Happy Git and Github for the useR Session 05 - Git Remote Setups + Daily Workflows

Boook club R-Ladies Bergen, R-Ladies Den Bosch, R-Ladies Amsterdam

Kylie Ainslie

Program for today

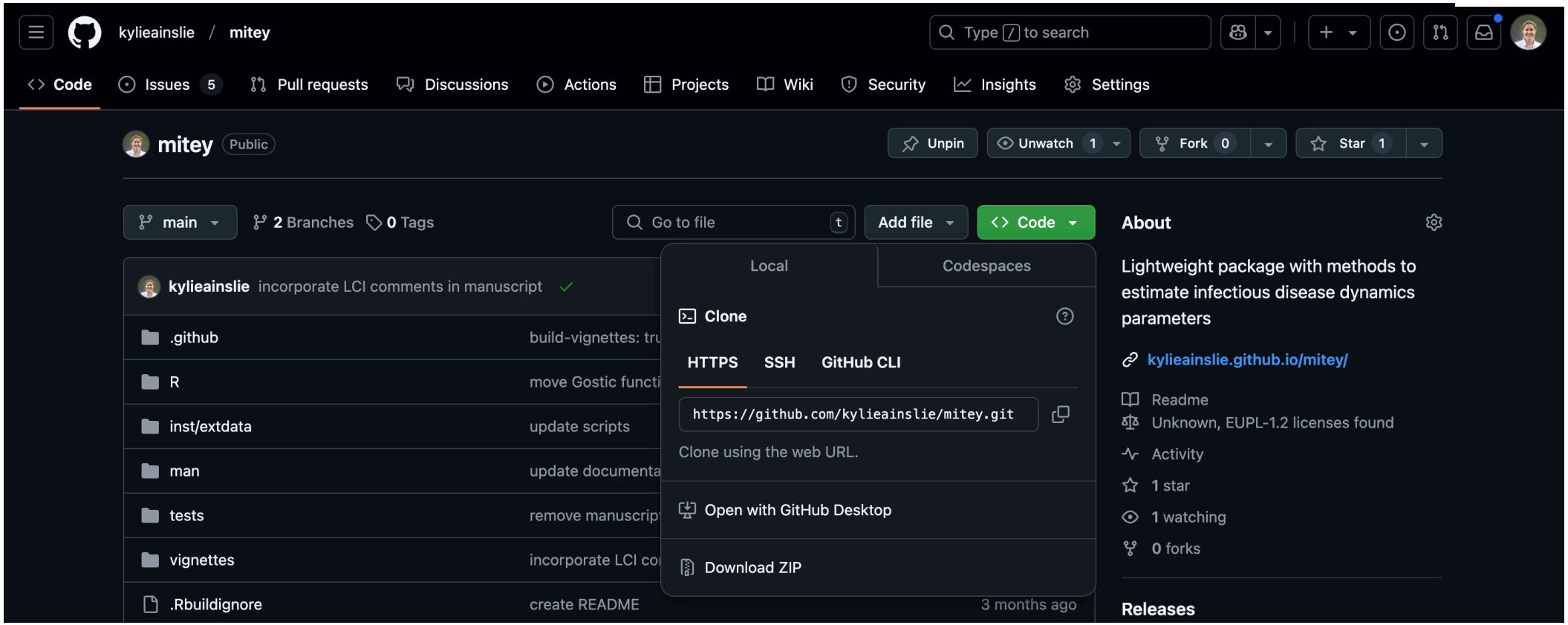
- Remote Setups
  - 25 Common remote setups
- Daily Workflows, Part 1
  - 27 The Repeated Amend
  - 28 Dealing with push rejection
  - 29 Pull, but you have local work

Chapter 25: Common remote setups

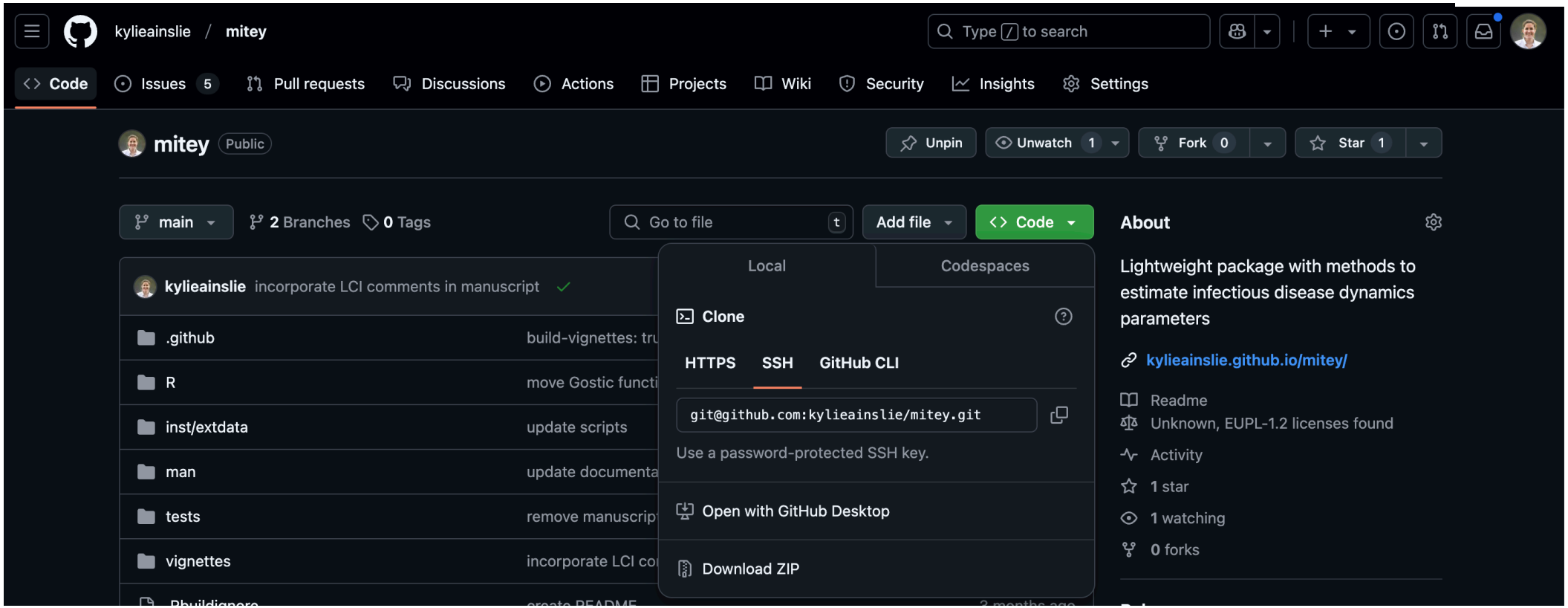
Recap

- A *remote* is repository hosted elsewhere than your local machine (e.g., on GitHub)
  - The remote is named `origin` or `upstream`.
  - On GitHub a remote URL looks like
    - `https://github.com/OWNER/REPO.git` or
    - `git@github.com:OWNER/REPO.git`

Recap


kylieainslie / mitey
Type / to search
Code
Issues 5
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
mitey
Public
Unpin
Unwatch 1
Fork 0
Star 1
main
2 Branches
0 Tags
Go to file
t
Add file
Code
kylieainslie incorporate LCI comments in manuscript
.github
build-vignettes: tru
R
move Gostic functi
inst/extdata
update scripts
man
update documenta
tests
remove manuscrip
vignettes
incorporate LCI co
.Rbuildignore
create README
3 months ago
Local
Codespaces
Clone
HTTPS
SSH
GitHub CLI
https://github.com/kylieainslie/mitey.git
Clone using the web URL.
Open with GitHub Desktop
Download ZIP
About
Lightweight package with methods to estimate infectious disease dynamics parameters
kylieainslie.github.io/mitey/
Readme
Unknown, EUPL-1.2 licenses found
Activity
1 star
1 watching
0 forks
Releases

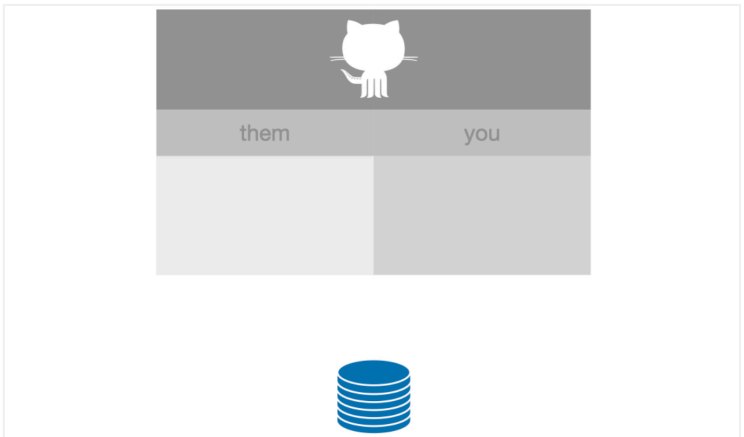
Recap


kylieainslie / mitey
Type / to search
Code
Issues 5
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
mitey
Public
Unpin
Unwatch 1
Fork 0
Star 1
main
2 Branches
0 Tags
Go to file
t
Add file
Code
kylieainslie incorporate LCI comments in manuscript
.github
build-vignettes: tru
R
move Gostic functi
inst/extdata
update scripts
man
update documenta
tests
remove manuscrip
vignettes
incorporate LCI co
Local
Codespaces
Clone
HTTPS
SSH
GitHub CLI
git@github.com:kylieainslie/mitey.git
Use a password-protected SSH key.
Open with GitHub Desktop
Download ZIP
About
Lightweight package with methods to estimate infectious disease dynamics parameters
kylieainslie.github.io/mitey/
Readme
Unknown, EUPL-1.2 licenses found
Activity
1 star
1 watching
0 forks

Recap

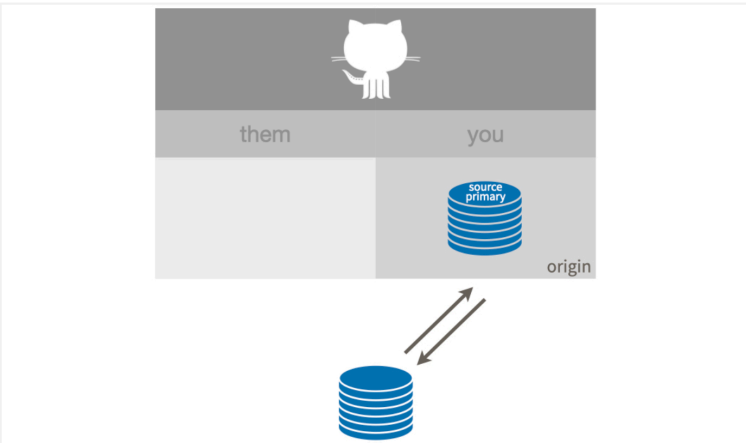
- A *fork* is a copy of a repository on a remote server (e.g., GitHub)
- A *clone* is a copy of a repository on your local machine

No GitHub



- To create a local Git repo:
  - `git init`
  - `usethis::use_git()`
  - In RStudio:
    - Existing Project: `Tools > Version Control > Project Setup`
    - New Project: Make sure “Create a Git repository” is selected

Yours

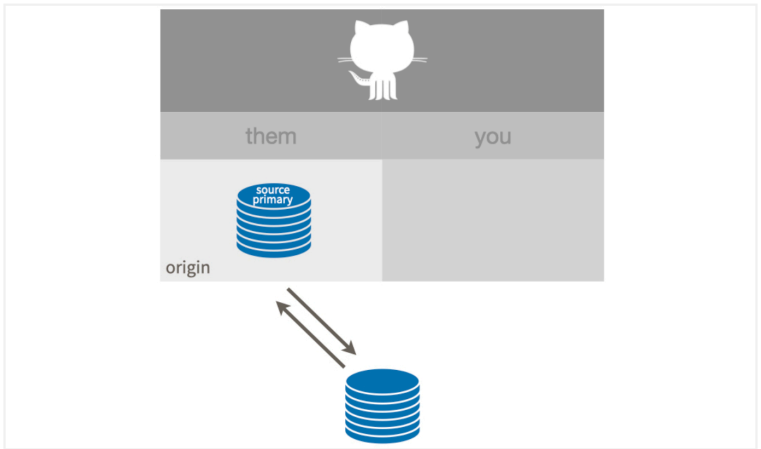


- A remote named `origin` is configured with permission to push/pull.
- `origin` on GitHub is a **source** repo.
- `origin` is your **primary** repo.

Yours - setup

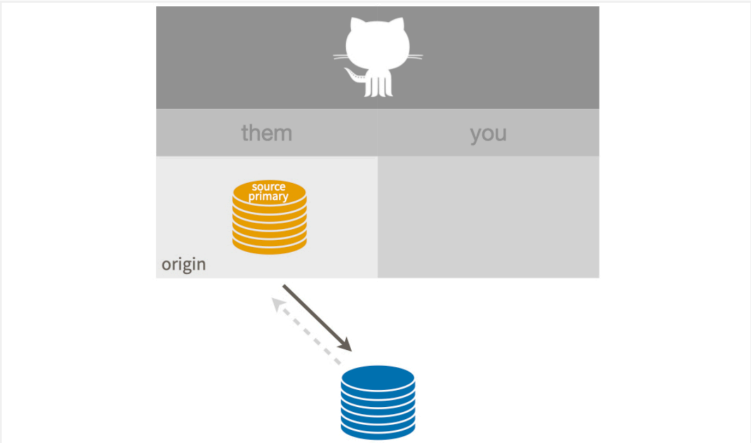
- If the **local** repo exists first:
  - For detailed instructions see Existing project, GitHub last.
  - `usethis::use_github()`.
  - Command line/RStudio: Task can't be completed fully (see Chapter 25.2)
- If the **remote** repo exists first:
  - For detailed instructions see New RStudio Project via git clone.
  - `usethis::create_from_github("OWNER/REPO", fork = FALSE)`.
  - Command line: `git clone <URL>`
  - RStudio: `File > New Project > Version Control > Git`

Ours



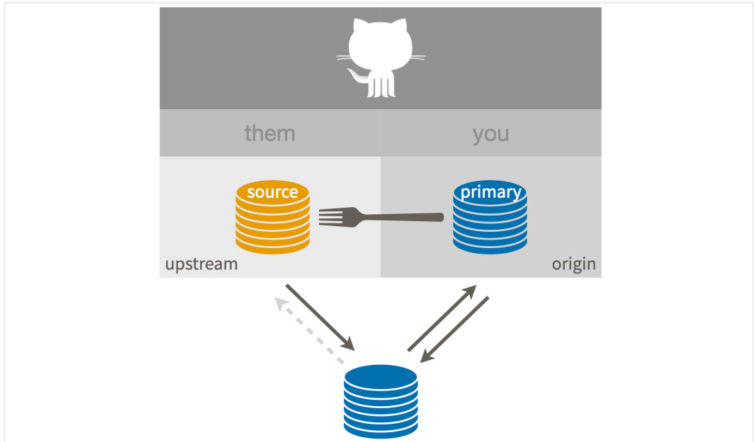
- `origin` is a **source**.
- `origin` is your **primary** repo.
- How to achieve?
  - same as above, but specify `usethis::use_github(organisation = "ORGNAME")`.

Theirs



- How does this setup happen?
- Cloning the source repo (via `git clone <UR` or Git client)
- `usethis::create_from_github("OWNER/RE fork = FALSE)`.
- What if you do want to make a pull request
  - Should have done fork-and-clone instead clone.

Fork (of theirs)

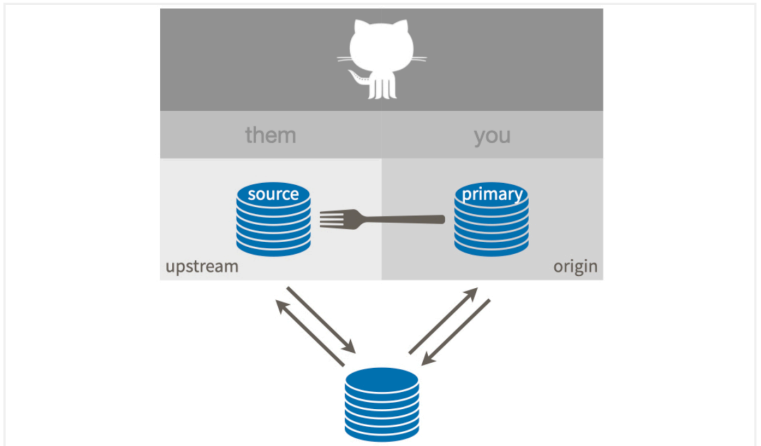


- You have a fork (`origin`) of the source repo (`upstream`).
- Your local repo can pull (not push) changes from `upstream`.
- `origin` is your **primary** repo (push/pull permission).
- You can make a pull request back to the source repo via your fork.

Fork (of theirs) - setup

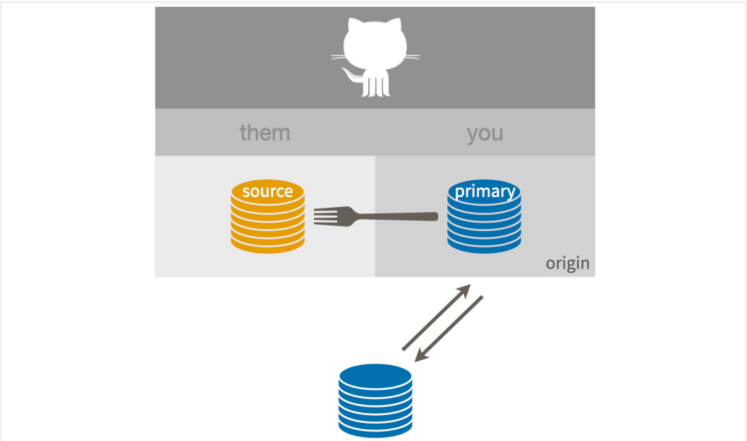
- Detailed instructions are given in Fork and clone.
- `usethis::create_from_github("OWNER/REPO", fork = TRUE)`.
- Command line Git or RStudio: You can't complete this task fully

Fork (of ours)



- Offers maximum flexibility for advanced us
- How to achieve?
  - Same as the regular fork setup above.
  - `usethis::create_from_github("OWNER/`
    `fork = TRUE)`.

Fork (salvageable)



- How does this happen?
  - Cloning your own fork, then stopping there.
- Convert into the fork setup above:
  - For detailed instructions see Finish the fork and clone setup.
  - Add source repo as the `upstream` remote.
  - Set `upstream/main` as the upstream tracking branch for local main.

Daily workflows, part 1

Chapter 27: The repeated amend

Recap

- A *commit* is a snapshot of your project at a particular point in time.
  - It captures changes made to files and records them in the version history.
  - `git commit`
- *Amend* is used to rewrite the last commit (e.g., update commit message or files).
  - `git commit --amend`
  - Allows fixing minor issues without creating a new commit

Mountain Climbing Analogy

Think of your project as a mountain you're climbing.

Mountain Climbing Analogy

Coding without commits is like free-climbing:

- faster in the short-term, but
- higher chance of catastrophic failure long-term

Mountain Climbing Analogy

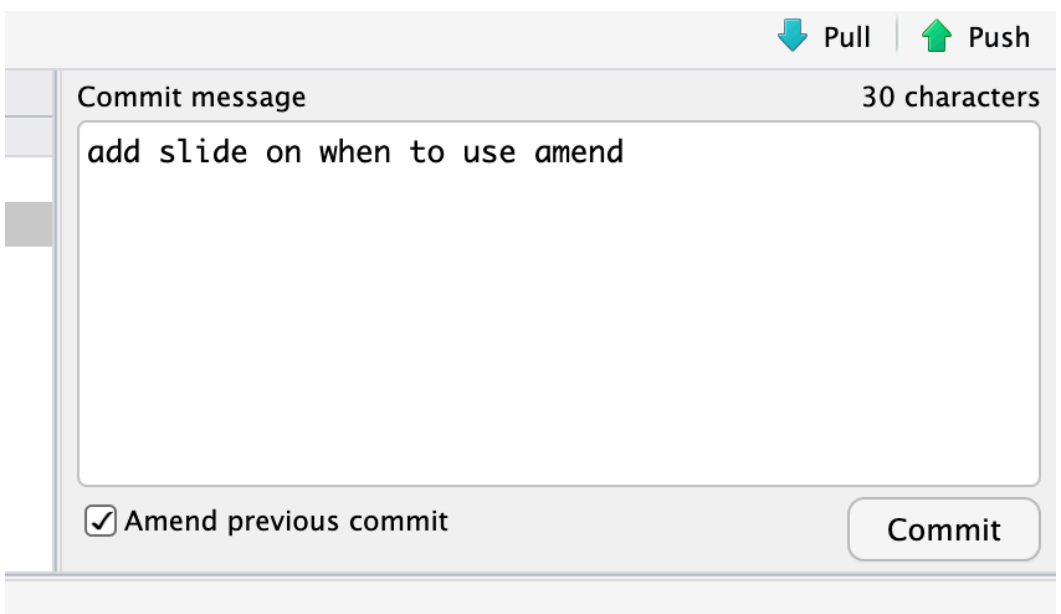
Using a commit is like using anchors when climbing.

Mountain Climbing Analogy

If you make a mistake, you can't fall past the previous commit

When to Use Amend

- If you made a minor error or forgot to add files
- If you need to update the commit message
  - `git commit --amend`
  - RStudio makes it very easy



Potential Pitfalls

- **Amending after pushing**:
  - You should avoid amending commits that have already been pushed to a shared repo
  - Amending pushed commits rewrites history and can cause problems for collaborators
  - Use with caution when working on shared repositories

Workflow - Step 1

Start with your project in a functional state.

- Use `git status` to check this.

```
kylieainslie@Kylies-MacBook-Pro happygitwithr % git status
On branch session_05
Your branch is up to date with 'origin/session_05'.
```

Workflow - Step 2

Initial status of project.

```
commit 4ea61df9a93bcc870e0a2e156a05cc4b606c44ca (origin/session_05)
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 09:37:12 2025 +0100

    add amend commit image

commit 93a5b79da3ccfef1c253c33349bb36935d9975ec
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 09:24:36 2025 +0100

    add slide on when to use amend

commit f6e9da5c57695647327b7778db209f00a3f0d6ad
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 08:42:01 2025 +0100

    add slides for Chapter 27

commit 5c5db43d46364e850cf58f74e85044c80bd27fce
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Wed Jan 29 14:30:25 2025 +0100

    add slides for Chapter 25
```

Workflow - Step 2

Make a small change towards a larger objective and commit it as `WIP` (work in progress)

```
git commit -m "WIP"
```

```
kylieainslie@Kylies-MacBook-Pro happygitwithr % git log
commit 20745d65e871170c6a29ae798cee39e013f972d8 (HEAD -> session_05)
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 09:40:29 2025 +0100

    WIP

commit 4ea61df9a93bcc870e0a2e156a05cc4b606c44ca (origin/session_05)
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 09:37:12 2025 +0100

    add amend commit image

commit 93a5b79da3ccfef1c253c33349bb36935d9975ec
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 09:24:36 2025 +0100

    add slide on when to use amend

commit f6e9da5c57695647327b7778db209f00a3f0d6ad
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Thu Jan 30 08:42:01 2025 +0100

    add slides for Chapter 27

commit 5c5db43d46364e850cf58f74e85044c80bd27fce
Author: Kylie Ainslie <ainslie.kylie@gmail.com>
Date:   Wed Jan 29 14:30:25 2025 +0100

    add slides for Chapter 25
```

Don't push!

Workflow - Step 2

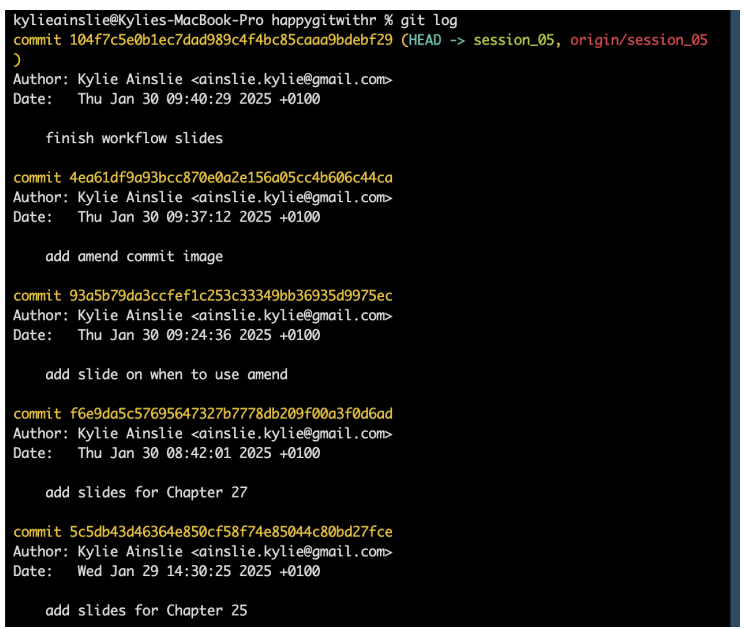
- Do a bit more work.
- Re-check that your project is still in a functional state.
- Stage and commit again, with amended commit: `git commit --amend --no-edit`

Don't push!

Workflow - Step 2

- Keep going until you've achieved your final objective.
- Now amend the commit, but add a real message

```
git commit --amend -m "Implement awesome feature"
```



Push!

Workflow - Step 3

What if you make some changes that break your project?

```
A -- B -- C -- WIP*
```

- You want to fall back to the last good state, represented by `WIP*`.
  - This is called a *hard reset* to the `WIP*` state.
  - Meaning: “reset my files to their state at the most recent commit”.

Workflow - Step 3

How to achieve?

- command line: `git reset --hard`
- RStudio:
  - Click on “Diff” or “Commit”.
  - Select a file with changes you do not want.
  - Use “Discard All” to discard all changes in that file.
  - Use “Discard chunk” to discard specific changes in a file.

Workflow - Step 4

Let's imagine you pushed this state to GitHub by mistake:

```
A -- B -- C -- WIP (85bf30a)
```

and proceeded to `git commit --amend` again locally, leading to this state:

```
A -- B -- C -- WIP* (6e884e6)
```

Workflow - Step 4

You have two choices:

1. You're working with collaborators:

- Save a copy of any files that exist locally (with changes) to a new file path, temporarily.
- Hard reset your local repo to C (`git reset --hard HEAD^`) and pull from GitHub.

```
A -- B -- C -- WIP (85bf30a)
```

- Bring any files saved temporarily elsewhere back into the repo now; save, stage, commit, and push.

```
A -- B -- C -- WIP (85bf30a) -- E
```

Workflow - Step 4

You have two choices:

2. You have no collaborators:

- Make sure your local commit has a real, non-"WIP" message.
- Force push your history to GitHub (`git push --force`).

```
A -- B -- C -- D
```

Chapter 28: Dealing with push rejection

Problem

Ladies

Our push is rejected!

```
$ git push
To https://github.com/YOU/REPO.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/YOU/REPO.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

This means that your local Git history and that on the GitHub remote are not compatible, i.e. they have diverged.

Solve the mystery

Use `git status` to figure out where the divergence has occurred.

Let's assume this is what we see on GitHub:

```
A -- B -- C
```

This is what we see locally:

```
A -- B -- D
```

Fix the problem

To reconcile the differences:

1. Pull the commit C (`git pull`)
2. Integrate the changes in C into your D-containing history.
3. Push again (`git push`)

```
A -- B -- C -- D
```

Lessons

1. She who pushes first wins!

- It is better for the overall health of a project to commit, push, and integrate more often, not less.

2. Stay in touch

- The sooner you know about C, the better. Pull (or fetch) often.

3. Use branches

- Branches afford explicit workflows for integrating different lines of work on your own terms.

Chapter 29: Pull, but you have local work

Problem

You want to pull upstream changes but have made local commits or uncommitted changes.

Example:

- Remote: `A--B--C`
- Local: `A--B--D`

Goal: Incorporate `C` into your local branch without losing `D` or uncommitted changes.

Case 1: Local Work is Uncommitted

Scenario:

- Remote: `A--B--C`
- Local: `A--B--(uncommitted changes)`

Happy Cases:

- Local changes don't overlap with remote changes (e.g., new files).
- Local changes affect different files than those changed upstream.

Outcome: `git pull` works and fast-forward merge happens.

```
                Remote: A--B--C

Local before 'git pull': A--B--(uncommitted changes)
 Local after 'git pull': A--B--C--(uncommitted changes)
```

Case 1: Local Work is Uncommitted

`git stash` works, sometimes

- Your changes affect a file that has also been changed in commit C.
- `git pull` won't work

Now what?

- `git stash` is a way to temporarily store some changes to get them out of the way.

```
git stash save
git pull
git stash pop
```

Outcome: `git stash` is successful.

Case 1: Local Work is Uncommitted

`git stash` with conflicts

- If the changes overlap, you'll get a conflict message when using `git stash pop`
- You must resolve the merge conflict.
  1. At each locus of conflict, pick one version or the other (upstream or stashed) or create a hybrid yourself.
  2. Remove the all the markers inserted to demarcate the conflicts.
  3. Save.

Case 1: Local Work is Uncommitted

`git stash` with conflicts

- Since `git stash pop` did not go smoothly,
  - we need to manually reset `git reset`
  - and delete the stash `git stash drop`

Outcome: We did it!

Case 2: Local work is committed

Scenario:

- Remote state is `A--B--C`.
- Local state is `A--B--D`.

Pull (fetch and merge)

- The simplest option is to fetch the commits from upstream and merge them
  - this is what `git pull` does
- If there are no conflicts, we're done!

Case 2: Local work is committed

Pull (fetch and merge)

- If commit C (on the remote) and commit D (local) have changes to the same parts of one or more files, this will result in a merge conflict.

```
jenny@2015-mbp ethel $ git pull
Auto-merging foo.R
CONFLICT (content): Merge conflict in foo.R
Automatic merge failed; fix conflicts and then commit the result.
```

- You must resolve each conflict before continuing.

Case 2: Local work is committed

Pull (fetch and merge)

Mark the affected file `foo.R` as resolved via `git add` and make an explicit `git commit` to finalize this merge.

```
jenny@2015-mbp ethel $ git add foo.R
jenny@2015-mbp ethel $ git commit
[main 20b297b] Merge branch 'main' of github.com:jennybc/ethel
```

Outcome: We've achieved this:

```
      Remote: A--B--C

Local before: A--B--D
 Local after: A--B--D--(merge commit)
                  \_C_/
```

Case 2: Local work is committed

Pull and rebase

- `git pull --rebase` creates a nicer history than `git pull` when integrating local and remote commits.
  - it creates a less cluttered commit history
  - can lead to more difficult to resolve merge conflicts

Outcome: If no conflicts, we get this:

```
      Remote: A--B--C

Local before: A--B--D
 Local after: A--B--C--D
```

- Jenny's rec for beginners: use `git pull`

Case 3: Other approaches

Scenario:

- Remote state is `A--B--C`.
- Local state is `A--B--(uncommitted changes)`.

Use a temporary branch for local work

The end of the session 5!

- **Next session:**
  - Subject: Daily Workflows, Part 2
  - Date: 18 February @ 17:00 CET
  - Presenter: Josefina Berardo
- **Meetup for the Chapters**
  - R-Ladies Amsterdam
  - R-Ladies Bergen
  - R-Ladies Den Bosch