

Git and GitHub: Collaborative Usage

Dominic Bordelon, University Library System

Agenda

1. GitHub Access Permissions
2. Forks: copies of a repository
3. Pull requests (PRs)
4. Merge conflicts: when branches collide
5. The blame feature
6. Repository management paradigms

About the trainer

Dominic Bordelon, Research Data Librarian

University Library System, University of Pittsburgh

dbordelon@pitt.edu

Services for the Pitt community:

- Consultations
- Training (on-request and via public workshops)
- Talks (on-request and publicly)
- Research collaboration

Support areas and interests:

- Computer programming fundamentals, esp. for data processing and analysis
- Open Science and Data Sharing
- Data stewardship/curation
- Research methods; science and technology studies

GitHub Access Permissions

Anyone can clone or pull from a *public* GH repo. (A repo may also be set to *private*.)

Only authorized Collaborators may *push* to a repo. To add Collaborators, click on your repo's Settings button (top right), then Collaborators from the left menu.

💡 If your repo has multiple Contributors, consider having everyone work in their own branch, to minimize the number of merges needed as users push and pull to the **origin** repo.

Forks: copies of a repository

Forks

A **fork** is a copy of the repository under a new owner (usually retains original name). The fork owner can make whatever modifications they desire. So if I want to make a derivative project of tidyverse/dplyr, I can fork it to create dojobo/dplyr (dojobo is my username).

Parent repository updates can be merged into the fork at any time. So using the dplyr example, I can merge new tidyverse/dplyr commits into dojobo/dplyr, keeping up with the new developments.

To see what forks exist: Insights > Forks. (See also: Network)

To create your own fork: click Fork on the main page.

tidyverse / dplyr

Type / to search

>

+

<> CodeIssues 53Pull requests 8ActionsSecurityInsights

dplyr

Public

Watch 245

Fork 2.1k

Star 4.6k

main 36 branches57 tags

Go to fileAdd fileCode

DavisVaughan

Increment version number to 1.1.3.9000

7374271 last month7,738 commits

.github	2023 upkeep (#6778)	7 months ago
R	Remove unneeded object docs	2 months ago
archive	run-in and run-all	5 years ago
data-raw	trivial change to dataset documentation (#6858)	2 months ago
data	Update the storms dataset (#6320)	last year
inst	master -> main (#6065)	2 years ago
man	Remove unneeded object docs	2 months ago
pkgdown/favicon	Update dplyr logo (#5248)	3 years ago

About

dplyr: A grammar of data manipulation

dplyr.tidyverse.org/

r

grammar

data-manipulation

Readme

Unknown, MIT licenses found

Code of conduct

Activity

4.6k stars


245 watching

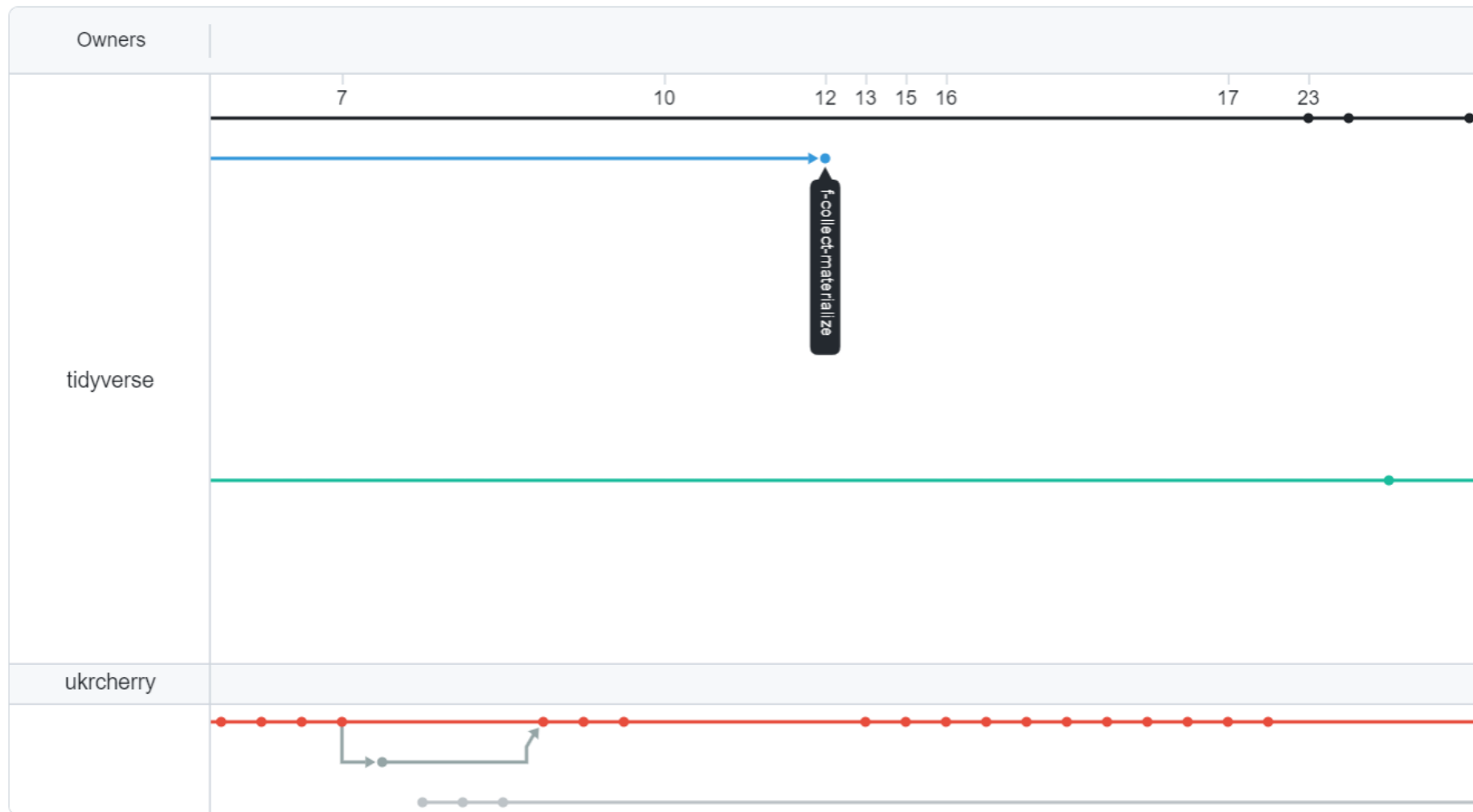
2.1k forks

Report repository

The Fork button, to make your own fork of a posted repo. Insights is also highlighted, where you may find the list of Forks and the Network graph (see next slide).

Git and GitHub: Collaborative Usage

 University of Pittsburgh | Library System



Part of the Network Graph for dplyr. Each row is a fork owner, each colored line is a branch, and each dot/node is a commit. The owner at bottom of image shows a one-commit branch and merge in dark gray, and a new branch (initial commit) in light gray. The black label near center of image shows the branch's name. The numbers at top are dates of a month.

Activity: Fork our repo

1. Go to our repo for this workshop (I will provide the URL).
2. Click on Fork at top right to fork it into your account. Read the default settings and leave them as-is. Click the Create fork button.
3. Examine the new repository you've created, and confirm that it is a fork.
4. Create a new branch and commit a new file (you may do this on the GH website, or on a local clone, which you then push).
5. Refresh your fork on GH and confirm that your commit is there.

Pull Requests (PRs): collaborative merging

Pull requests (PRs)

Once a collaborator has developed their branch sufficiently and wants it merged to `main`, they make a request to repo administrators. This is called a **pull request (PR)** because you are asking to have your branch's commits *pulled* into `main`. (Not consistent with the `git pull` command!)

If the admin says Yes to the PR, the commits are merged into the target repo.

Depending on the nature of the commits, the repository's contribution policies, and your relationship to the repo admins, you may need to make a case for your PR, or may even be outright rejected.

To make a PR, click Pull Requests (top middle) and click New pull request.

💡 A PR can occur across separate forks, or across branches within the same repository.

Activity: Examine pull requests

1. On the tidyverse/dplyr repo, click on Pull requests.
2. Review some of the open PRs.
3. Click on the “Closed” button to see a list of Closed PRs, and review a few of those. Examine at least one accepted PR and one rejected PR.

Activity: Create a Pull Request

You will ask to have the commit you created in a previous activity, merged into our repo for this workshop.

1. Go to the repo for this workshop.
2. Initiate a PR by clicking Pull Request and then New pull request.
3. On the Compare Changes screen, select the appropriate branches for merging. The workshop repo and branch should be on the left, and your fork and repo on the right.
4. Examine the diffs and create a PR.

Then I will accept the PRs.

Merge conflicts: when branches collide

Merge conflicts

When you merge Branch B into Branch A, git will attempt to auto-merge, i.e., automatically reconcile the two branches. An example of a straightforward auto-merge is a new file created in Branch A, and a modified existing file in Branch B: both changes take effect.

If the merge can't be resolved automatically, then merging halts:

```
$ git merge branch_b
Auto-merging file1
CONFLICT (content): Merge conflict in file1
Automatic merge failed; fix conflicts and then commit the result.
```

and a *merge conflict* is generated at each appropriate position in the files.

Here's an example of what a merge conflict looks like:

```
<<<<<<< HEAD
content from branch A
=====
content from branch b
>>>>>>> branch_b
```

The ===== is a divider between the contributions from each branch. (Note: **HEAD** is git's term for your current position, or which commit you currently have checked out. In the merge situation, it corresponds to the most recent commit of Branch A.) You'll have to manually reconcile the content in the "fenced" area, then delete the divider and fences.

Once all merge conflicts are resolved, you can stage and commit the changes, to complete the merge.

The blame feature

The blame feature

The **blame** feature, which is not really about “blaming,” tells you, for a given file, which user committed (wrote) each line. This can be useful when reconstructing history, bug-hunting, or re-assessing current design choices.

Similar to `log`, `blame` is available on the command-line, but probably best experienced in a richer platform like GitHub.

romainfrancois Update gitHub actions (#5188) ...

b3d85c1 · 3 years ago History

Code

Blame

14 lines (13 loc) · 232 Bytes

Raw

Older  Newer

Contributors

3

7 years ago



Supress codecov comments

1

`comment: false`

6 years ago



Make codecov quiet (#2535)



2

`coverage:`

3

`status:`

4

`project:`

5

`default:`

6

`target: auto`

3 years ago



Update gitHub actions (#5188)



7

`threshold: 1%`

8

`informational: true`

9

`patch:`

10

`default:`

11

`target: auto`

12

`threshold: 1%`

13

`informational: true`

14

A file in the tidyverse/dplyr repo, codecov.yml. The view has been switched from Code (the default) to Blame. Note that each line in the file is assigned an author and commit.

Repository management paradigms

Repository management paradigms

- Branching workflows: <https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>
- Distributed workflows: <https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows>

Resources for learning and practice

- Pro Git book, available online: <https://git-scm.com/book/en/v2>
- Git Gud, an in-browser git simulator: <https://nic-hartley.github.io/git-gud/>
- Learning resources collected by GitHub: <https://docs.github.com/en/get-started/quickstart/git-and-github-learning-resources>
- Cheat sheet for commands usable in Git Bash and macOS Terminal: <https://github.com/RehanSaeed/Bash-Cheat-Sheet>

Or you can look for git and GitHub integrations in software you already use, for example: [git in VS Code](#), [git in RStudio](#), [git in PyCharm](#)

Or you can delve into one of these topics:

- [GitHub Pages](#): easily build and host an associated page for your project (or create an empty repo solely to host a webpage)
- [GitHub Actions](#): automate workflows, such as running a suite of tests on your code every time a new commit is pushed, to mitigate introduction of new bugs.

