then a lower-precendence file's setting (i.e. from _projr-<profile>.yml or _projr.yml) is used. Empty settings are by default indicated by ~.

See Also

.profile_create_local.yml_get

projr_profile_delete *Delete a projr profile from _projr.yml*

Description

Deletes a projr profile from _projr.yml and/or its corresponding _projr-<profile>.yml file.

Usage

```
projr_profile_delete(profile)
```

Arguments

profile	character. projr profile to delete.

Value

invisible(TRUE).

projr_profile_delete_local *Delete local* projr *settings file.*

Description

Deletes _projr-local.yml file.

Usage

```
projr_profile_delete_local()
```

projr_profile_get *Get active projr profile*

Description

Get active projr profile(s).

Usage

```
projr_profile_get()
```

Details

Note that local and default are not returned, but are always active (if they exist). local corresponds to _projr-local.yml and default to _projr.yml.

Value

Character vector of length equal to number of active profiles.

`projr_renv_restore` *Restore or Update renv Lockfile Packages*

Description

Functions to manage the restoration and updating of packages specified in the `renv` lockfile.

- .renv_restore()‘: Restores packages from the lockfile, attempting to install the lockfile versions.
- .renv_update()‘: Updates packages to their latest available versions, ignoring the lockfile versions.
- .renv_restore_and_update()‘: First restores packages from the lockfile, then updates them to the latest versions.

Usage

```
projr_renv_restore(
  github = TRUE,
  non_github = TRUE,
  biocmanager_install = FALSE
)

projr_renv_update(
  github = TRUE,
  non_github = TRUE,
  biocmanager_install = FALSE
)

projr_renv_restore_and_update(
  github = TRUE,
  non_github = TRUE,
  biocmanager_install = FALSE
)
```

Arguments

`github`	Logical. Whether to process GitHub packages. Default is `TRUE`.
`non_github`	Logical. Whether to process non-GitHub packages (CRAN and Bioconductor). Default is `TRUE`.
`biocmanager_install`	Logical. If `TRUE`, Bioconductor packages will be installed using `BiocManager::install`; otherwise, `renv::install("bioc::<package_name>")` will be used. Default is `FALSE`.

Details

Control whether to process GitHub packages, non-GitHub packages (CRAN and Bioconductor), or both using the `github` and `non_github` arguments.

Value

Invisibly returns TRUE upon successful completion.

Examples

```
## Not run:
# Restore all packages
projr_renv_restore()

# Update all packages
projr_renv_update()

# Restore and then update all packages
projr_renv_restore_and_update()

# Only restore non-GitHub packages
projr_renv_restore(github = FALSE)

# Only update GitHub packages
projr_renv_update(non_github = FALSE)

## End(Not run)
```

---

projr_renv_test *Test renv restore*

Description

Tests `renv::restore()` without using the cache in a clean, temporary environment. Automatically creates a temporary project directory, initializes `renv`, copies required files, disables the cache via `.Rprofile`, and then performs `renv::restore()`. Afterwards, it deletes the first library path where `renv` restored packages.

Note: To ensure isolation, the test runs in a directory that is completely separate from the parent project and executes `Rscript` with the `--vanilla` option. The `--vanilla` flag seems essential to prevent the project `renv` settings from being affected by testing.

Usage

```
projr_renv_test(files_to_copy = NULL, delete_lib = TRUE)
```

Arguments

`files_to_copy`	character vector. Paths to files to copy into the temporary directory before restoring. Note that `renv.lock` is always copied.
`delete_lib`	Logical. If TRUE, the restored library path is deleted after the test. Default is TRUE.

Value

TRUE if renv::restore() succeeds, FALSE otherwise.

---

projr_restore_repo *Restore project artefact directories*

Description

Use projr_content_update() to restore all artefacts needed for the current project. If the project isn't available locally yet, projr_restore_repo() will clone it and then restore its artefacts.

Usage

```
projr_restore_repo(
  repo,
  path = NULL,
  label = NULL,
  pos = NULL,
  type = NULL,
  title = NULL
)

projr_restore_repo_wd(
  repo,
  label = NULL,
  pos = NULL,
  type = NULL,
  title = NULL
)

projr_content_update(
  label = NULL,
  pos = NULL,
  type = NULL,
  title = NULL,
  clear = FALSE
)
```

Arguments

repo	character. GitHub repository ("owner/repo" or "repo"). (Only for repository restoration functions.) Must be a single non-empty character string.
path	character or NULL. Local path for cloning the repository. Default is NULL, creating a subdirectory named after the repo. "." restores directly into the current directory. Must be NULL or a single non-empty character string.

label	character vector or NULL. Specifies labels of artefacts to restore. Default is NULL, restoring all raw artefacts (e.g. raw-data). Must be NULL or a non-empty character vector with valid directory labels.
pos	character vector or NULL. Specifies preferred source: "source" (directories) or "dest" (build destinations). Default is NULL, checking both in order. Must be NULL or one/both of "source" and "dest".
type	character or NULL. Remote type: "local" or "github". Default is NULL, automatically choosing the first available remote. Must be NULL or one of the valid remote types.
title	character or NULL. Remote title as specified in _projr.yml. Default is NULL, using the first available title for the selected type. Must be NULL or a single non-empty character string.
clear	logical. If TRUE, clears existing local artefact directories before restoration. Default is FALSE.

Details

These functions restore artefact directories from remote sources:

- projr_content_update() restores artefacts in an existing local project without any cloning required. Requires a manifest.csv file in the project root.
- projr_restore_repo() clones a GitHub repository into a subdirectory (or specified path), then restores artefacts from that repository's remote sources.
- projr_restore_repo_wd() clones directly into the current working directory, then restores artefacts.

Input Validation: All parameters are validated before execution:

- label: Must be NULL or a non-empty character vector of valid directory labels
- pos: Must be NULL or contain only "source" and/or "dest"
- type: Must be NULL or one of "local" or "github"
- title: Must be NULL or a single character string
- repo: Must be a single non-empty character string
- path: Must be NULL or a single non-empty character string

Error Handling: The functions handle errors gracefully:

- Missing manifest.csv triggers an informative error
- Invalid labels or missing remote sources result in warning messages and skipped restoration
- Git clone failures are caught and reported
- Errors during restoration are caught per label, allowing partial success

Value

Invisibly returns TRUE if all restorations are successful, FALSE otherwise. For projr_content_update(), returns FALSE if no labels are found to restore or if any restoration fails. For projr_restore_repo(), returns FALSE if cloning or restoration fails.

Examples

```
## Not run:
# Restore all raw artefacts in existing local project
projr_content_update()

# Restore specific labels
projr_content_update(label = c("raw-data", "cache"))

# Restore from specific source type
projr_content_update(type = "local", title = "archive")

# Clone repository into subdirectory and restore artefacts
projr_restore_repo("owner/repo")

# Clone to specific path
projr_restore_repo("owner/repo", path = "my-project")

# Clone repository into current directory and restore artefacts
projr_restore_repo_wd("owner/repo")

## End(Not run)
```

---

projr_unignore_manual *Manually Unignore Files or Directories in* .gitignore *and* .Rbuildignore

---

Description

These functions allow manual addition of files and directories to the .gitignore and .Rbuildignore files **after** the projr-managed block, thereby forcing them to be *not* ignored.

- .unignore_manual: General function to unignore both files and directories in both .gitignoreand.Rbuildignore‘. If a path does not exist, it is treated as a file.
- .unignore_manual_dir: Specifically unignores directories in both .gitignoreand.Rbuildignore‘.
- .unignore_manual_file: Specifically unignores files in both .gitignoreand.Rbuildignore‘.
- .unignore_manual_dir_git and .unignore_manual_file_git: Add directories or files explicitly (with a ! prefix) to .gitignore.
- .unignore_manual_dir_rbuild and .unignore_manual_file_rbuild: Add directories or files explicitly (with a ! prefix) to .Rbuildignore.

Usage

```
projr_unignore_manual(unignore)

projr_unignore_manual_dir(unignore)
```

```
projr_unignore_manual_file(unignore)

projr_unignore_manual_file_git(unignore)

projr_unignore_manual_dir_git(unignore)

projr_unignore_manual_file_rbuild(unignore)

projr_unignore_manual_dir_rbuild(unignore)
```

Arguments

unignore	A character vector of file or directory paths to be unignored. Paths must be valid non-empty strings.

Details

These functions provide fine-grained control for cases where users want to *undo* any ignoring behavior for specific paths permanently. They do not interact with the automated ignore management system of `projr`.

- Non-existent paths provided to .unignore_manual‘ are assumed to be files.
- For `.gitignore`, unignored directories are automatically appended with `/**` if missing, then prepended with `!`, ensuring proper Git *unignore* syntax.
- For `.Rbuildignore`, paths are converted to regular expressions using `glob2rx()`, and then prepended with `!` for compatibility with R's build tools.

Value

Invisibly returns `TRUE` if the operation succeeds, or `FALSE` if the input contains invalid (empty) paths.

See Also

.ignore_manual `for manually ignoring paths, and` `.ignore_auto` for dynamically managed ignore entries.

Examples

```
# Manually unignore files and directories
projr_unignore_manual(c("output", "tempfile.log"))

# Specifically unignore directories
projr_unignore_manual_dir("data")

# Specifically unignore files
projr_unignore_manual_file("README.md")
```

projr_use_data projr *drop-in replacement for usethis::use_data*

Description

usethis::use_data always saves data to data/, which conflicts with the temporary directories used by .build_dev‘ and makes it difficult to restore after failed output builds.

.use_datais a drop-in replacement forusethis::use_data, which saves data to the temporary directory when safe = TRUE‘. This makes it easier to restore the project after a failed output build.

The only other difference is that .use_datainvisibly returns the path to the saved data file, whereasusethis::use_d

Usage

```
projr_use_data(
  ...,
  internal = FALSE,
  overwrite = FALSE,
  compress = "bzip2",
  version = 2,
  ascii = FALSE,
  safe = TRUE
)
```

Arguments

...	Unquoted names of existing objects to save.
internal	If FALSE, saves each object in its own .rda file in the data/ directory. These data files bypass the usual export mechanism and are available whenever the package is loaded (or via data() if LazyData is not true). If TRUE, stores all objects in a single R/sysdata.rda file. Objects in this file follow the usual export rules. Note that this means they will be exported if you are using the common exportPattern() rule which exports all objects except for those that start with ..
overwrite	By default, .use_data()will not overwrite existing files. If you really want to do so, set th
compress	Choose the type of compression used by save(). Should be one of "gzip", "bzip2", or "xz".
version	The serialization format version to use. The default, 2, was the default format from R 1.4.0 to 3.5.3. Version 3 became the default from R 3.6.0 and can only be read by R versions 3.5.0 and higher.
ascii	if TRUE, an ASCII representation of the data is written. The default value of ascii is FALSE which leads to a binary file being written. If NA and version >= 2, a different ASCII representation is used which writes double/complex numbers as binary fractions.
safe	logical. Whether to save data to a temporary directory (in <cache>/"projr"/v<version>/data/) or "data/". Default is the temporary directory (TRUE).

Details

Taken directly from the documentation for the original `usethis` function, and adjusted slightly.

See Also

The data chapter of R Packages.

Examples

```
## Not run:
x <- 1:10
y <- 1:100

projr_use_data(x, y) # For external use
projr_use_data(x, y, internal = TRUE) # For internal use

## End(Not run)
```

`projr_version_get` *Returns project version*

Description

Returns project version

Usage

```
projr_version_get()
```

Value

Character.

`projr_version_set` *Set Project Version*

Description

Sets the project version manually in the `DESCRIPTION` file and optionally in a `VERSION` file. This is useful in cases where you need to increment the version manually, for example, if a collaborator has pushed changes and you want to manually set your version before merging.

Usage

```
projr_version_set(version, only_if_exists = TRUE)
```

Arguments

version	A character string specifying the version to set. It may include a development component (e.g., "1.2.3-dev") or just the stable version (e.g., "1.2.3").
only_if_exists	A logical flag indicating whether to update the VERSION file only if it already exists (TRUE) or to create it if it doesn't exist (FALSE). Defaults to TRUE.

Value

Invisibly returns TRUE if successful.

---

projr_yml_check *Check active* projr *settings.*

Description

Checks correctness of active projr settings.

Usage

```
projr_yml_check(profile = NULL)
```

Arguments

profile	character(). Profile whose file needs to be checked. If not supplied, then the merged profile is used. Default is NULL.

Value

Returns TRUE if all checks pass. Otherwise throws an error.

---

projr_yml_cite_set *Set citation options*

Description

.yml_cite_setsets the citation options in_projr.yml‘.

The options are:

- codemeta: whether to generate a codemeta.json file.
- cff: whether to generate a CITATION.cff file. Indexable by GitHub and Zenodo.
- inst-citation: whether to generate a CITATION file in the inst/ directory. If the project is installed as an R package, then citation data can be generated by citation(package = <project_name>).

The default is to leave all the settings unchanged.

If these settings are not setting in _projr.yml, then the default is to not generate any citation files.

.yml_cite_set_default sets all citation options to default (FALSE‘).

Usage

```
projr_yml_cite_set(
  all = NULL,
  codemeta = NULL,
  cff = NULL,
  inst_citation = NULL,
  simplify_identical = TRUE,
  simplify_default = TRUE,
  profile = "default"
)

projr_yml_cite_set_default(
  profile = "default",
  simplify_identical = TRUE,
  simplify_default = TRUE
)
```

Arguments

`all`	logical. If not `NULL`, then all citation options are set to this value. Default is `NULL`.
`codemeta`	logical. Whether to generate a codemeta.json file. If `NULL`, then setting is not changed.
`cff`	logical. Whether to generate a CITATION.cff file. If `NULL`, then setting is not changed.
`inst_citation`	logical. Whether to generate a CITATION file in the inst/ directory. If `NULL`, then setting is not changed.
`simplify_identical`	logical. If `TRUE`, then if all the settings are the same then only `cite: TRUE` or `cite: FALSE` is written to `_projr.yml`. Default is `TRUE`.
`simplify_default`	logical. If `TRUE`, then if all the settings are the same and equl to the default (`FALSE`), then the settings are not recorded in the `projr` configuration file (as the default will be equal to it).
`profile`	character. The profile to write to. Default is "default", in which case it writes to `_projr.yml`.

Examples

```
## Not run:
# set all to TRUE
projr_yml_cite_set(all = TRUE)

# set all to FALSE
projr_yml_cite_set(all = FALSE)

# set only cff to FALSE
```

```
projr_yml_cite_set(cff = FALSE)

# revert to defaults
projr_yml_cite_set()

## End(Not run)
```

`projr_yml_dest_add_github` *Add a GitHub release as a destination*

Description

Add a GitHub release as a destination to a _projr.yml file.

Usage

```
projr_yml_dest_add_github(
  title,
  content,
  structure = NULL,
  overwrite = TRUE,
  send_cue = NULL,
  send_strategy = NULL,
  send_inspect = NULL,
  profile = "default"
)
```

Arguments

`title`	character. Title of the GitHub release. Can use title as `@version`, in which case the release will be entitled the version of the project at the time. If not supplied, then will automatically be generated from `content`.
`content`	character vector. Labels of directories to include in the upload. Options are the labels of directories in the active `projr` configuration, as well as "docs", "data" and "code".
`structure`	"latest" or "archive". Structure of the remote. Default is NULL.
`overwrite`	logical. Whether to rewrite an existing entry. Default is TRUE.
`send_cue`	"always", "if-change" or "never". When to send to the remote. Default is NULL.
`send_strategy`	"upload-all", "upload-missing", "sync-purge" or "sync-diff". How to synchronise to the remote. Default is NULL.
`send_inspect`	"manifest" , "file" or "none". What to look at to find files on the remote. Default is NULL.
`profile`	character. Profile to write the settings to. Default is "default".

`projr_yml_dest_add_local` *Add a local directory as a destination*

Description

Add a local directory as a destination to a _projr.yml file.

Usage

```
projr_yml_dest_add_local(
  title,
  content,
  path,
  structure = NULL,
  overwrite = TRUE,
  send_cue = NULL,
  send_strategy = NULL,
  send_inspect = NULL,
  profile = "default"
)
```

Arguments

`title`	character. The name of the directory. Has no effect besides helping structure `_projr.yml`. If not supplied, will be made equal to `content`.
`content`	character vector. Labels of directories to include in the upload. Options are the labels of directories in the active `projr` configuration.
`path`	character. Path to the directory. If a relative path is given, then it is taken as relative to the project home directory. Must be supplied.
`structure`	"latest" or "archive". Structure of the remote. If "latest", then `path` simply contains the latest versions of the contents. If "version", then `path` will contain a directory for each version. If not supplied, will be `archive`.
`overwrite`	logical. Whether to rewrite an existing entry of the same title in the specified `projr` configuration file. Default is TRUE.
`send_cue`	"always", "if-change" or "never". When to send to the remote. Default is NULL.
`send_strategy`	"upload-all", "upload-missing", "sync-purge" or "sync-diff". How to synchronise to the remote. Default is NULL.
`send_inspect`	"manifest" , "file" or "none". What to look at to find what are the files on the remote, and their versions. Default is NULL.
`profile`	character. Profile to write the settings to. If "default", then written to `_projr.yml`, otherwise written to `_projr-<profile>.yml`. The default is "default".

projr_yml_dir_license_get

Get license configuration for a directory

Description

Get license configuration for a specific directory label from `_projr.yml`.

Usage

```
projr_yml_dir_license_get(label, profile = "default")
```

Arguments

label	character. Directory label (e.g., "output", "raw-data", "docs", "cache").
profile	character. Profile to use. Default is "default".

Value

License configuration (character or list) or NULL if no license is configured.

Examples

```
## Not run:
# Get license for output directory
projr_yml_dir_license_get("output")

## End(Not run)
```

projr_yml_dir_license_rm

Remove license configuration for a directory

Description

Remove license configuration for a specific directory label from `_projr.yml`.

Usage

```
projr_yml_dir_license_rm(label, profile = "default")
```

Arguments

label	character. Directory label (e.g., "output", "raw-data", "docs", "cache").
profile	character. Profile to use. Default is "default".

Value

Invisible TRUE if license configuration was removed.

Examples

```
## Not run:
# Remove license configuration for output directory
projr_yml_dir_license_rm("output")

## End(Not run)
```

`projr_yml_dir_license_set` *Set license for a directory*

Description

Set license configuration for a specific directory label in `_projr.yml`. Licenses are automatically generated during builds.

Usage

```
projr_yml_dir_license_set(
  type,
  label,
  authors = NULL,
  year = NULL,
  profile = "default"
)
```

Arguments

`type`	character. License type. Supported types: "CC-BY", "CC0", "Apache-2.0", "MIT", "Proprietary". Common variations are also accepted (e.g., "ccby", "apache", "mit").
`label`	character. Directory label (e.g., "output", "raw-data", "docs", "cache").
`authors`	character vector. Authors or copyright holders. If NULL, attempts to get from DESCRIPTION file or uses "Project Authors" as default.
`year`	integer. Copyright year. If NULL, uses current year.
`profile`	character. Profile to use. Default is "default".

Value

Invisible TRUE if license was set successfully.

Examples

```
## Not run:
# Simple format - just license type
projr_yml_dir_license_set("CC-BY", "output")

# Full format with custom authors and year
projr_yml_dir_license_set(
  "MIT",
  "raw-data",
  authors = c("Jane Doe", "John Smith"),
  year = 2024
)

# Set license for multiple directories
projr_yml_dir_license_set("Apache-2.0", "output")
projr_yml_dir_license_set("Apache-2.0", "docs")

## End(Not run)
```

`projr_yml_dir_license_update`

Update LICENSE files with current DESCRIPTION authors

Description

Regenerates LICENSE files for directories with existing license configurations, using authors from the DESCRIPTION file. This is useful after updating package authors or for propagating authors to raw data directories.

Usage

```
projr_yml_dir_license_update(labels = NULL, profile = "default")
```

Arguments

`labels`	character vector. Directory labels to update. If NULL (default), updates all directories with license configurations.
`profile`	character. Profile to use. Default is "default".

Value

Invisible character vector of labels that were updated.

Examples

```
## Not run:
# Update all directories with license configurations
projr_yml_dir_license_update()

# Update specific directories
projr_yml_dir_license_update(c("raw-data", "output"))

## End(Not run)
```

`projr_yml_dir_path_set` *Set directory path*

Description

`projr_yml_dir_path_set` sets the path for a directory label in the project.

`projr_yml_dir_path_rm` removes the custom path setting for a directory, reverting to default behavior.

Usage

```
projr_yml_dir_path_set(label, path, profile = "default")

projr_yml_dir_path_rm(label, profile = "default")
```

Arguments

`label`	character. Directory label to configure (e.g., "output", "raw-data", "cache"). Must be a valid directory label.
`path`	character. Path to the directory.
`profile`	character. Profile to modify. If `"default"` (the default), modifies the default profile (`_projr.yml`). If another character vector, modifies `_projr-<profile>.yml`.

Examples

```
## Not run:
# Set path for output directory
projr_yml_dir_path_set("output", "_my_output")

# Set path for cache directory
projr_yml_dir_path_set("cache", "_my_cache")

# Revert to default path
projr_yml_dir_path_rm("output")

## End(Not run)
```

projr_yml_get *Get active* projr *settings and checks for validity*

Description

Gets active projr settings, which merges settings and resolves conflicts between local (_projr-local.yml), profile (_projr-<projr>.yml or as - keys in _projr.yml) and default (_projr.yml) settings. Where there are conflicts, local settings has highest precedence (i.e. are always preferred) and default settings have lowest precedence (i.e. are never preferred).

Note that an error is thrown if the active settings are invalid.

Usage

```
projr_yml_get(profile = NULL, check = FALSE)
```

Arguments

profile	character. projr profile to use. If NULL, then the active profile is used. Default is NULL.
check	logical. Whether to check the validity of the settings. Default is FALSE.

Value

A named list, if the settings are valid.

See Also

.yml_get_unchecked.yml_check

projr_yml_git_set *Set Git options*

Description

.yml_git_set‘ sets Git options for the project.

The options are:

- commit: whether to commit changes automatically upon project builds.
- add_untracked: whether to add untracked files automatically upon project builds.
- push: whether to push changes automatically upon project builds.

The default is to leave all the settings unchanged.

If these settings are not setting in _projr.yml, then the default is to commit, add untracked files and push.

.yml_git_set_default sets all Git options to default (TRUE‘).

Usage

```
projr_yml_git_set(
  all = NULL,
  commit = NULL,
  add_untracked = NULL,
  push = NULL,
  simplify_identical = TRUE,
  simplify_default = TRUE,
  profile = "default"
)

projr_yml_git_set_default(
  profile = "default",
  simplify_identical = TRUE,
  simplify_default = TRUE
)
```

Arguments

all	logical. Whether to set all the options to TRUE or FALSE. If NULL, then commit, add_untracked and push are used. Default is NULL.
commit	logical. Whether to commit changes automatically upon project builds. If NULL, then setting is not changed. Default is NULL.
add_untracked	logical. Whether to add untracked files automatically upon project builds. If NULL, then setting is not changed. Default is NULL.
push	logical. Whether to push changes automatically upon project builds. If NULL, then setting is not changed. Default is NULL.
simplify_identical	logical. If TRUE, then if all the settings are the same (for commit, push and add_untracked), then only git: TRUE or git: FALSE is written to _projr.yml. Default is TRUE.
simplify_default	logical. If TRUE, then if the settings are the same as the default (which is TRUE), then the settings are removed from _projr.yml. Default is TRUE.
profile	character. Profile to add the script to. If "default"`` (the default), the script is added to the de local.yml, _projr-.ymland_projr.yml) and written to _projr.yml. If another character vector, t .yml‘.

Examples

```
## Not run:
# set all to TRUE
projr_yml_git_set(all = TRUE)

# set all to FALSE
projr_yml_git_set(all = FALSE)

# set only add_untracked to FALSE
```

```
projr_yml_git_set(add_untracked = FALSE)

# revert to defaults
projr_yml_git_set_default()

## End(Not run)
```

`projr_yml_hooks_add` *Build hook-related functions*

Description

Convenience functions to add or remove hooks to run before or after the build.

- `projr_yml_hooks_add`: Add hook script(s) to run before or after the build.
- `projr_yml_hooks_rm_all`: Remove all hooks.
- `projr_yml_hooks_add_pre`: Add hook(s) to run before the build.
- `projr_yml_hooks_add_post`: Add hook(s) to run after the build.

Usage

```
projr_yml_hooks_add(path, stage, overwrite = TRUE, profile = "default")

projr_yml_hooks_rm_all(profile = "default")

projr_yml_hooks_add_pre(path, overwrite = TRUE, profile = "default")

projr_yml_hooks_add_post(path, overwrite = TRUE, profile = "default")
```

Arguments

`path`	character vector. Path(s) to hook scripts, relative to project root (if not absolute).
`stage`	"pre", "post", or "both". Whether to run the hook before the build ("pre"), after the build ("post"), or in both stages ("both"). Hooks with stage "pre" are stored under `build.hooks.pre`, hooks with stage "post" are stored under `build.hooks.post`, and hooks with stage "both" are stored under `build.hooks.both`.
`overwrite`	logical. Whether to overwrite existing hooks or append to them. Default is `TRUE`.
`profile`	character. Profile to add the hook to. If `"default"` (the default), the hook is added to the default profile, which is `_projr.yml`.

Details

Within a stage (pre- or post-build), hooks are run in the order specified in `_projr.yml`. They are not run in the same environment as the build process. The pre-build hooks are run immediately after bumping the project version (if that is done) and immediately before committing the present state of the code to Git. The post-build hooks are run immediately after committing the present state of the code to Git, and before distributing project artifacts to the remotes.

Hooks are stored as simple character vectors in the YAML:

```
build:
  hooks:
    pre: ["pre-hook.R"]
    post: ["post-hook.R"]
    both: ["both-hook.R"]
```

`projr_yml_par_add` *Add the* `parameters` *key*

Description

Add the `parameters` key to the `projr` configuration.

Usage

```
projr_yml_par_add(profile = "default")
```

Arguments

`profile`	character. If `NULL`, then the default profile is used. Default is `"default"`.

`projr_yml_renv_set` *Set renv snapshot options*

Description

`projr_yml_renv_set` sets renv snapshot options for the project.

The option controls whether `renv::snapshot()` is called before and after project builds.

If the setting is not present in `_projr.yml`, then the default is `TRUE` (renv snapshots are performed).

Usage

```
projr_yml_renv_set(renv = NULL, simplify_default = TRUE, profile = "default")
```

Arguments

`renv`	logical. Whether to snapshot renv before and after builds. If `NULL`, then setting is not changed. Default is `NULL`.
`simplify_default`	logical. If `TRUE`, then if the setting is the same as the default (which is `TRUE`), then the setting is removed from `_projr.yml`. Default is `TRUE`.
`profile`	character. Profile to add the setting to. If `"default"` (the default), the setting is added to the default profile, which is `_projr.yml`. If `NULL`, then the active profile is used (i.e the merge of `_projr-local.yml`, `_projr-<profile>.yml` and `_projr.yml`) and written to `_projr.yml`. If another character vector, then the corresponding profile is used and written to `_projr-<profile>.yml`.

Examples

```
## Not run:
# enable renv snapshots (default)
projr_yml_renv_set(TRUE)

# disable renv snapshots
projr_yml_renv_set(FALSE)

# revert to default (removes setting from YAML)
projr_yml_renv_set(TRUE, simplify_default = TRUE)

## End(Not run)
```

`projr_yml_restrictions_set` *Set build restrictions*

Description

`projr_yml_restrictions_set` sets build restrictions in `_projr.yml`.

The options are:

- branch: controls which branches can perform builds. If `TRUE` (default), builds are allowed on any branch. If a character vector, builds are only allowed on matching branches. If `FALSE`, builds are restricted (treated as empty character vector).
- not_behind: whether to check if branch is behind remote upstream. If `TRUE` (default), build fails if branch is behind remote. If `FALSE`, no check is performed.

Usage

```
projr_yml_restrictions_set(
  branch = NULL,
  not_behind = NULL,
  profile = "default"
)
```

Arguments

`branch`	logical or character. Controls which branches can perform builds. If `TRUE`, builds allowed on any branch (default). If a character vector, builds only allowed on these branches. If `FALSE`, builds restricted on all branches. If `NULL`, setting is not changed. Default is `NULL`.
`not_behind`	logical. Controls whether to check if branch is behind remote upstream. If `TRUE` (default), build fails if branch is behind. If `FALSE`, no check is performed. If `NULL`, setting is not changed. Default is `NULL`.
`profile`	character. The profile to write to. Default is "default", in which case it writes to `_projr.yml`.

Examples

```
## Not run:
# Allow builds on any branch (default)
projr_yml_restrictions_set(branch = TRUE)

# Allow builds only on main and dev branches
projr_yml_restrictions_set(branch = c("main", "dev"))

# Restrict builds on all branches
projr_yml_restrictions_set(branch = FALSE)

# Disable check for being behind remote
projr_yml_restrictions_set(not_behind = FALSE)

# Enable check for being behind remote (default)
projr_yml_restrictions_set(not_behind = TRUE)

## End(Not run)
```

`projr_yml_script_add` *Build script-related functions*

Description

Convenience functions to add or remove scripts to run before or after the build.

- .yml_script_add‘: Add a script to run before or after the build.
- .yml_script_rm‘: Remove scripts to run.

.yml_script_add_pre and `.yml_script_add_post` are wrappers around .yml_script_addthat `set` thestageargument `to`"pr
removes all scripts.

Usage

```
projr_yml_script_add(
  path,
  title,
  stage,
  cue = NULL,
  overwrite = TRUE,
  profile = "default"
)

projr_yml_script_rm(title, path = NULL, profile = "default")

projr_yml_script_rm_all(profile = "default")
```

```
projr_yml_script_add_pre(
  path,
  title,
  cue = NULL,
  overwrite = TRUE,
  profile = "default"
)

projr_yml_script_add_post(
  path,
  title,
  cue = NULL,
  overwrite = TRUE,
  profile = "default"
)
```

Arguments

path	character vector. Path(s) to scripts, relative to project root (if not absolute).
title	character. Title for set of scripts. Initial and trailing spaces are removed, and the middle spaces are converted to dashes. For example, " a b " is converted to "a-b". ‘
stage	"pre" or "post". Whether to run the script before or after the build.
cue	"build", "dev", "patch", "minor" or "major". Which minimum build level triggers the scripts. "build" and "dev" are equivalent, and always trigger the scripts.
overwrite	logical. Whether to overwrite any script settings of the same title in the projr configuration file. If FALSE and there already exists a key under build/script with the name title, an error is thrown. Default is TRUE.
profile	character. Profile to add the script to. If "default"`` (the default), the script is added to the de

Details

Within a stage (pre- or post-build), scripts are run in the order set in _projr.yml. They are not run in the same environment as the build process. The pre-build scripts are run immediately after bumping the project version (if that is done) and immediately before committing the present state of the code to Git. The post-build scripts are run immediately after committing the present state of the code to Git, and before distributing project artefacts to the remotes.

Index