Package 'piggyback'

October 6, 2018

Version 0.0.8

Title Managing Larger Data on a GitHub Repository

Description Because larger (> 50 MB) data files cannot easily be committed to git, a different approach is required to manage data associated with an analysis in a GitHub repository. This package provides a simple work-around by allowing larger (up to 2 GB) data files to piggyback on a repository as assets attached to individual GitHub releases. These files are not handled by git in any way, but instead are uploaded, downloaded, or edited directly by calls through the GitHub API. These data files can be versioned manually by creating different releases. This approach works equally well with public or private repositories. Data can be uploaded and downloaded programmatically from scripts. No authentication is required to download data from public repositories.

URL https://github.com/ropensci/piggyback

BugReports https://github.com/ropensci/piggyback/issues

License GPL-3

Encoding UTF-8

LazyData true

ByteCompile true

Imports gh, httr, jsonlite, git2r, fs, usethis, crayon, clisymbols, magrittr, lubridate, memoise

Suggests spelling, readr, covr, testthat, datasets, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.1.0

Language en-US

NeedsCompilation no

Author Carl Boettiger [aut, cre, cph] (<https://orcid.org/0000-0002-1642-628X>), Mark Padgham [ctb] (<https://orcid.org/0000-0003-2172-5265>)

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2018-10-06 21:00:11 UTC

R topics documented:

`piggyback-package` *piggyback: Managing Larger Data on a GitHub Repository*

Description

Because larger (> 50 MB) data files cannot easily be committed to git, a different approach is required to manage data associated with an analysis in a GitHub repository. This package provides a simple work-around by allowing larger (up to 2 GB) data files to piggyback on a repository as assets attached to individual GitHub releases. These files are not handled by git in any way, but instead are uploaded, downloaded, or edited directly by calls through the GitHub API. These data files can be versioned manually by creating different releases. This approach works equally well with public or private repositories. Data can be uploaded and downloaded programmatically from scripts. No authentication is required to download data from public repositories.

Details

It has two main modes or workflows:

- `pb_upload()` / `pb_download()`: Upload and download individual files to/from the desired release of the specified repository
- `pb_track()`: Use a `git-lfs` style tracking of specific file types

Author(s)

Maintainer: Carl Boettiger <cboettig@gmail.com> (0000-0002-1642-628X) [copyright holder]

Other contributors:

- Mark Padgham (0000-0003-2172-5265) [contributor]

See Also

Useful links:

- https://github.com/ropensci/piggyback
- Report bugs at https://github.com/ropensci/piggyback/issues

pb_delete *Delete an asset attached to a release*

Description

Delete an asset attached to a release

Usage

```
pb_delete(file = NULL, repo = guess_repo(), tag = "latest",
  .token = get_token())
```

Arguments

file	file(s) to be deleted from the release. If NULL (default when argument is omitted), function will delete all attachments to the release. delete
repo	Repository name in format "owner/repo". Will guess the current repo if not specified.
tag	tag for the GitHub release to which this data should be attached.
.token	GitHub authentication token. Typically set from an environmental variable, e.g. in a .Renviron file or with Sys.setenv(GITHUB_TOKEN = "xxxxx"), which helps prevent accidental disclosure of a secret token when sharing scripts.

Value

TRUE (invisibly) if a file is found and deleted. Otherwise, returns NULL (invisibly) if no file matching the name was found.

Examples

```
readr::write_tsv(mtcars, "mtcars.tsv.gz")
## Upload
pb_upload("mtcars.tsv.gz",
          repo = "cboettig/piggyback-tests",
           overwrite = TRUE)
pb_delete("mtcars.tsv.gz",
          repo = "cboettig/piggyback-tests",
          tag = "v0.0.1")
```

pb_download *Download data from an existing release*

Description

Download data from an existing release

Usage

```
pb_download(file = NULL, dest = usethis::proj_get(),
  repo = guess_repo(), tag = "latest", overwrite = TRUE,
  ignore = "manifest.json", use_timestamps = TRUE,
  show_progress = TRUE, .token = get_token())
```

Arguments

file	name or vector of names of files to be downloaded. If NULL, all assets attached to the release will be downloaded.
dest	name of vector of names of where file should be downloaded. Should be a directory or a list of filenames the same length as file vector. Can include paths to files, but any directories in that path must already exist.
repo	Repository name in format "owner/repo". Will guess the current repo if not specified.
tag	tag for the GitHub release to which this data is attached
overwrite	Should any local files of the same name be overwritten? default TRUE.
ignore	a list of files to ignore (if downloading "all" because file=NULL).
use_timestamps	If TRUE, then files will only be downloaded if timestamp on GitHub is newer than the local timestamp (if overwrite=TRUE). Defaults to TRUE.
show_progress	logical, should we show progress bar for download? Defaults to TRUE.
.token	GitHub authentication token. Typically set from an environmental variable, e.g. in a .Renviron file or with Sys.setenv(GITHUB_TOKEN = "xxxxx"), which helps prevent accidental disclosure of a secret token when sharing scripts.

Examples

```
 ## Download a specific file.
 ## (dest can be omitted when run inside and R project)
 piggyback::pb_download("data/iris.tsv.gz",
                        repo = "cboettig/piggyback-tests",
                        dest = tempdir())

## Not run:
 ## Download all files
 piggyback::pb_download(repo = "cboettig/piggyback-tests",
```

```
                        dest = tempdir())

## End(Not run)
```

---

pb_download_url *Get the download url of a given file*

Description

Returns the URL download for a public file. This can be useful when writing scripts that may want to download the file directly without introducing any dependency on `piggyback` or authentication steps.

Usage

```
pb_download_url(file = NULL, repo = guess_repo(), tag = "latest",
  .token = get_token())
```

Arguments

`file`	name or vector of names of files to be downloaded. If `NULL`, all assets attached to the release will be downloaded.
`repo`	Repository name in format "owner/repo". Will guess the current repo if not specified.
`tag`	tag for the GitHub release to which this data is attached
`.token`	GitHub authentication token. Typically set from an environmental variable, e.g. in a `.Renviron` file or with `Sys.setenv(GITHUB_TOKEN = "xxxxx")`, which helps prevent accidental disclosure of a secret token when sharing scripts.

Value

the URL to download a file

Examples

```
pb_download_url("data/iris.tsv.xz",
                repo = "cboettig/piggyback-tests",
                tag = "v0.0.1")
```

pb_list *List all assets attached to a release*

Description

List all assets attached to a release

Usage

```
pb_list(repo = guess_repo(), tag = NULL, ignore = "manifest.json",
  .token = get_token())
```

Arguments

repo	Repository name in format "owner/repo". Will guess the current repo if not specified.
tag	which release tag do we want information for? If NULL (default), will return a table for all available release tags.
ignore	a list of files to ignore (if downloading "all" because file=NULL).
.token	GitHub authentication token. Typically set from an environmental variable, e.g. in a .Renviron file or with Sys.setenv(GITHUB_TOKEN = "xxxxx"), which helps prevent accidental disclosure of a secret token when sharing scripts.

Details

To preserve path information, local path delimiters are converted to .2f when files are uploaded as assets. Listing will display the local filename, with asset names converting the .2f escape code back to the system delimiter.

Value

a data.frame of release asset names, (normalized to local paths), release tag, timestamp, owner, and repo.

Examples

```
## Not run:
pb_list("cboettig/piggyback-tests")

## End(Not run)
```

pb_new_release *Create a new release on GitHub repo*

Description

Create a new release on GitHub repo

Usage

```
pb_new_release(repo = guess_repo(), tag, commit = "master",
  name = tag, body = "Data release", draft = FALSE,
  prerelease = FALSE, .token = get_token())
```

Arguments

repo	Repository name in format "owner/repo". Will guess the current repo if not specified.
tag	tag to create for this release
commit	Specifies the commit-ish value that determines where the Git tag is created from. Can be any branch or commit SHA. Unused if the git tag already exists. Default: the repository's default branch (usually `master`).
name	The name of the release. Defaults to tag.
body	Text describing the contents of the tag. default text is "Data release".
draft	default `FALSE`. Set to `TRUE` to create a draft (unpublished) release.
prerelease	default `FALSE`. Set to `TRUE` to identify the release as a pre-release.
.token	GitHub authentication token. Typically set from an environmental variable, e.g. in a `.Renviron` file or with `Sys.setenv(GITHUB_TOKEN = "xxxxx")`, which helps prevent accidental disclosure of a secret token when sharing scripts.

Examples

```
## Not run:
pb_new_release("cboettig/piggyback-tests", "v0.0.5")

## End(Not run)
```

pb_track *Track data files of a given pattern or location*

Description

Track data files of a given pattern or location

Usage

```
pb_track(glob = NULL, repo_root = usethis::proj_get())
```

Arguments

glob	vector of file names and/or glob pattern (e.g. *.csv, data/*.csv) which will be tracked by piggyback. Omit (default NULL) to just return a list of files currently tracked.
repo_root	repository root, will be guessed by usethis otherwise.

Details

Note: tracked patterns are simply written to .pbattributes (analogous to .gitattributes in git-lfs.) You can also edit this file manually. You will probably want to check in .psattributes to as to version control., with git add .psattributes. Note that tracked file patterns will also be added to .gitignore.

Value

list of tracked files (invisibly)

Examples

```
## Not run:
## Track all .csv and .tsv files
pb_track(c("*.tsv", "*.tsv.gz"))


## End(Not run)
```

pb_upload *Upload data to an existing release*

Description

NOTE: you must first create a release if one does not already exists.

Usage

```
pb_upload(file, repo = guess_repo(), tag = "latest", name = NULL,
  overwrite = TRUE, use_timestamps = TRUE, show_progress = TRUE,
  .token = get_token(), dir = ".")
```

Arguments

file	path to file to be uploaded
repo	Repository name in format "owner/repo". Will guess the current repo if not specified.
tag	tag for the GitHub release to which this data should be attached.
name	name for uploaded file. If not provided will use the basename of `file` (i.e. filename without directory)
overwrite	overwrite any existing file with the same name already attached to the on release? Defaults to TRUE
use_timestamps	logical, if TRUE, then files will only be downloaded if timestamp on GitHub is newer than the local timestamp (if `overwrite=TRUE`). Defaults to TRUE.
show_progress	logical, show a progress bar be shown for uploading? Defaults to TRUE.
.token	GitHub authentication token. Typically set from an environmental variable, e.g. in a `.Renviron` file or with `Sys.setenv(GITHUB_TOKEN = "xxxxx")`, which helps prevent accidental disclosure of a secret token when sharing scripts.
dir	directory relative to which file names should be based.

Examples

```
## Not run:
# Needs your real token to run

readr::write_tsv(mtcars,"mtcars.tsv.xz")
pb_upload("mtcars.tsv.xz", "cboettig/piggyback-tests")

## End(Not run)
```

Index