Package: blaseRtemplates (via r-universe)

August 31, 2024

Title Generates a structured R project with custom templates.

Version 0.0.0.9210

Description This package generates an R project with customized templates. It also packages bash scripts for blaserlab collaborative git workflow.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL https://github.com/blaserlab/blaseRtemplates, https://blaserlab.github.io/blaseRtemplates/

BugReports https://github.com/blaserlab/blaseRtemplates/issues

Imports fs, gert, stringr, usethis, prompt, dplyr, gitcreds, renv, withr, devtools, purrr, tibble, pak (>= 0.2.1), cli (>= 3.2.0), rlang, rstudioapi, pkgcache, digest, readr, rjson

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

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

RemoteUrl https://github.com/blaserlab/blaseRtemplates

RemoteRef HEAD

RemoteSha 4654f891a09c8852638f4fb4b979b68bea6654ce

Contents

`cache_fun` *cache a single package*

Description

cache a single package

Usage

```
cache_fun(
  package,
  cache_loc = fs::path(Sys.getenv("BLASERTEMPLATES_CACHE_ROOT"), "library"),
  permissions
)
```

catch_blasertemplates_root

Make sure you are in a properly-formatted blaseRtemplates project.

Description

Make sure you are in a properly-formatted blaseRtemplates project.

Usage

```
catch_blasertemplates_root()
```

establish_new_bt

Establish a New blaseRtemplates Installation

Description

This function sets up a directory structure that is designed to work well with blaseRtemplates functions. It will create directories where indicated, write a standard .Rprofile document, and modify the user .Renviron file to configure R properly. The function will not overwrite existing directories and will fail when an existing installation is already present in the same location. The only modification outside of the existing directory is to the .Renviron file. If you have existing configurations in your .Renviron file, they will only be changed if they are conflicting. If this is problematic, then you should archive your .Renviron file before running. The new .Renviron file will cause R to use the new .Rprofile in the cache directory and skip existing user- and project-level .Rprofiles.

For the individual user: Identify the location for your blaseRtemplates cache and your R projects. For convenience, these can be within the same parent directory. The cache will hold versioned and hashed binary package files in the library directory. These are the actual instructions used by R when you call up functions. The version/hash structure is tracked by your projects and allows reproducibility. The user_project directory holds symlinks to the specific version/hash packages being used. A new directory is created for each new project. The source directory is used by the package installer, pak, to archive the source files for each package. You should rarely need to look in there. The logs directory holds .Rhistory files. One is created for each R session you open. The .tsv files catalog the entire collection of packages in your cache with their dependencies. The .Rprofile is what sets up your R session to use the proper user_package directory, ensures you are working in a valid project, generates startup messages and sets some helpful options.

This function will create 1 project, called baseproject, in the projects directory which will ensure you are always operating in a project environment. Working outside of a project can have dangerous/unintended consequences.

For the system administrator: the directory structure will work equally well for a multiuser system (e.g. rstudio server) with minor modifications. The .Rprofile generated by the function can be used as Rprofile.site and will apply to all users of that R installation. Each user should have their own project directories with a baseproject on the system. The .Renviron file should be modified for each user to point to this directory and saved in the home directory. There should be a subdirectory

in user_projects with each user's ID for their project libraries. Ownership should be given to the user/group and permissions set accordingly, e.g. 755.

Upgrading R versions: since package binaries don't work across minor version changes in R, e.g. 4.2 -> 4.3, you will have to create a new cache directory each time this changes.

Usage

```
establish_new_bt(cache_path, project_path)
```

Arguments

cache_path	Path to the blaseRtemplates cache root. Should include the R major and minor version numbers in the final directory. This and all intermediate directories will be created.
project_path	Path to the R projects directory. For convenience, can put this in the same parent directory as cache_path, but not strictly necessary.

See Also

cli_alert create, copy, path_package, path, file_access, dir_ls, path_file str_detect, str_replace proj_utils

Examples

```
## Not run:
if(interactive()){
 establish_new_bt(cache_path = "<some_directory>/r_4_2_cache", project_path = "<some_directory>/projects")
 }

## End(Not run)
```

---

find_unlinked_packages

Find Unlinked Packages

Description

Find Unlinked Packages

Usage

```
find_unlinked_packages(lib_path)
```

fix_another_library *Fix a User-Project Library*

Description

Sometimes, the user_project library can break. This can happen if there are failures in the upstream install functions. Or if the links are deleted for some reason. If everything is broken, you may need to repair the user project library. To do this, exit the project and enter a working project. Delete the links in the offending user project library. Then run this function to relink. Use either the library catalog file from the project itself, or if this is also corrupted with bad information, run an older version that worked or a version from another project that worked. If you can identify the problematic package in the course of these fixes, then you should probably delete it from your cache entirely.

Usage

```
fix_another_library(file, dir)
```

Arguments

file	The library catalog tsv file to read from.
dir	The user project library to repair.

Value

Nothing

See Also

path_math, create, path, path_file read_delim pull

get_all_deps *get all package dependencies*

Description

get all package dependencies

Usage

```
get_all_deps(package)
```

get_new_library *Get A New Project Library*

Description

Use this to replace the current symlinked library with a new version. By default, the function will link to the newest version of all packages available in the cache. Alternatively, identify another project library catalog to replace the current version.

Usage

```
get_new_library(newest_or_file = "newest")
```

Arguments

newest_or_file	Which set of packages to symlink, Default: 'newest'

Value

Uninstalled packages hashes.

gitcreds_set *Interactively Set Github PAT*

Description

This wraps gitcreds::gitcreds_set() and adds a system call to edit the user global git config to set the cache timeout to 1 billion seconds if running a Linux system.

Usage

```
gitcreds_set()
```

Value

nothing

See Also

gitcreds_get

git_easy_branch *Easily Create or Switch Git Branches*

Description

Supply this function with a branch name. If the branch exists it will switch to the branch. If not, it will pull any changes from remote and then create the branch. Any uncommitted work will be carried over to the new branch in the same state. Avoid repeatedly switching branches with work in different states of completion since this may cause conflicts

Usage

```
git_easy_branch(branch)
```

Arguments

branch	A character string with the branch name to create or switch to.

Value

nothing git_branch

git_push_all *Push to All Remotes*

Description

This uses gert to look up all active remotes and then runs gert::git_push() to each.

Usage

```
git_push_all()
```

Value

nothing

git_rewind_to *Rewind Git History*

Description

This function uses git revert to rewind history to a prior commit. First make sure all of your changes have been committed. Then run gert::git_log() to identify the "good" commit you want to rewind to. Supply this as the argument to this function. A new commit will be made with a helpful message. Commit history is not changed so you can always rewind the rewind etc....

Usage

```
git_rewind_to(commit)
```

Arguments

commit	Hash of the commit you want to rewind the state of your repository to. Requires a minimum of 7 characters.

Value

a tibble with the new git commit log after rewinding

See Also

git_commit

git_safe_merge *Safely Merge your Working Branch*

Description

This function updates default branch (usually "main") from remote. This pulls in any changes from other contributors. Then it merges the working branch into the upstream branch.

Usage

```
git_safe_merge(branch = NULL, upstream = "main")
```

Arguments

branch	The working branch you wish to merge, Default: NULL
upstream	The default upstream branch you wish to merge into, Default: NULL

Value

nothing

See Also

git-default-branch git_branch,git_commit

`git_update_branch` *Update a Working Git Branch*

Description

This function updates a git branch via rebase from a default upstream branch (default is "main"). You can explicitly provide the names of your working branch and the default upstream branch.

Usage

```
git_update_branch(branch = NULL, upstream = "main")
```

Arguments

`branch`	The working branch you wish to update, Default: NULL
`upstream`	The default upstream branch you wish to update from, Default: NULL

Value

nothing

See Also

git-default-branch git_branch,git_stash

`hash_fun` *Hash a single package*

Description

Hash a single package

Usage

```
hash_fun(package)
```

hash_n_cache *hash one or more functions and then cache them and update the catalogs. Default permissions are set to 777.*

Description

hash one or more functions and then cache them and update the catalogs. Default permissions are set to 777.

Usage

```
hash_n_cache(
  lib_loc = .libPaths()[1],
  cache_loc = fs::path(Sys.getenv("BLASERTEMPLATES_CACHE_ROOT"), "library"),
  verbose = TRUE,
  permissions = "777"
)
```

initialize_github *Initialize A Project By Forking A Github Repo*

Description

This function wraps usethis::create_from_github, making some useful default choices. Because this function forks the project, git will set up the originator as an upstream remote. Using blaseRtemplates::git_push_all will push to both the originator and the collaborator's github.

Usage

```
initialize_github(repo, dest = NULL, open = TRUE)
```

Arguments

repo	The repo to clone. Must be in the form of github_user/repo_name. If private, you must be a collaborator and have permission to fork the repo from the owner.
dest	Destination directory. This directory will become the parent directory for the project you are forking. If NULL, the default, it will put the project in the directory defined by the usethis.destdir option. Set this in ~/.Rprofile.
open	Whether to open the forked project, Default: TRUE

Value

nothing

initialize_package *Initialize a Package Using a Standard Template*

Description

This wraps `usethis::create_package()` and adds a few additional templates.

Usage

```
initialize_package(
  path,
  fields = list(),
  roxygen = TRUE,
  check_name = TRUE,
  rstudio = rstudioapi::isAvailable(),
  open = rlang::is_interactive(),
  fresh_install = FALSE,
  path_to_cache_root = Sys.getenv("BLASERTEMPLATES_CACHE_ROOT")
)
```

Arguments

`path`	path/name for the new package. It should include letters and "." only to be CRAN-compliant.
`fields`	named list of fields in addition to/overriding defaults for the DESCRIPTION file, Default: list()
`roxygen`	do you plan to use roxygen2 to document package?, Default: TRUE
`check_name`	check if name is CRAN-compliant, Default: TRUE
`rstudio`	makes an Rstudio project, default is true
`open`	to open or not, Default: rlang::is_interactive()

See Also

`isAvailable` `is_interactive` `use_template` `defer`

initialize_project *Create a package or project using a structured template*

Description

These functions create an R project:

- `create_project()` creates a non-package project, i.e. a data analysis project

Both functions can be called on an existing project; you will be asked before any existing files are changed.

This function is a modification of usethis::create_project

Usage

```
initialize_project(
  path,
  rstudio = rstudioapi::isAvailable(),
  open = rlang::is_interactive(),
  fresh_install = FALSE,
  path_to_cache_root = Sys.getenv("BLASERTEMPLATES_CACHE_ROOT")
)
```

Arguments

path	A path. If it exists, it is used. If it does not exist, it is created, provided that the parent path exists.
rstudio	If TRUE, calls use_rstudio() to make the new package or project into an RStudio Project.
open	If TRUE, activates the new project: • If RStudio desktop, the package is opened in a new session. • If on RStudio server, the current RStudio project is activated. • Otherwise, the working directory and active project is changed.

Value

Path to the newly created project or package, invisibly.

---

install_one_package *Install One Package*

Description

Use this to install a new package. Choosing "new_or_update" will go to the package repository, get the latest version of the software, install into your cache and link to your project library. Choosing "link_from_cache" will get you the latest version in the cache Also, use this option with either "which_version" or "which_hash" to install specific versions.

Usage

```
install_one_package(
  package,
  how = c("link_from_cache", "new_or_update", "tarball"),
  which_version = NULL,
  which_hash = NULL
)
```

Arguments

package	Package name or path to tarball. Prefix with "repo\/" for github source packages and "bioc::" for bioconductor.
how	How to install the package, Default: c("link_from_cache", "new_or_update", "tarball")
which_version	Package version to install, Default: NULL
which_hash	Package hash to install, Default: NULL

Value

nothing

---

link_deps *link package dependencies*

Description

link package dependencies

Usage

```
link_deps(package)
```

---

link_one_new_package *link one new package*

Description

link one new package

Usage

```
link_one_new_package(package, version = NULL, hash = NULL)
```

`project_data` *Activate Project Data*

Description

Use this to update, install and/or load project data. Usual practice is to provide the path to a directory holding data package tarballs. This function will find the newest version, compare that to the versions in the cache and used in the package and give you the newest version. Alternatively, provide the path to a specific .tar.gz file to install and activate that one.

If a specific version is requested, i.e. a specific .tar.gz file, and this version is already cached, it will be linked and not reinstalled. If for some reason there are multiple hashes with the same version number (usually because a package was rebuilt without incrementing the version), then the latest hash of that version will be linked.

This function now accepts multiple paths, i.e. multiple independent data packages, in the form of a character vector of length >= 1. After deciding which version to install based on the inputs, the function will load all of the data objects into a single environment called deconflicted.data. The problem with loading multiple data packages into the same environment is that there may be name conflicts and objects get overridden. The problem with keeping them in separate environments is that they are difficult to specify and access. Here is how this function deals with these problems:

- If `length(path) > 1`, the function will require a vector for the argument deconflict_string of the same length. The first element of deconflict_string will be added as a suffix to the data object from the first package in path, etc. For example if the first value of the argument deconflict_string is ".my.project.data", then all objects in the package will be suffixed with .my.project.data.
- Note that you will have to reference the object correctly in your code using the proper suffix.
- Also note that all of the elements of deconflict_string must be unique. But an empty string, i.e. "", is also a valid input which means that all of the names of the data objects from that package will be unchanged. This is helpful if you have a lot of code using one data package but at a later time decide you need to add a different data package. Make the deconflict string `c("", ".my.new.data")` and you don't have to change any of your old code.
- Make sure you include a separator like . or _ but not a space as the first character of each element of deconflict_string.
- If only a single package is loaded, there will be no conflicts and by default, deconflict_string is set to "".

As before, all data elements are loaded as promises which means that they are loaded into memory only when called.

Usage

```
project_data(path, deconflict_string = "")
```

Arguments

path	Path or vector of paths to data directory/ies.
deconflict_string	Character vector used to disambiguate objects from packages in path, Default: ”

Value

loads data as promises as a side effect

See Also

cli_abort, cli_alert read_delim, cols path, path_file pmap, map str_detect, str_extract, str_remove, str_replace filter, pull, arrange, slice as_tibble

---

rec_get_deps *recursively get package dependencies*

Description

recursively get package dependencies

Usage

```
rec_get_deps(
  needed,
  checked = character(0),
  deps = character(0),
 catalog = fs::path(Sys.getenv("BLASERTEMPLATES_CACHE_ROOT"), "dependency_catalog.tsv")
)
```

---

regenerate_bt_configs *Regenerate .Rprofile and .Renviron Files*

Description

If you have deleted or otherwise broken your .Rprofile or .Renviron files, you may have difficulty connecting to the package cache. This function will regenerate both for you. The existing .Rprofile must be deleted manually. You can choose to archive the old version if you wish. It will be replaced with the standard .Rprofile from blaseRtemplates. The .Renviron file will be modified by removing the damaged lines and replacing them with the correct ones. You must supply the correct file path locations to your cache directory and project directory, otherwise your R installation will be configured incorrectly.

Usage

```
regenerate_bt_configs(cache_path, project_path)
```

Arguments

cache_path	path to the cache directory
project_path	path to the projects directory

Value

nothing

See Also

file_access, path, copy, path_package cli_abort, cli_alert str_detect, str_replace

---

regenerate_git_commands

Regenerate a Git Commands File

Description

Since this file is ignored by git, you will have to regenerate it when forking a repository. This function writes the template file to your R directory as "regenerated_git_commands.R".

Usage

```
regenerate_git_commands()
```

Value

nothing

---

report_template_rmd

Make an Rmd Report from the Saved Template

Description

Shortcut function to save the blaseRtemplates report template in Rmd format within an Rmd folder.

Usage

```
report_template_rmd(report_name = NULL)
```

Arguments

report_name	Name for the Rmd file, Default: NULL

Details

Makes a new Rmd file with the supplied file name

Value

Returns nothing

See Also

`str_detect` `cli_alert` `create`, `path` `use_template`

`reset_prompt` *Reset Your Console Prompt*

Description

The prompt package adds a nice feature but has some limitations, namely, that it does not respect changing git branches and has to be manually re-called. This defeats the purpose. Blasertemplates git functions automoatically call prompt to change the prompt label when switching branches, but this will not happen if you change branches using the terminal, the git panel or other git branching functions. Therefore this function is provided to manually reset your prompt to the current branch.

Usage

```
reset_prompt()
```

Value

nothing

See Also

`set_prompt`,`prompt_git`

`update_dependency_catalog`
update the dependency catalog

Description

update the dependency catalog

Usage

```
update_dependency_catalog()
```

`update_package_catalog` *rewrite/update the package catalog*

Description

rewrite/update the package catalog

Usage

```
update_package_catalog()
```

`write_project_library_catalog` *Write A New Project Library Catalog*

Description

In the current version of blaseRtemplates, the package library is cached at the location designated by the environment variable "BLASERTEMPLATES_CACHE_ROOT". There is a single cache for all users and projects. The cache holds the binary software used by each package. The packages for each project are connected to the cache by symlinks. The cache is versioned so that different projects can use different versions if desired. Use this function to write a tab-delimited file listing the packages used by each project.

This file will be written to the "library_catalogs" directory within each project. The filename incorporates the user name so everyone working on the project will have their own. Use `get_new_library()` to adopt a new version of all packages

Usage

```
write_project_library_catalog()
```

Value

returns nothing

Index