Managing Larger Data on a GitHub Repository

Carl Boettiger[1]

1 University of California, Berkeley

DOI: 10.21105/joss.00971

Software

- Review
- Repository
- Archive

Submitted: 21 September 2018
Published: 24 September 2018

Piggyback: Working with larger data in GitHub

GitHub has become a central component for preserving and sharing software-driven analysis in academic research (Ram, 2013). As scientists adopt this workflow, a desire to manage data associated with the analysis in the same manner soon emerges. While small data can easily be committed to GitHub repositories along-side source code and analysis scripts, files larger than 50 MB cannot. Existing work-arounds introduce significant complexity and break the ease of sharing (Boettiger, 2018a).

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. `piggyback` provides a workflow similar to Git LFS ("Git LFS," 2018), in which data files can be tracked by type and pushed and pulled to GitHub with dedicated commands. These files are not handled by git in any way, but instead are uploaded, downloaded, or edited directly by calls through the GitHub API ("GitHub API version 3," 2018). 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.

Examples

As long as a repository has at least one release, users can upload a set of specified files from the current repository to that release by simply passing the file names to `pb_upload()`. Specify individual files to download using `pb_download()`, or use no arguments to download all data files attached to the latest release. Alternatively, users can track files by a given pattern: for instance, `pb_track("*.csv")` will track all `*.csv` files in the repository. Then use `pb_upload(pb_track())` to upload all currently tracked files. `piggyback` compares timestamps to avoid unnecessary transfer. The `piggyback` package looks for the same `GITHUB_TOKEN` environmental variable for authentication that is used across GitHub APIs. Details are provided in an introductory vignette (Boettiger, 2018b).

References

Boettiger, C. (2018a). Piggyback comparison to alternatives. Retrieved from https://ropensci.github.io/piggyback/articles/alternatives.html

Boettiger, C. (2018b). Piggyback Data atop your GitHub Repository! Retrieved from https://ropensci.github.io/piggyback/articles/intro.html

Git LFS. (2018). https://git-lfs.github.com/. Retrieved from https://git-lfs.github.com/

GitHub API version 3. (2018). https://developer.github.com/v3/. Retrieved from https://developer.github.com/v3/

Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine*, *8*(1), 7. doi:10.1186/1751-0473-8-7