Package: datastorr (via r-universe)

June 1, 2026

Title Simple Data Versioning

Version 0.0.4

Author Rich FitzJohn

Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>

Description Simple dataversioning using GitHub to store data.

License MIT + file LICENSE

LazyData true

URL https://github.com/ropenscilabs/datastorr

BugReports https://github.com/ropenscilabs/datastorr/issues

Imports httr, jsonlite, rappdirs, storr (>= 1.0.0), tibble, stringr

Suggests knitr, rmarkdown, testthat (>= 0.11.0), whisker

RoxygenNote 7.0.2

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev libssl-dev

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

Date/Publication 2021-07-07 13:27:19 UTC

RemoteUrl https://github.com/traitecoevo/datastorr

RemoteRef HEAD

RemoteSha 22a6371aa3e54e3071491e145be85e88f6cdb34c

Contents

`autogenerate` *Autogenerate a datastorr interface*

Description

Autogenerate an datastorr interface for a package. The idea is to run this function and save the resulting code in a file in your package. Then users will be able to download data and you will be able to relase data easily.

Usage

```
autogenerate(repo, read, filename = NULL, name = basename(repo),
  roxygen = TRUE)
```

Arguments

`repo`	Name of the repo on github (in username/repo format)
`read`	*name* of a function to read the data. Do not give the function itself!
`filename`	Name of the file to read. If not given, then the single file in a release will be read (but you will need to provide a filename on upload). If given, you cannot change the filename ever as all releases will be assumed to have the same filename.
`name`	Name of the dataset, used in generating the functions. If omitted the repo name is used.
`roxygen`	Include roxygen headers for the functions?

Details

In addition to running this, you will need to add `datastorr` to the `Imports:` section of your DESCRIPTION. To upload files you will need to set your `GITHUB_TOKEN` environment variable. These steps will be described more fully in a vignette.

More complete instructions:

Let `pkg` be `basename(repo)`; the name of the package and of the GitHub repository.

First, create a new R package, e.g. `devtools::create(pkg)`.

Then, copy the result of running `autogenerate` into a file in that package, e.g.

```
  writeLines(autogenerate(repo, read),
             file.path(pkg, "datastorr.R"))
devtools::document(pkg)
```

Create a new git repository for this package, and add all the files in the package, and commit.

On GitHub, create a repository for the package and push your code there.

At this point you are now ready to start making releases by loading your package and running `pkg::<name>_release()`.

Examples

```
writeLines(autogenerate("richfitz/datastorr.example",
                        read = "readRDS", name = "mydata"))
writeLines(autogenerate("richfitz/datastorr.example",
                        read = "readRDS", name = "mydata",
                        roxygen = FALSE))
```

`datastorr` *Fetch data from a datastorr repository*

Description

Create a lightweight datastorr interface (rather than using the full package approach). This approach is designed for the "files that don't fit in git" use-case.

Usage

```
datastorr(repo, path = NULL, metadata = "datastorr.json",
  branch = "master", private = FALSE, refetch = FALSE,
  version = NULL, extended = FALSE)

datastorr_versions(..., local = TRUE)
```

Arguments

`repo`	Either a github repo in the form `<username>/<repo>` (e.g., `"richfitz/data"` or the path to a json file on your filesystem.
`path`	The path to store the data at. Using `NULL` will
`metadata`	The name of the metadata file within the repo (if `repo` refers to a github repo. The default is `datastorr.json` at the root of the repository, but any other filename can be used.
`branch`	The branch in the repo to use. Default is `master`.
`private`	A logical indicating if the repository is private and therefor if authentication will be needed to access it.
`refetch`	Refetch the metadata file even if it has already been downloaded previously.
`version`	Which version to download (if `extended` is `FALSE` – the default). By default the most recent version on the remote, or the current version locally will be fetched.
`extended`	Don't fetch the data, but instead return an object that can query data, versions, etc.
`...`	Arguments passed through to `datastorr`
`local`	Return information on local versions?

Details

Note that the package approach is likely to scale better; in particular it allows for the reading function to be arbitrarily complicated, allows for package installation and loading, etc. With this simple interface you will need to document your dependencies carefully. But it does remove the requirement for making a package and will likely work pretty well as part of an analysis pipeline where your dependencies are well documented anyway.

Examples

```
## Not run:
path <- tempfile()
dat <- datastorr("richfitz/data", path, extended = TRUE)
dat$list()
dat()

## End(Not run)
```

`datastorr_auth` *datastorr/GitHub authentication*

Description

Authentication for accessing GitHub. This will first look for a GitHub personal token (stored in the `GITHUB_TOKEN` or `GITHUB_PAT` environment variables, and then try authenticating with OAuth.

Usage

```
datastorr_auth(required = FALSE, key = NULL, secret = NULL,
  cache = TRUE, token_only = FALSE)

setup_github_token(path = "~/.Renviron")
```

Arguments

`required`	Is authentication required? Reading from public repositories does not require authentication so there's no point worrying if we can't get it. datastorr will set this when appropriate internally.
`key, secret`	The application key and secret. If `NULL`, uses datastorr's key. But if you have your own application feel free to replace these with your own.
`cache`	Logical, indicating whether we should cache the token. If `TRUE`, the token will be cached at `datastorr_auth()`, so that it is accessible to all datastorr usages. Note that this is affected by the `datastorr.path` global option. Alternatively, set `FALSE` to do no caching and be prompted each session or a string to choose your own filename. Or set the `GITHUB_TOKEN` or `GITHUB_PAT` environment variables to use a token rather than OAuth.
`token_only`	return the token only
`path`	Path to environment file; the default is the user environment variable file which is usually a good choice.

Details

Run this `datastorr_auth` function to force setting up authentication with OAuth. Alternatively, run `setup_github_token` to set up a personal access token. Either can be revoked at any time https://github.com/settings/tokens to revke a personal access token and https://github.com/settings/applications to revoke the OAuth token.

`datastorr_path` *Location of datastorr files*

Description

Location of datastorr files. This is determined by `rappdirs` using the `user_data_dir` function. Alternatively, if the option `datastorr.path` is set, that is used for the base path. The path to data from an actual repo is stored in a subdirectory under this directory.

Usage

```
datastorr_path(repo = NULL)
```

Arguments

`repo`	An optional repo (of the form `user/repo`, though this is not checked).

Details

Files in this directory can be deleted at will (e.g., running `unlink(datastorr_path(), recursive = TRUE)` will delete all files that datstorr has ever downloaded. The only issue here is that the OAuth token (used to authenticate with GitHub) is also stored in this directory.

`github_release_create` *Create a github release*

Description

Create a github release for your package. This tries very hard to do the right thing but it's not always straightforward. It first looks for your package. Then it will work out what your last commit was (if `target` is NULL), the version of the package (from the DESCRIPTION). It then creates a release on GitHub with the appropriate version number and uploads the file `filename` to the release. The version number in the DESCRIPTION must be greater than the highest version number on GitHub.

Usage

```
github_release_create(info, description = NULL, filenames = NULL,
  target = NULL, ignore_dirty = FALSE, yes = !interactive())
```

Arguments

info	Result of running github_release_info
description	Optional text description for the release. If this is omitted then GitHub will display the commit message from the commit that the release points at.
target	Target of the release. This can be either the name of a branch (e.g., master, origin/master), existing tag *without a current release* or an SHA of a commit. It is an error if the commit that this resolves to locally is not present on GitHub (e.g., if your branch is ahead of GitHub). Push first!
ignore_dirty	Ignore non-checked in files? By default, your repository is expected to be in a clean state, though files not known to git are ignored (as are files that are ignored by git). But you must have no uncommited changes or staged but uncommited files.
yes	Skip the confirmation prompt? Only prompts if interactive.
filename	Filename to upload; optional if in info. If listed in info, filename can be different but the file will be renamed to info$filename on uploading. If given but not in info, the uploaded file will be basename(filename) (i.e., the directory will be stripped).

Details

This function requires a system git to be installed and on the path. The version does not have to be particularly recent.

This function also requires the GITHUB_TOKEN environment variable to be set, and for the token to be authorised to have write access to your repositories.

---

github_release_del *Delete version*

Description

Delete a local copy of a version (or all local copies). Note that that does not affect the actual github release in any way!.

Usage

```
github_release_del(info, version)
```

Arguments

info	Result of running github_release_info
version	Version to delete. If NULL it will delete the entire storr

github_release_get *Get data*

Description

Get a version of a data set, downloading it if necessary.

Usage

```
github_release_get(info, version = NULL)
```

Arguments

info	Result of running github_release_info
version	Version to fetch. If NULL it will get the current version as returned by github_release_version_current

github_release_info *Github release information*

Description

Information to describe how to process github releases

Usage

```
github_release_info(repo, read, private = FALSE, filename = NULL,
  path = NULL)
```

Arguments

repo	Name of the repo in username/repo format.
read	Function to read the file. See Details.
private	Is the repository private? If so authentication will be required for all actions. Setting this is optional but will result in better error messages because of the way GitHub returns not found/404 (rather than forbidden/403) errors when accessing private repositories without authorisation.
filename	Optional filename. If omitted, all files in the release can be used. If the filename contains a star ("*") it will be treated as a filename glob. So you can do filename = "*.csv" to match all csv files (dynamically computed on each release).
path	Optional path in which to store the data. If omitted we use datastorr_path to generate a reasonable path.

Details

The simplest case is where the data are stored in a single file attached to the release (this is different to the zip/tar.gz files that the web interface displays). For example, a single csv file. In that case the filename argument can be safely ommited and we'll work it out based on the filename.

github_release_versions *Get release versions*

Description

Get release versions

Usage

```
github_release_versions(info, local = TRUE)

github_release_version_current(info, local = TRUE)
```

Arguments

info	Result of running github_release_info
local	Should we return local (TRUE) or github (FALSE) version numbers? Github version numbers are pulled once per session only. The exception is for github_release_version_curre which when given local = TRUE will fall back on trying github if there are no local versions.

Author(s)

Rich FitzJohn

release *Release data to a datastorr repository*

Description

Create a relase for a simple datastorr (i.e., non-package based).

Usage

```
release(repo, version, description = NULL, filename = NULL,
  path = NULL, metadata = "datastorr.json", branch = "master",
  private = FALSE, refetch = FALSE, target = NULL,
  ignore_dirty = FALSE, yes = !interactive())
```

Arguments

repo	Either a github repo in the form <username>/<repo> (e.g., "richfitz/data" or the path to a json file on your filesystem.
version	A version number for the new version. Should be of the form x.y.z, and may or may not contain a leading "v" (one will be added in any case).
description	Optional text description for the release. If this is omitted then GitHub will display the commit message from the commit that the release points at.
filename	Filename to upload; optional if in datastorr.json. If listed, filename can be different but the file will be renamed on uploading. If given but not in info, the uploaded file will be basename(filename) (i.e., the directory will be stripped).
path	The path to store the data at. Using NULL will
metadata	The name of the metadata file within the repo (if repo refers to a github repo. The default is datastorr.json at the root of the repository, but any other filename can be used.
branch	The branch in the repo to use. Default is master.
private	A logical indicating if the repository is private and therefor if authentication will be needed to access it.
refetch	Refetch the metadata file even if it has already been downloaded previously.
target	The SHA or tag to attach the release to. By default, will use the current HEAD, which is typically what you want to do.
ignore_dirty	Ignore non-checked in files? By default, your repository is expected to be in a clean state, though files not known to git are ignored (as are files that are ignored by git). But you must have no uncommited changes or staged but uncommited files.
yes	Skip the confirmation prompt? Only prompts if interactive.

Index