Git and GitHub: Solo Usage

Dominic Bordelon, University Library System

Agenda

1. Navigating GitHub and cloning repositories
2. Repository history: commits and branches
3. Using git locally: initializing, staging, committing
4. Posting your git repo to GitHub
5. Resources for learning and practice

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

Git versus GitHub

- Open-source software (owned by no one)
- **Version control software**
- Command-line application (although there are graphical clients)
- Comes installed on Linux and macOS; installed on Windows via https://gitforwindows.org/
- Est. 2005
- https://git-scm.com/

- Owned by MicroSoft
- **Website, server for Git repositories**, and company
- Feature-rich: social features; Actions for automated workflows; hosting of pages
- Est. 2008
- https://github.com/

Navigating GitHub and cloning repositories

We will use the public repository for the dplyr R package as an example.

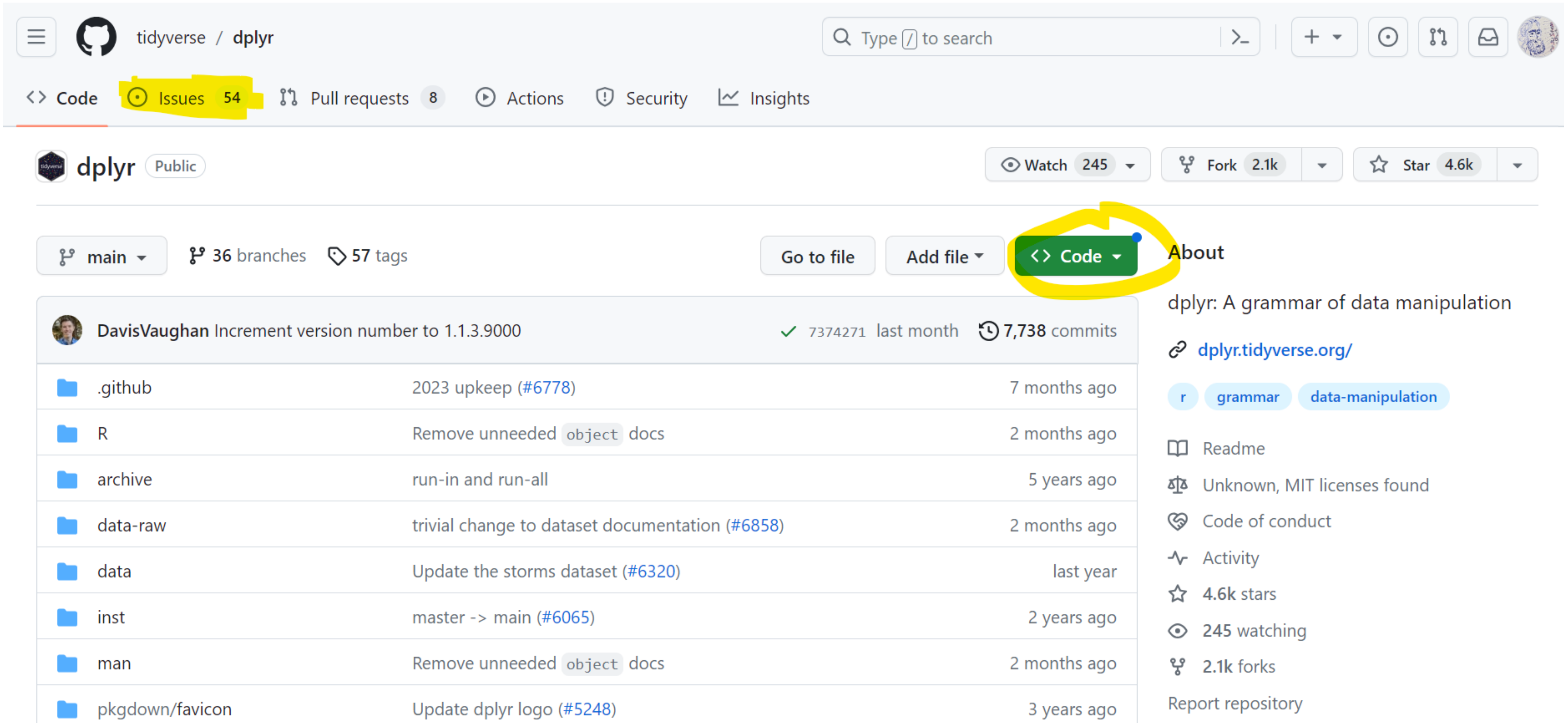
https://github.com/tidyverse/dplyr

Note:

- Username / repository name format (`tidyverse/dplyr`)
- File listing, README: contents of the main page
- Issues: tab where bugs, feature requests, etc. may be reported

Using the green Code button, you can download a zip of the repository, or **clone** it to your computer using the `git clone` command.

Whenever the source (or “`origin`”) repository updates on github.com, downstream clones may afterwards fetch and integrate updates.


tidyverse / dplyr

Code | Issues 54 | Pull requests 8 | Actions | Security | Insights

dplyr Public

Watch 245 | Fork 2.1k | Star 4.6k

main | 36 branches | 57 tags | Go to file | Add file | Code

DavisVaughan Increment version number to 1.1.3.9000 ✓ 7374271 last month 7,738 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

Note Issues tab, Code button, and file listing of the `tidyverse/dplyr` repository.

Repository history: commits and branches

Commits and branches

The repository's changes over time are documented as snapshots or **commits**.

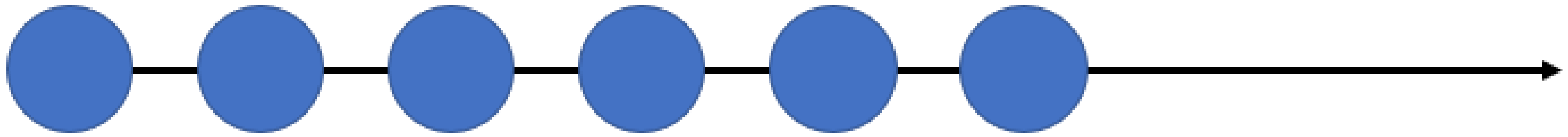
- A commit is a *changeset* or description of each repository change occurring since the previous commit.
- Each commit has a *hash* or tag that looks like: `ea6fb94` (for example)
- Each commit also typically has a *message* that briefly describes what change(s) the commit contains.

A series of commits in sequence is a **branch**. The user creates a branch to begin a new "line of development," choosing an existing commit as the starting point of the branch.

A **merge** occurs when branch B pulls in a commit from branch A and combines them into a new commit on branch B.

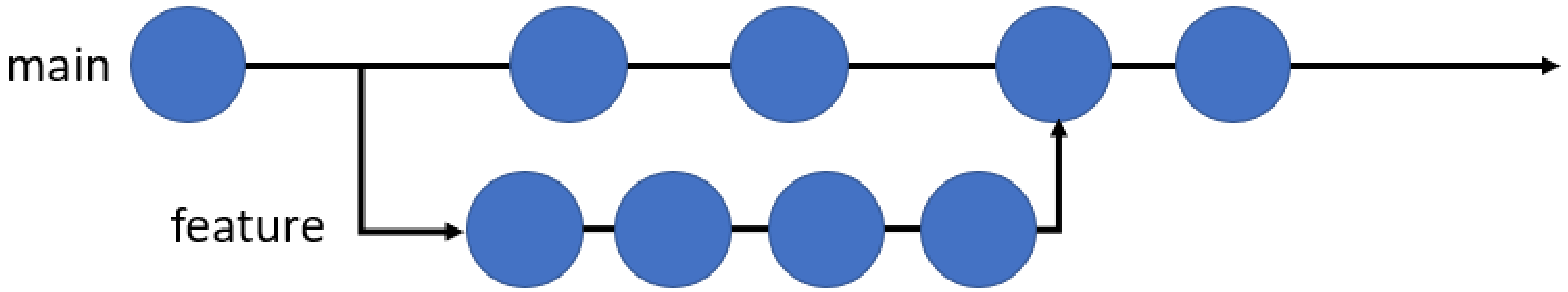
💡 One's current location in the graph is called HEAD. Any commit (node) of the graph may be *checked out* to view the repository state at that moment. HEAD moves to the checked-out commit.

Version control: simplest scenario



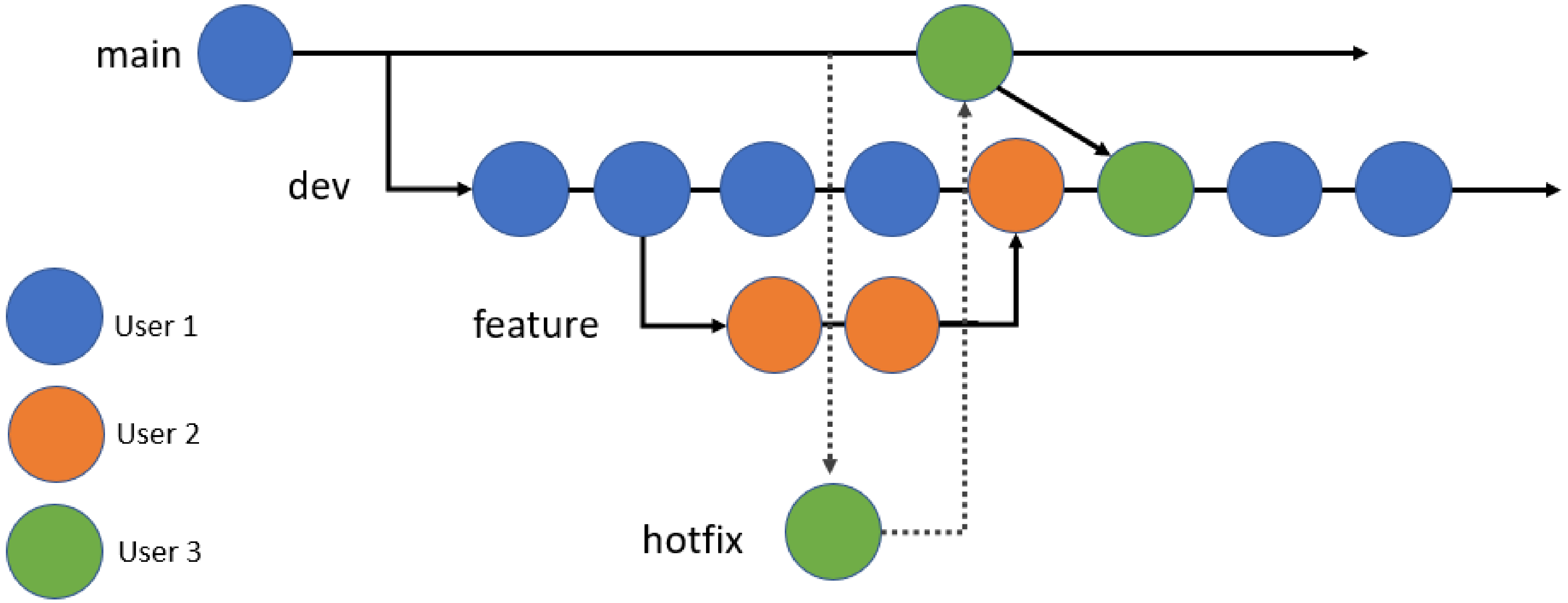
💡 Aim for each commit to be "atomic," i.e., accomplishing one unit of work or task.

Version control: branching



💡 Aim for each branch to segment out a bounded part of the project. Branches may be of any length.

Version control: branching, multi-user



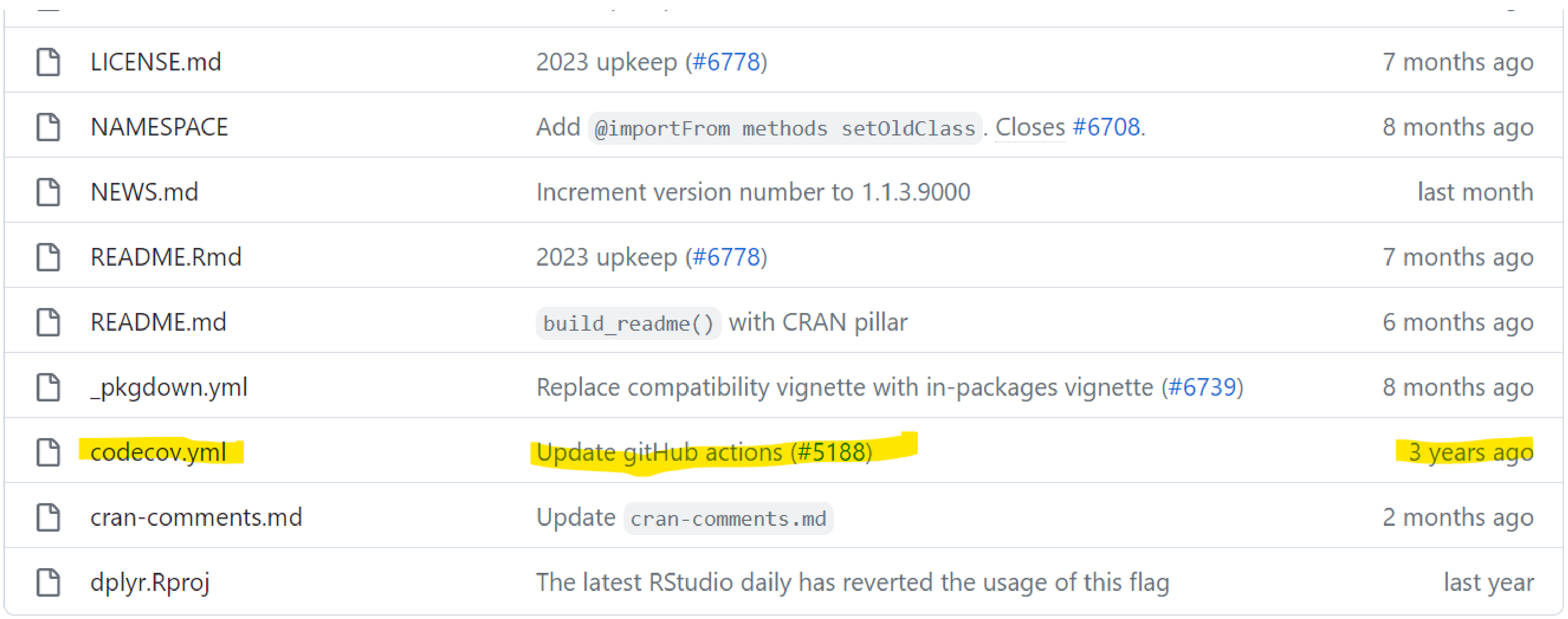
Commit and Branch info on GitHub

Commits are prominent on GH: they appear on file listings, to show the message+age of the last edit (see next slide). The message (as well as any hash) links to the commit.

Branches are accessed at the top left. Switching to a branch changes the file view.

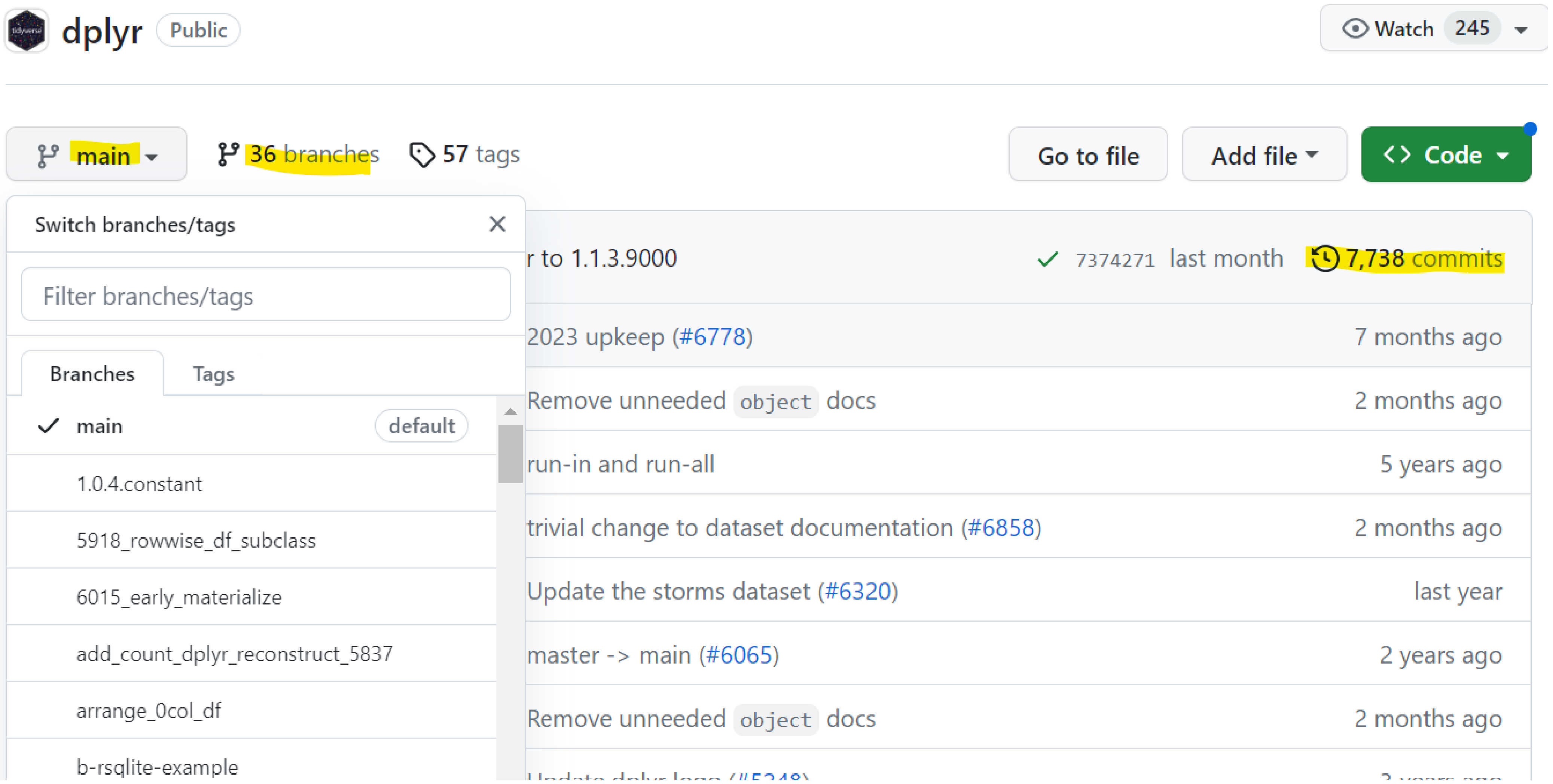
For dplyr repository, note:

- Commits
- Branches
- Pull Requests (how branches get merged on GitHub)
- Insights > Network (graph visualization)



LICENSE.md	2023 upkeep (#6778)	7 months ago
NAMESPACE	Add `@importFrom methods setOldClass`. Closes #6708.	8 months ago
NEWS.md	Increment version number to 1.1.3.9000	last month
README.Rmd	2023 upkeep (#6778)	7 months ago
README.md	`build_readme()` with CRAN pillar	6 months ago
_pkgdown.yml	Replace compatibility vignette with in-packages vignette (#6739)	8 months ago
codecov.yml	Update gitHub actions (#5188)	3 years ago
cran-comments.md	Update `cran-comments.md`	2 months ago
dplyr.Rproj	The latest RStudio daily has reverted the usage of this flag	last year

Example file listing. The highlighted file, `codecov.yml`, was last updated 3 years ago in a commit whose message was "Update gitHub actions (#5188)". Clicking the message shows the page for the commit (full description + files changed + file diffs). #5188 refers and links to an Issue.



dplyr Public

Watch 245

main

36 branches

57 tags

Go to file

Add file

Code

Switch branches/tags

Filter branches/tags

Branches

Tags

- main default
- 1.0.4.constant
- 5918_rowwise_df_subclass
- 6015_early_materialize
- add_count_dplyr_reconstruct_5837
- arrange_0col_df
- b-rsqlite-example

r to 1.1.3.9000 ✓ 7374271 last month 7,738 commits

2023 upkeep (#6778)	7 months ago
Remove unneeded `object` docs	2 months ago
run-in and run-all	5 years ago
trivial change to dataset documentation (#6858)	2 months ago
Update the storms dataset (#6320)	last year
master -> main (#6065)	2 years ago
Remove unneeded `object` docs	2 months ago

Branch button expanded. Click on a branch name to switch to it. Click on commits, right, to see a log of the current branch's commits.

Activity: Find and clone (download) a repo

1. Search on GitHub for a term that interests you (ex. “genomics”)
2. Select a repo that looks potentially interesting. Skim its README and assess information such as project age, number of contributors and commits, and most recent commit.
3. Download or clone the repo to your computer. Have a look at some of the files.

Remotes

The local clone of the repo on your computer is configured to “remember” the GH instance. The GH instance is configured as a **remote**. In particular, it is a special remote called **`origin`**.

You may run `git pull` anytime, to fetch and merge new commits that have been posted to `origin` since you last checked. (`git fetch` downloads commits without merging them automatically, for inspection prior to merging.)

There won’t be anything to pull yet, since you’ve only just cloned the repo, but pulling becomes very important when:

- tracking a project for a long time (you want to keep your local copy up-to-date), and/or
- working collaboratively, wherein others are committing to `origin` concurrently with you.

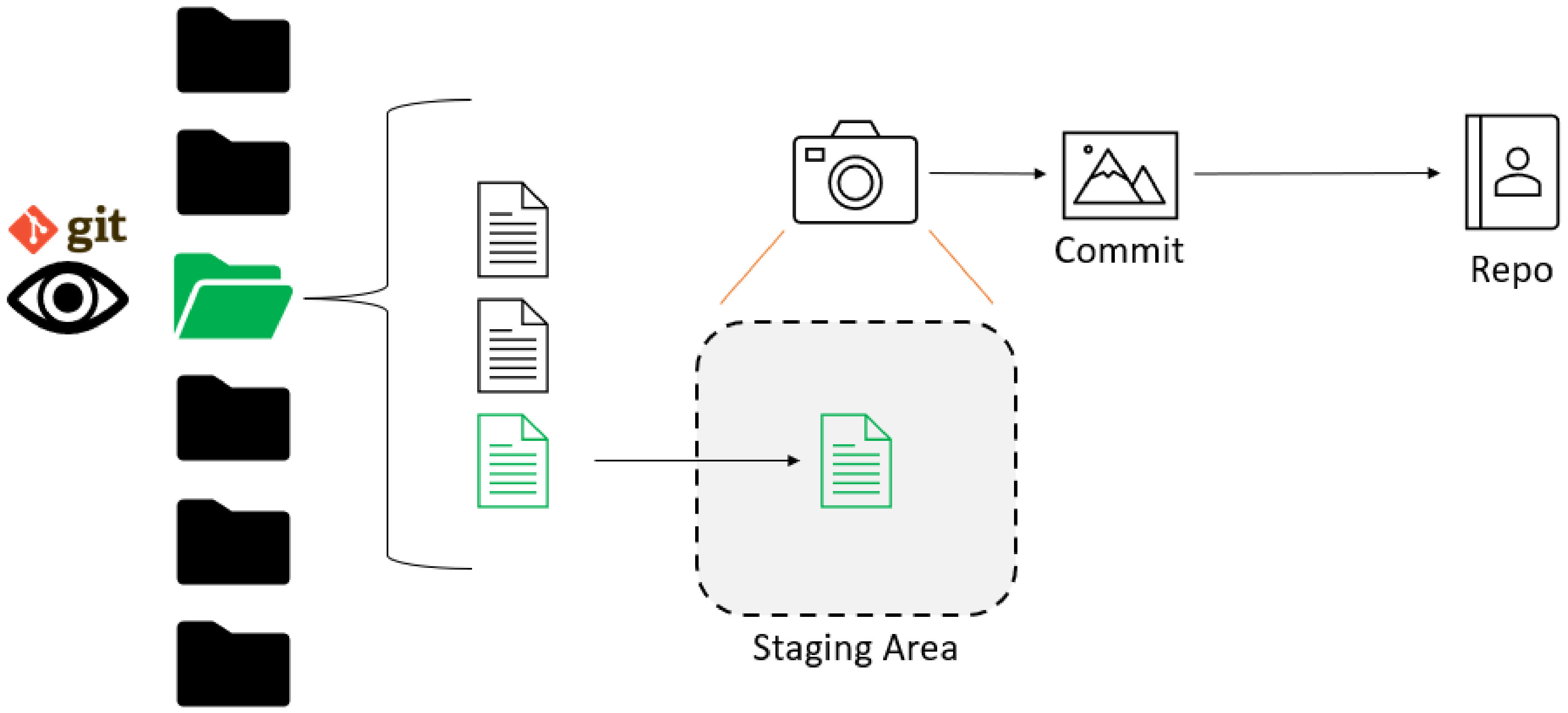
Using git locally: initializing, staging, committing

Activity: Practice committing, branching, and merging

We will practice committing, branching, and merging.

1. Go to Git Gud at https://nic-hartley.github.io/git-gud/. This is a browser-based git emulator.
2. `HEAD` is at `c0`, the initial commit on branch `b0`. Use the `git commit` command to make some commits.
3. Use the `git branch` command to create a new branch.
4. Use `git checkout b1` to checkout branch `b1`. Make some commits to the new branch.
5. Switch back to `b0` and then make a commit. Where does the commit appear?
6. Switch back to `b1`. Update `b1` by using `merge` on the `b0` commit that you just made.
7. Make another commit to `b1`. Then switch back to `b0` and merge `b1` back into `b0`.

How git works in your file system



git monitors the project directory for changes. Changes are *added* to the *staging area* by the user. Once all desired changes are staged for the current action or task, the user *commits* them to the repository.

Initializing, staging, committing

Important commands:

`git init`	Initialize a git repository in the current directory
`git status`	Check status of dir's git repo and staging area
`git add`	Add file(s) to staging area
`git commit -m "message"`	Commit staged files with a message
`git branch`	Create a branch
`git checkout 12e45f`	Check out commit #12e45f
`git checkout mybranch`	Check out latest commit on mybranch
`git merge mybranch`	Merge mybranch into the current branch

Activity: init, add, commit

1. Create a new folder, for a new project (let's pretend). Initialize a repo using `git init`.
2. Add one or more files to the directory (new in a text editor, or copied from another directory).
3. Run `git status` to see that you have 1+ untracked file(s).
4. Add your file(s) to the staging area using `git add`.
5. Run `git commit -m "my message"` to commit the staged files, with your desired message.
6. Run `git status` to confirm that the staging area is empty again.
7. Run `git log` to see the record of your commit.

Posting your git repo to GitHub

In GitHub, click “Create New Repository.” This will be a placeholder. Follow the directions on the next screen after creating, to **push** (upload) your repo into this placeholder.

After pushing, you should see your latest commit online. Woohoo!

If there are files you want to physically keep in the folder, yet exclude from git’s tracking—for example, a file with API credentials—list it in a plain text file named `.gitignore`.

- gitignore documentation: https://git-scm.com/docs/gitignore
- and/or start with a template for your favored language/platform from this repo: https://github.com/github/gitignore

Open Science practices

- In the README file, describe:
  - Purpose, scope, and authorship of the project
  - Contents of repository; file manifest
  - (for software/code) installation requirements, how to use
  - (for data) methods used to generate and process the data
- A LICENSE file can tell others your terms of reuse. GitHub also has a license template chooser.
- GitHub is good for short- and medium-term sharing. But long-term, persistent availability is only guaranteed by purpose-built research repositories like Zenodo (which has a great GitHub integration).
  - Zenodo will mint a DOI for you, which makes your code and data easily citable.

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.