Package: piggyback (via r-universe)

October 28, 2024

Version 0.1.5.9005

Title Managing Larger Data on a GitHub Repository

Description Helps store files as GitHub release assets, which is a convenient way for large/binary data files to piggyback onto public and private GitHub repositories. Includes functions for file downloads, uploads, and managing releases via the GitHub API.

URL https://docs.ropensci.org/piggyback/, https://github.com/ropensci/piggyback

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

License GPL-3

Encoding UTF-8

ByteCompile true

Imports cli, glue, gh, httr, jsonlite, fs, memoise, rlang

Suggests arrow, spelling, readr, covr, testthat, knitr, rmarkdown, gert, withr, magrittr

VignetteBuilder knitr

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Language en-US

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

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

RemoteRef master

RemoteSha da844846355fa40515e009f9875e00ad2e8bf2f1

Contents

`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.

Author(s)

Maintainer: Carl Boettiger `<cboettig@gmail.com>` (ORCID) [copyright holder]

Authors:

- Tan Ho (ORCID)

Other contributors:

- Mark Padgham (ORCID) [contributor]
- Jeffrey O Hanson (ORCID) [contributor]
- Kevin Kuo (ORCID) [contributor]

See Also

Useful links:

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

---

`.gh_api_url` *GitHub API URL*

Description

Reads environment variable GITHUB_API_URL to determine base URL of API. Same as gh package. Defaults to `https://api.github.com`.

Usage

```
.gh_api_url()
```

Value

string: API base url

See Also

https://gh.r-lib.org/#environment-variables

`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 = gh::gh_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	string: GH repository name in format "owner/repo". Default guess_repo() tries to guess based on current working directory's git repository
tag	string: tag for the GH release, defaults to "latest"
.token	GitHub authentication token, see gh::gh_token()

Value

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

Examples

```
## Not run:
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")

## End(Not run)
```

---

pb_download *Download data from an existing release*

Description

Download data from an existing release

Usage

```
pb_download(
  file = NULL,
  dest = ".",
  repo = guess_repo(),
  tag = "latest",
  overwrite = TRUE,
  ignore = "manifest.json",
  use_timestamps = TRUE,
  show_progress = getOption("piggyback.verbose", default = interactive()),
  .token = gh::gh_token()
)
```

Arguments

file	character: vector of names of files to be downloaded. If NULL, all assets attached to the release will be downloaded.
dest	character: path to destination directory (if length one) or vector of destination filepaths the same length as file. Any directories in the path provided must already exist.
repo	string: GH repository name in format "owner/repo". Default guess_repo() tries to guess based on current working directory's git repository
tag	string: tag for the GH release, defaults to "latest"
overwrite	boolean: should any local files of the same name be overwritten? default TRUE
ignore	character: vector of files to ignore (used if downloading "all" via file=NULL)
use_timestamps	DEPRECATED.
show_progress	logical, show a progress bar be shown for uploading? Defaults to interactive() - can also set globally with options("piggyback.verbose")
.token	GitHub authentication token, see gh::gh_token()

Examples

```
## Download a specific file.
## (if dest is omitted, will write to current directory)
dest <- tempdir()
piggyback::pb_download(
  "iris.tsv.gz",
  repo = "cboettig/piggyback-tests",
  tag = "v0.0.1",
  dest = dest
)
list.files(dest)
## Download all files
piggyback::pb_download(
  repo = "cboettig/piggyback-tests",
  tag = "v0.0.1",
  dest = dest
)
list.files(dest)
```

`pb_download_url` *Get the download url of a given file*

Description

Returns the URL download for a given file. This can be useful when using functions that are able to accept URLs.

Usage

```
pb_download_url(
  file = NULL,
  repo = guess_repo(),
  tag = "latest",
  url_type = c("browser", "api"),
  .token = gh::gh_token()
)
```

Arguments

`file`	character: vector of names of files to be downloaded. If `NULL`, all assets attached to the release will be downloaded.
`repo`	string: GH repository name in format "owner/repo". Default `guess_repo()` tries to guess based on current working directory's git repository
`tag`	string: tag for the GH release, defaults to "latest"
`url_type`	choice: one of "browser" or "api" - default "browser" is a web-facing URL that is not subject to API ratelimits but does not work for private repositories. "api" URLs work for private repos, but require a GitHub token passed in an Authorization header (see examples)
`.token`	GitHub authentication token, see `gh::gh_token()`

Value

the URL to download a file

Examples

```
# returns browser url by default (and all files if none are specified)
browser_url <- pb_download_url(
  repo = "tanho63/piggyback-tests",
  tag = "v0.0.2"
  )
print(browser_url)
utils::read.csv(browser_url[[1]])
```

```
# can return api url if desired
api_url <- pb_download_url(
  "mtcars.csv",
  repo = "tanho63/piggyback-tests",
  tag = "v0.0.2"
  )
print(api_url)

# for public repositories, this will still work
utils::read.csv(api_url)

# for private repos, can use httr or curl to fetch and then pass into read function
gh_pat <- Sys.getenv("GITHUB_PAT")

if(!identical(gh_pat, "")){
  resp <- httr::GET(api_url, httr::add_headers(Authorization = paste("Bearer", gh_pat)))
  utils::read.csv(text = httr::content(resp, as = "text"))
}

# or use pb_read which bundles some of this for you
```

`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, .token = gh::gh_token())
```

Arguments

`repo`	string: GH repository name in format "owner/repo". Default `guess_repo()` tries to guess based on current working directory's git repository
`tag`	which release tag(s) do we want information for? If `NULL` (default), will return a table for all available release tags.
`.token`	GitHub authentication token, see `gh::gh_token()`

Value

a data.frame of release asset names, release tag, timestamp, owner, and repo.

See Also

`pb_releases` for a list of all releases in repository

Examples

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

## End(Not run)
```

---

pb_read *Read one file into memory*

Description

A convenience wrapper around writing an object to a temporary file and then uploading to a specified repo/release. This convenience comes at a cost to performance efficiency, since it first downloads the data to disk and then reads the data from disk into memory. See `vignette("cloud_native")` for alternative ways to bypass this flow and work with the data directly.

Usage

```
pb_read(
  file,
  ...,
  repo = guess_repo(),
  tag = "latest",
  read_function = guess_read_function(file),
  .token = gh::gh_token()
)
```

Arguments

`file`	string: file name
`...`	additional arguments passed to `read_function` after file
`repo`	string: GH repository name in format "owner/repo". Default `guess_repo()` tries to guess based on current working directory's git repo
`tag`	string: tag for the GH release, defaults to "latest"
`read_function`	function: used to read in the data, where the file is passed as the first argument and any additional arguments are subsequently passed in via `....` Default `guess_read_function(file)` will check the file extension and try to find an appropriate read function if the extension is one of rds, csv, tsv, parquet, txt, or json, and will abort if not found.
`.token`	GitHub authentication token, see `gh::gh_token()`

Value

Result of reading in the file in question.

See Also

Other pb_rw: guess_read_function(), guess_write_function(), pb_write()

Examples

```
try({ # try block is to avoid CRAN issues and is not required in ordinary usage
 piggyback::pb_read("mtcars.tsv.gz", repo = "cboettig/piggyback-tests")
})
```

---

pb_releases *List releases in repository*

Description

This function retrieves information about all releases attached to a given repository.

Usage

```
pb_releases(
  repo = guess_repo(),
  .token = gh::gh_token(),
  verbose = getOption("piggyback.verbose", default = TRUE)
)
```

Arguments

repo	GitHub repository specification in the form of "owner/repo", if not specified will try to guess repo based on current working directory.
.token	a GitHub API token, defaults to gh::gh_token()
verbose	defaults to TRUE, use FALSE to silence messages

Value

a dataframe of all releases available within a repository.

Examples

```
try({ # wrapped in try block to prevent CRAN errors
 pb_releases("nflverse/nflverse-data")
})
```

`pb_release_create` *Create a new release on GitHub repo*

Description

Create a new release on GitHub repo

Usage

```
pb_release_create(
  repo = guess_repo(),
  tag,
  commit = NULL,
  name = tag,
  body = "Data release",
  draft = FALSE,
  prerelease = FALSE,
  .token = gh::gh_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 full commit SHA (not the short hash). 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, see `[gh::gh_token()]`

See Also

Other release_management: `pb_release_delete()`

Examples

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

## End(Not run)
```

pb_release_delete *Delete release from GitHub repo*

Description

Delete release from GitHub repo

Usage

```
pb_release_delete(repo = guess_repo(), tag, .token = gh::gh_token())
```

Arguments

repo	Repository name in format "owner/repo". Defaults to `guess_repo()`.
tag	tag name to delete. Must be one of those found in `pb_releases()$tag_name`.
.token	GitHub authentication token, see `[gh::gh_token()]`

See Also

Other release_management: `pb_release_create()`

Examples

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

## 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 = "use_timestamps",
  use_timestamps = NULL,
  show_progress = getOption("piggyback.verbose", default = interactive()),
  .token = gh::gh_token(),
  dir = NULL
)
```

Arguments

`file`	string: path to file to be uploaded
`repo`	string: GH repository name in format "owner/repo". Default `guess_repo()` tries to guess based on current working directory's git repository
`tag`	string: tag for the GH release, defaults to "latest"
`name`	string: name for uploaded file. If not provided will use the basename of `file` (i.e. filename without directory)
`overwrite`	choice: overwrite any existing file with the same name already attached to the on release? Options are "use_timestamps", TRUE, or FALSE: default "use_timestamps" will only overwrite files where the release timestamp is newer than the local file.
`use_timestamps`	DEPRECATED.
`show_progress`	logical, show a progress bar be shown for uploading? Defaults to `interactive()` - can also set globally with options("piggyback.verbose")
`.token`	GitHub authentication token, see `gh::gh_token()`
`dir`	directory relative to which file names should be based, defaults to NULL for current working directory.

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)
```

---

pb_write *Write one object to repo/release*

Description

A convenience wrapper around writing an object to a temporary file and then uploading to a specified repo/release.

Usage

```
pb_write(
  x,
  file,
  ...,
  repo = guess_repo(),
  tag = "latest",
  write_function = guess_write_function(file),
  .token = gh::gh_token()
)
```

Arguments

`x`	object: memory object to save to piggyback
`file`	string: file name
`...`	additional arguments passed to `write_function`
`repo`	string: GH repository name in format "owner/repo". Default `guess_repo()` tries to guess based on current working directory's git repo
`tag`	string: tag for the GH release, defaults to "latest"
`write_function`	function: used to write an R object to file, where the object is passed as the first argument, the filename as the second argument, and any additional arguments are subsequently passed in via `...`. Default `guess_write_function(file)` will check the file extension and try to find an appropriate write function if the extension is one of rds, csv, tsv, parquet, txt, or json, and will abort if not found.
`.token`	GitHub authentication token, see `gh::gh_token()`

Value

Writes file to release and returns github API response

See Also

Other pb_rw: `guess_read_function()`, `guess_write_function()`, `pb_read()`

Examples

```
pb_write(mtcars, "mtcars.rds", repo = "tanho63/piggyback-tests")
#>  Uploading to latest release: "v0.0.2".
#>  Uploading mtcars.rds ...
#> |============================================================| 100%
```

Index