

# **Github tutorial**

Sarah Endicott

2023-06-02

# Table of contents

|   |           |
|---|-----------|
| <b>Using GitHub for Reproducible Research</b>     | <b>4</b>  |
| <b>1 Installation and Setup</b>                   | <b>5</b>  |
| 1.1 Installation . . . . .                        | 5         |
| 1.2 Set username and email for git . . . . .      | 5         |
| 1.3 Let git talk to GitHub . . . . .              | 5         |
| 1.4 Start a new project with GitHub . . . . .     | 6         |
| 1.5 Work on a project . . . . .                   | 7         |
| 1.5.1 Sync changes to GitHub: Push . . . . .      | 8         |
| 1.5.2 Sync local copy from GitHub: Pull . . . . . | 8         |
| 1.6 Add an existing project to GitHub . . . . .   | 8         |
| 1.7 Issues with installation . . . . .            | 9         |
| 1.8 Git terminology summary 1 . . . . .           | 9         |
| <b>2 Using Git and GitHub</b>                     | <b>10</b> |
| 2.1 Tour of GitHub repository . . . . .           | 10        |
| 2.1.1 Code . . . . .                              | 10        |
| 2.1.2 File on GitHub . . . . .                    | 10        |
| 2.1.3 Viewing a commit . . . . .                  | 12        |
| 2.1.4 Issues . . . . .                            | 13        |
| 2.1.5 Settings . . . . .                          | 13        |
| 2.1.6 GitHub Organizations . . . . .              | 14        |
| 2.2 Daily Git Usage . . . . .                     | 14        |
| 2.2.1 When to commit . . . . .                    | 14        |
| 2.2.2 What to commit . . . . .                    | 14        |
| 2.2.3 When to push . . . . .                      | 15        |
| 2.2.4 When to pull . . . . .                      | 15        |
| 2.2.5 Merge conflicts in Push/Pull . . . . .      | 15        |
| 2.2.6 Resolving merge conflicts . . . . .         | 15        |
| 2.2.7 Avoiding merge conflicts . . . . .          | 16        |
| 2.3 Tools for Collaboration . . . . .             | 17        |
| 2.3.1 Branches . . . . .                          | 17        |
| 2.3.2 Forks . . . . .                             | 21        |
| 2.4 Git Terminology Summary 2 . . . . .           | 21        |

|          |                                      |           |
|----------|--------------------------------------|-----------|
| <b>3</b> | <b>Miscellaneous tips and tricks</b> | <b>22</b> |
| 3.1      | Easy Extensions . . . . .            | 22        |
| 3.2      | Advanced Uses . . . . .              | 22        |
|          | <b>References</b>                    | <b>23</b> |

# Using GitHub for Reproducible Research

This contains a set of tutorials for setting up and learning to use Git and GitHub with R and RStudio. It borrows heavily from [Happy Git With R](#) by Jenny Bryan but with less details.

# 1 Installation and Setup

## 1.1 Installation

- Install R and Rstudio (available from ECCC Software Centre)
  - Software Centre can be very slow to install, you can install R and RStudio without administrator permissions from the internet as well <https://posit.co/download/rstudio-desktop/>
- Install Git <https://gitforwindows.org/> (does not require admin permissions)
  - NOTE: When asked about “Adjusting your PATH environment”, make sure to select “Git from the command line and also from 3rd-party software”. Otherwise, accept the defaults.
- Create a GitHub account <https://github.com/>
- Install R packages: Open RStudio and run in the console:
  - `install.packages("usethis")`

## 1.2 Set username and email for git

- `usethis::use_git_config(user.name = "Jane Doe", user.email = "jane@example.org")`
  - `user.name` is the name associated with your git commits so just make it informative for your collaborators (ie: actual name, github username)
  - `user.email` must match your GitHub account email

## 1.3 Let git talk to GitHub

- `usethis::create_github_token()`
  - Opens GitHub webpage: select “repo”, “user”, and “workflow” scopes
- `gitcreds::gitcreds_set()`

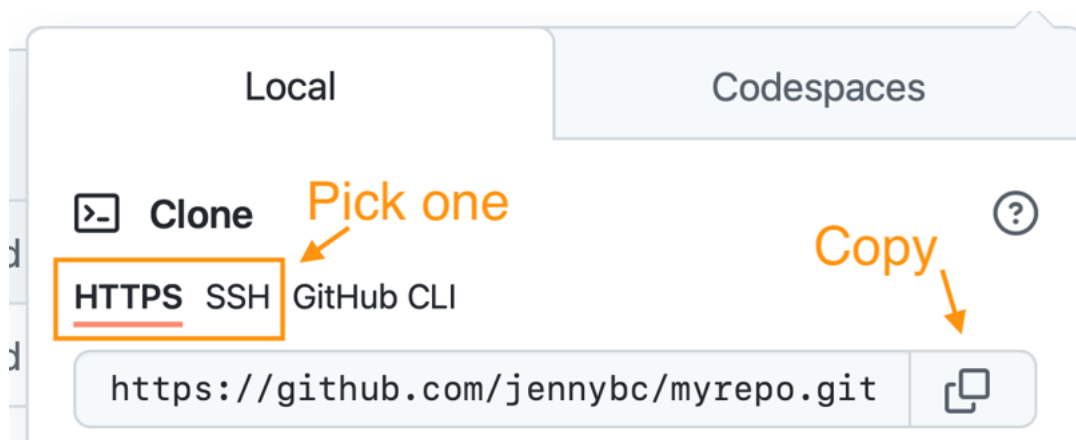
## 1.4 Start a new project with GitHub

**Step 1:** Make a new repo on GitHub

- Go to <https://github.com> and make sure you are logged in.
- Near “Repositories”, click the big green “New” button.
  - How to fill this in:
    - \* Repository template: No template.
    - \* Repository name: myrepo . Like a variable name, in code: descriptive but brief, no whitespace. Letters, digits, - , . , or \_ are allowed.
    - \* Description: any short description of the project
    - \* Public.
    - \* Initialize this repository with: Add a README file.
    - \* Click the big green button that says “Create repository”.

**Step 2:** Copy repo URL

- Now click the big green button that says “<> Code”.
- Copy a clone URL to your clipboard. Use the HTTPS URL.



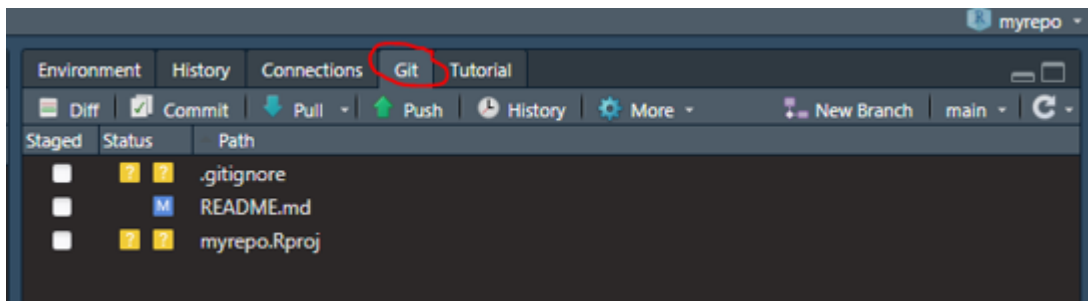
**Step 3:** Clone into a new project in RStudio

- File > New Project > Version Control > Git. In the “repository URL” paste the URL of your new GitHub repository.
  - Be intentional about where you create this Project. Don’t put it inside another git repository.
  - I suggest you “Open in new session”.
- Click “Create Project” to create a new directory,

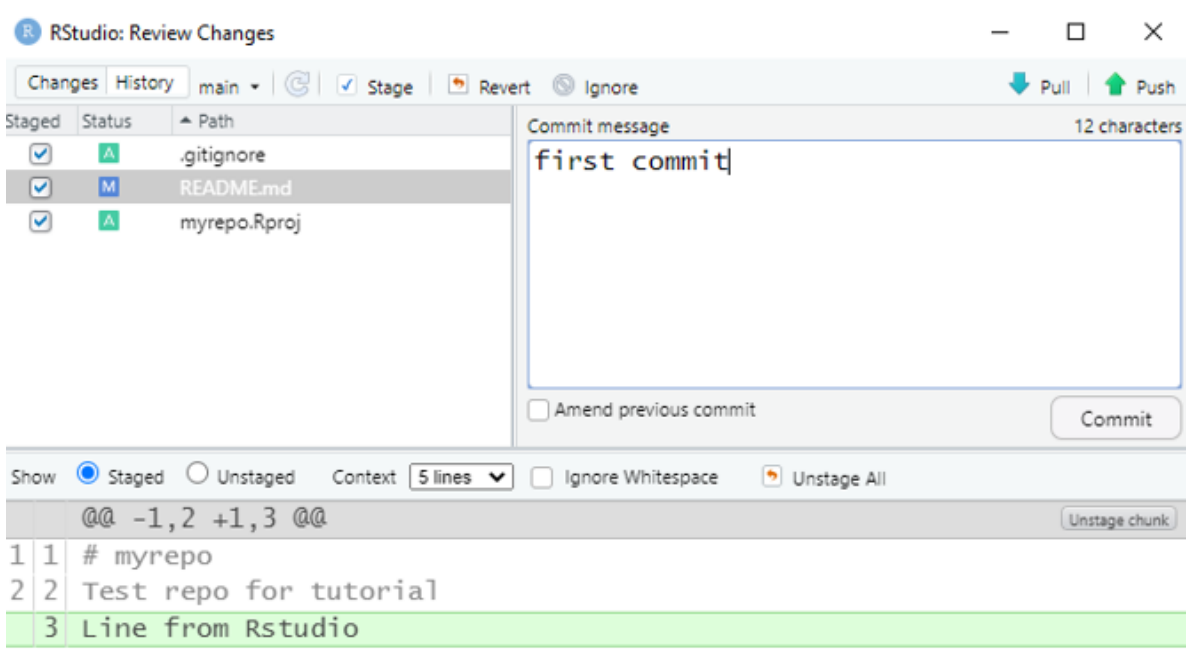
- This should download the README.md file that we created on GitHub in the previous step. Look in RStudio's file browser pane for the README.md file.
- Behind the scenes, RStudio has done this for you: `git clone https://github.com/see24/myrepo.git`

## 1.5 Work on a project

- Edit the README.md file, e.g., by adding the line “This is a line from RStudio”.
- Save the file locally
- On the Git pane click commit

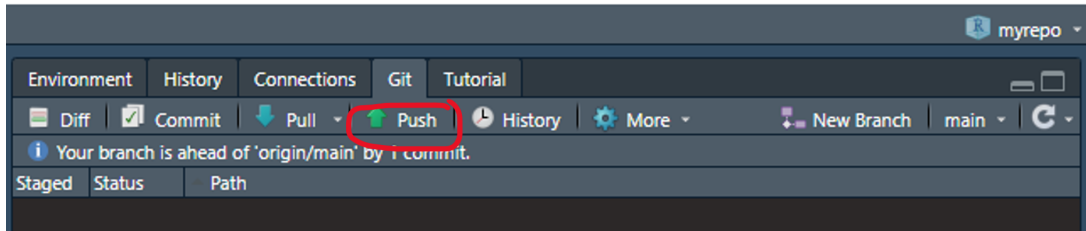


- In the pop-up review the changes at the bottom
- Check the “Staged” box and type a commit message and click “Commit”



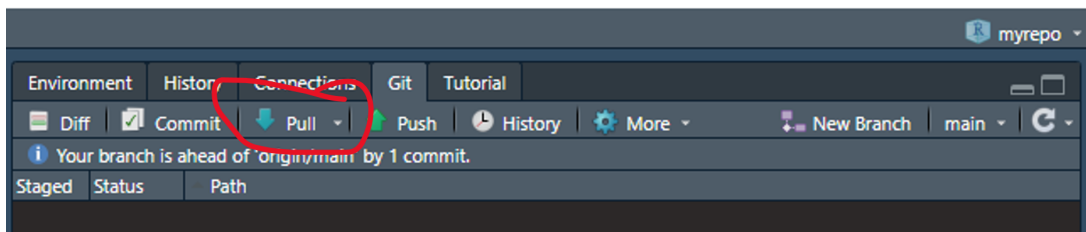
### 1.5.1 Sync changes to GitHub: Push

- Click “Push” in the Rstudio Git pane
- Look at the repo on GitHub so see the new line is there



### 1.5.2 Sync local copy from GitHub: Pull

- In the GitHub repo main page
- In the upper right corner of the Readme, click on the pencil
- Add a line eg : “Line added from GitHub.”
- Click “Commit changes.”
- In RStudio click Pull on the Git pane
- You should see the new line in the Readme



## 1.6 Add an existing project to GitHub

- Create a new repo and Rstudio project in the same way as above
- Simply copy all files into the newly created folder on your local computer
- Stage and commit all files that you **want to store on GitHub**
  - Nothing sensitive ie passwords, keys etc (you can have a private repo if you are not ready to share code with the world)
  - Probably not large datasets
  - Use . gitignore to avoid git tracking things (more on this later)



*This is the simplest way to do it but there are more advanced, more traditional git ways to do it: <https://happygitwithr.com/existing-github-last.html>*

## 1.7 Issues with installation

If RStudio is not finding a git installation: + Restart RStudio and try again + If still not working, run this in the windows command line: `git --exec-path` + Copy the path, then in RStudio click Tools > Global Options > Git/SVN and set the Git executable by clicking browse, pasting the path in the address bar and selecting the git.exe file. + Restart RStudio again

See <https://happygitwithr.com/rstudio-see-git.html> for more instructions on troubleshooting

## 1.8 Git terminology summary 1

- Repository (repo): Folder that contains a hidden .git file that tracks changes made to files in that folder. The folder can “live” on your local computer or a server like GitHub’s. On GitHub the repository is also the web page where all the files are stored among other things
- Push: Copy changes from your local version of the repo to the GitHub version
- Pull: Copy changes from the GitHub version of the repo to your local version
- Clone: make a copy of a git repository. By default in R studio this is connected to the GitHub version (called the remote or origin)
- Commit: A marker that is kept in the git history and helps to incrementally track changes. Made useful by descriptive commit messages

## 2 Using Git and GitHub

### 2.1 Tour of GitHub repository

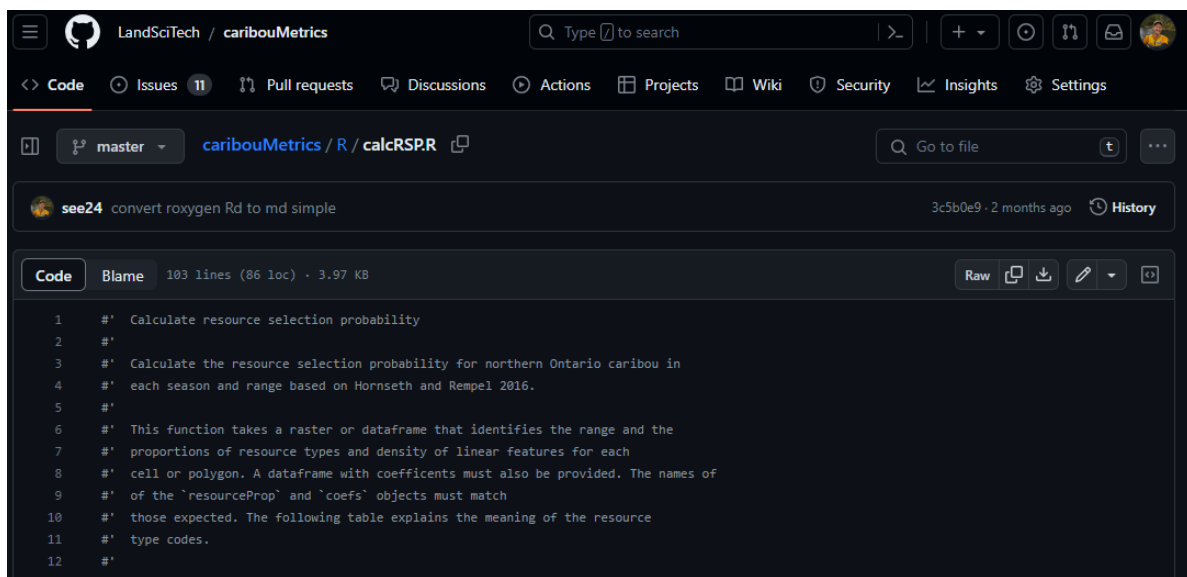
<https://github.com/LandSciTech/caribouMetrics>

#### 2.1.1 Code

- Top folder of repo.
- Acts as landing page.
- Displays the readme file.

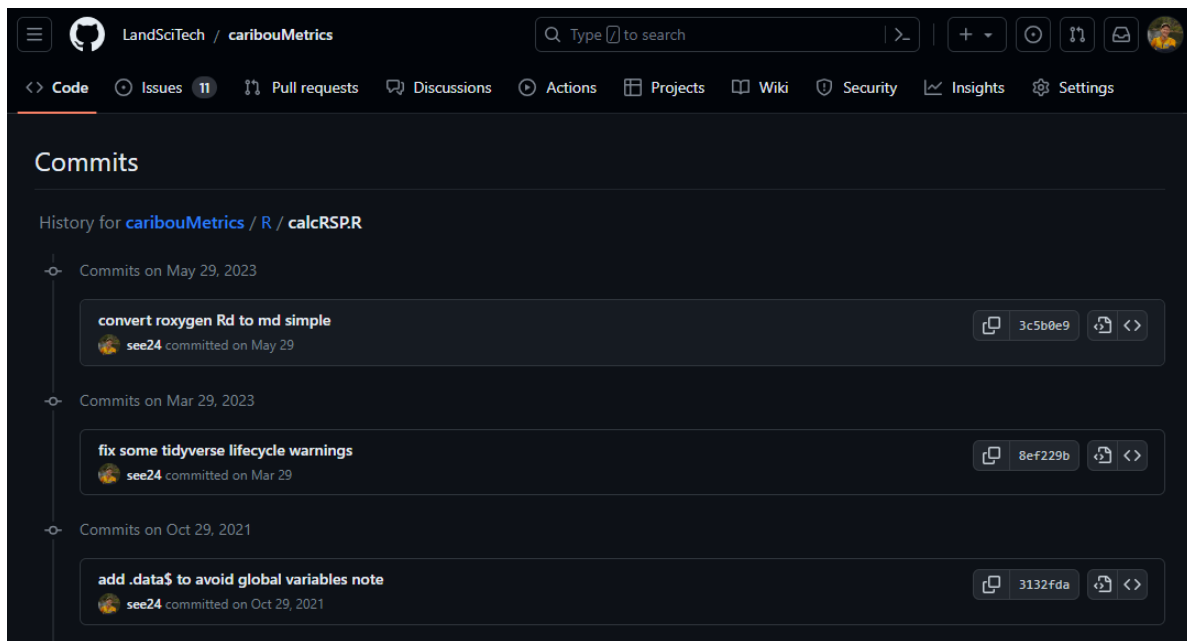
#### 2.1.2 File on GitHub

- Can view and edit code on GitHub.

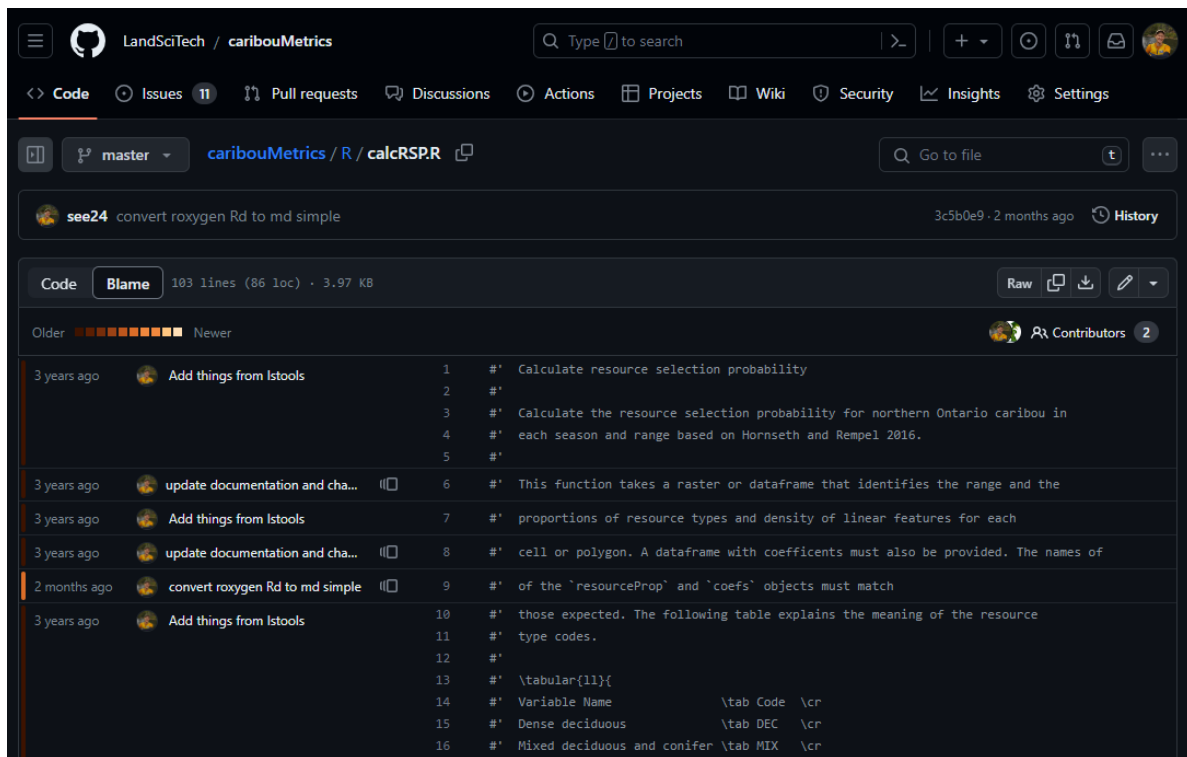
A screenshot of a GitHub repository page. The repository is 'LandSciTech / caribouMetrics'. The file being viewed is 'caribouMetrics / R / calcRSPR'. The file is 103 lines (86 loc) and 3.97 KB. The commit is by 'see24' with the message 'convert roxygen Rd to md simple' and hash '3c5b0e9' from 2 months ago. The code is shown in a dark theme with line numbers 1 through 12 visible. The code is a comment block in R, explaining the function 'calcRSPR' and its parameters. The code is as follows:

```
1 #' Calculate resource selection probability
2 #'
3 #' Calculate the resource selection probability for northern Ontario caribou in
4 #' each season and range based on Hornseth and Rempel 2016.
5 #'
6 #' This function takes a raster or dataframe that identifies the range and the
7 #' proportions of resource types and density of linear features for each
8 #' cell or polygon. A dataframe with coefficients must also be provided. The names of
9 #' of the 'resourceProp' and 'coefs' objects must match
10 #' those expected. The following table explains the meaning of the resource
11 #' type codes.
12 #'
```

- Can see what has changed over time.
- History shows all commits that have affected the file.



- Blame shows for each line, who last edited, when, and the commit message.



### 2.1.3 Viewing a commit

- Shows all files changed and highlights which parts changed.

remove raster in examples and docs. Test mostly working but not movin...  
-g window

migrate-terra

see24 committed last week 1 parent 59baa7c commit f82f9f1

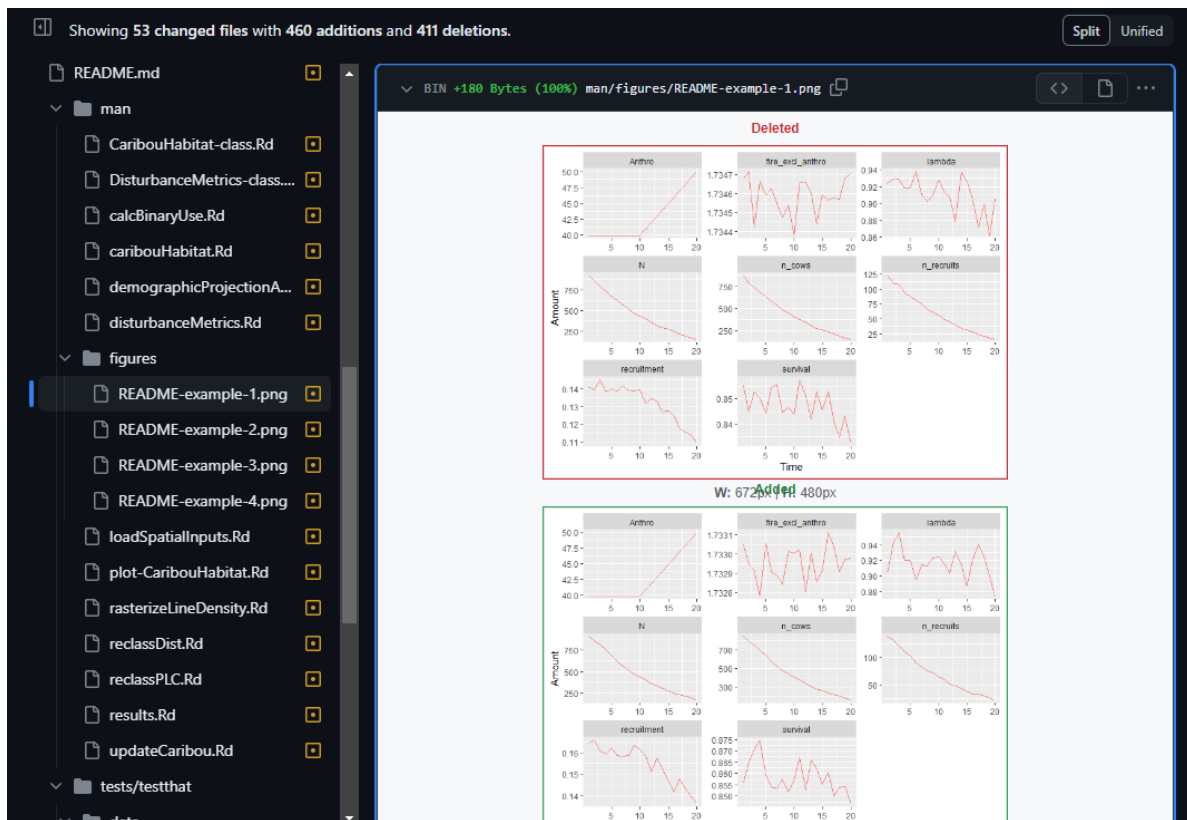
Showing 53 changed files with 460 additions and 411 deletions.

updateDisturbance.R  
README.Rmd  
README.md  
man  
CaribouHabitat-class.Rd  
DisturbanceMetrics-class....  
calcBinaryUse.Rd  
caribouHabitat.Rd  
demographicProjectionA...  
disturbanceMetrics.Rd  
figures

16 NAMESPACE

|    |   |    |                                       |
|----|---|----|---------------------------------------|
| 32 | import(purrr)                           | 32 | import(purrr)                         |
| 33 | import(sf)                              | 33 | import(sf)                            |
| 34 | import(tidyr)                           | 34 | import(tidyr)                         |
| 35 | - importClassesFrom(raster,RasterLayer) | 35 | + importClassesFrom(terra,SpatRaster) |
| 36 | importFrom(methods,"slot<-")            | 36 | importFrom(methods,"slot<-")          |
| 37 | importFrom(methods,as)                  | 37 | importFrom(methods,as)                |
| 38 | importFrom(methods,is)                  | 38 | importFrom(methods,is)                |
| 39 | importFrom(methods,new)                 | 39 | importFrom(methods,new)               |
| 40 | importFrom(methods,slot)                | 40 | importFrom(methods,slot)              |
| 41 | importFrom(methods,slotNames)           | 41 | importFrom(methods,slotNames)         |
| 42 | - importFrom(raster,addLayer)           |    |                                       |
| 43 | - importFrom(raster,compareRaster)      |    |                                       |
| 44 | - importFrom(raster,cover)              |    |                                       |

- Works for images too!



## 2.1.4 Issues

A place for tracking things that need to get done or reporting bugs or unexpected results.

- Can be submitted by anyone with access to the repo.
- Once they are dealt with they can be closed but are still kept.
- If you are reporting a bug it is key to have a [minimum reproducible example](#) so others can see what you are seeing and try to help.
- anywhere else on GitHub if you type # and then the issue number it will automatically link to the issue. In a commit message this will link the commit to the issue so you can see what was changed to address it. Example [here](#).
- Issues can also be used for To-do lists or project planning and can be tagged with a certain category or assigned to specific users.

## 2.1.5 Settings

Repository options and settings. There are a lot of complex options but the most beginner relevant are:

- Changing repository visibility: Bottom of General section, usually changing to public once far enough along
- Deleting repository: Bottom of General section, no going back, consider archiving if it is just out of date
- Adding collaborators: under Access > Collaborators and teams, click add people and type their GitHub user name. Only needed for private repositories

### 2.1.6 GitHub Organizations

- You can create an organization to group repos together and manage collaborators. For example:
- <https://github.com/LandSciTech>
- <https://github.com/PredictiveEcology>
- <https://github.com/na-pops>

## 2.2 Daily Git Usage

### 2.2.1 When to commit

- Often!
- But not too often!
  - Use “Amend previous commit” checkbox when you want to make sure to commit but aren’t sure you are done yet
- Try to make each commit distinct and accomplish one thing. eg:
  - Make data cleaning script
  - Create exploratory plots
  - Fix bug in data cleaning handling of dates
  - Update exploratory plots with dates

### 2.2.2 What to commit

- Everything!
  - Git can track any file but it does a better job with raw text files (eg: .R, .Rmd, .html, .md, .py, .sh, .txt)
  - For files like word docs or pdfs it can’t track the content and tracks the whole file every time you make a change
- Except!:

- Nothing sensitive ie passwords, keys etc (you can have a private repo if you are not ready to share code with the world but still don't store passwords)
- Probably not large datasets. I just keep these locally but would be better to have them on a shared drive and download them programmatically
- .gitignore: a file at the top level of git repo that tells git what not to track.
  - Uses regular expressions to match file or folder names or types.
  - Example file: <https://github.com/LandSciTech/caribouMetrics/blob/master/.gitignore>

### 2.2.3 When to push

- Fairly often. If you are working alone pushing is a way to back up your files. If you are collaborating it is a way to make your work available to others. If you don't push and then a collaborator makes changes to the same file it gets a bit tricky (but fixable).
- Once you push you can't use the "Amend previous commit" trick
- If you find yourself reluctant to push because you aren't ready for others to use your work consider making a branch (see below)

### 2.2.4 When to pull

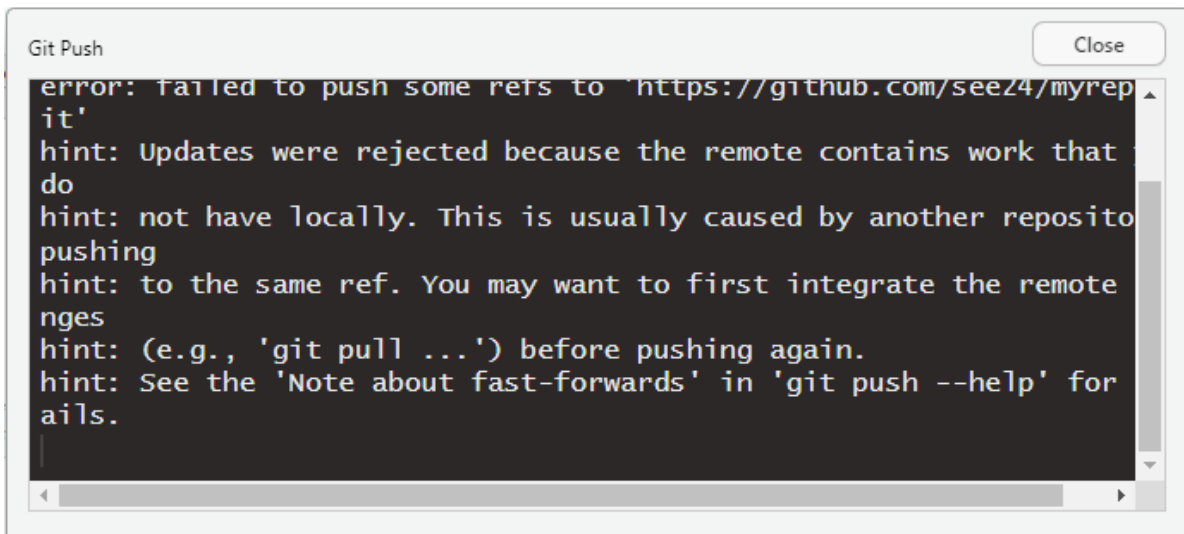
- Ideally every day, or when a collaborator lets you know they pushed
- Pulling often prevents getting out of sync with collaborators
- Before pulling be sure to commit all your local work
- Good practice to pull before pushing but git will normally warn you if you forget.

### 2.2.5 Merge conflicts in Push/Pull

- If a collaborator pushed their changes after you last pulled you will need to pull before you can push. If your changes don't conflict git will automatically merge their changes with yours.
- Merge conflicts: when a collaborator made changes that overlap your changes. Git can't automatically fit them together you have to review and pick the part to keep.

### 2.2.6 Resolving merge conflicts

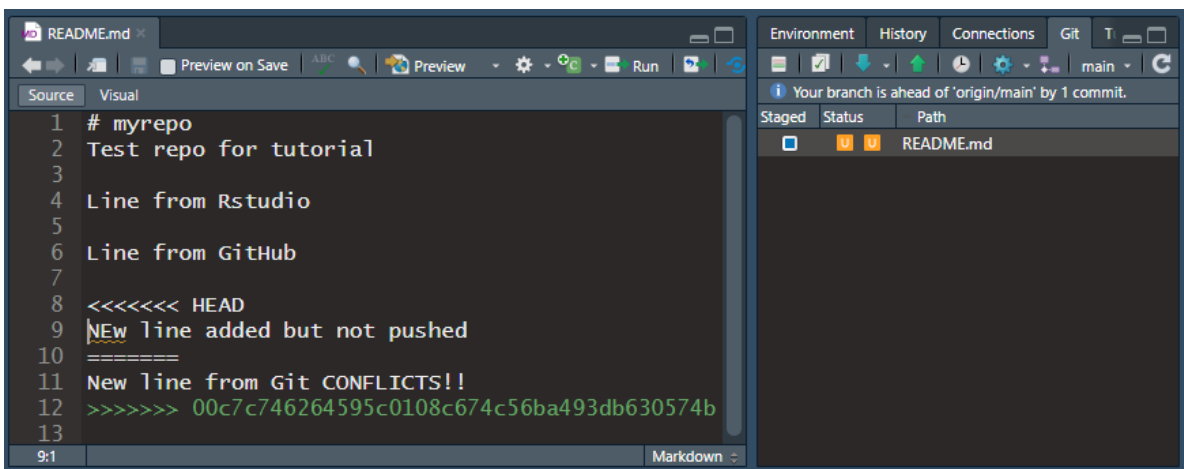
This is a typical scenario with a merge conflict: "Ah! I pulled at the start of the day but then a collaborator pushed a change to the same line and now when I try to push it says I have to pull first."

A screenshot of a 'Git Push' terminal window. The window has a title bar with 'Git Push' and a 'Close' button. The terminal text shows an error: 'error: failed to push some refs to 'https://github.com/see24/myrepo'. It then provides hints: 'Updates were rejected because the remote contains work that you do not have locally. This is usually caused by another repository pushing to the same ref. You may want to first integrate the remote changes (e.g., 'git pull ...') before pushing again. See the 'Note about fast-forwards' in 'git push --help' for details.'

```
Git Push
Close

error: failed to push some refs to 'https://github.com/see24/myrepo'
hint: Updates were rejected because the remote contains work that
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
ails.
```

“And then when I pull I get merge conflicts!”

A screenshot of the RStudio IDE. The main editor shows a file named 'README.md' with a merge conflict. The conflict is between the local 'HEAD' and a remote commit. The text in the editor is: '1 # myrepo', '2 Test repo for tutorial', '3', '4 Line from Rstudio', '5', '6 Line from GitHub', '7', '8 <<<<<< HEAD', '9 New line added but not pushed', '10 =====', '11 New line from Git CONFLICTS!!', '12 >>>>>> 00c7c746264595c0108c674c56ba493db630574b', '13'. The status bar at the bottom shows '9:1' and 'Markdown'. On the right, the 'Git' pane shows the file 'README.md' with an orange 'U' icon, indicating a conflict.

```
README.md
Source Visual
1 # myrepo
2 Test repo for tutorial
3
4 Line from Rstudio
5
6 Line from GitHub
7
8 <<<<<< HEAD
9 New line added but not pushed
10 =====
11 New line from Git CONFLICTS!!
12 >>>>>> 00c7c746264595c0108c674c56ba493db630574b
13
9:1 Markdown
```

Not too hard to fix. Go through each file that has the orange U in the Git pane. Find the location of the conflict. HEAD is your local version and the alphanumeric string is the commit id for the remote version that conflicts. Pick the one you want and delete all the marker lines («<, === and »>). Then commit and continue on with your work.

## 2.2.7 Avoiding merge conflicts

- Pull regularly
- Keep in touch with collaborators so you are not working on the same lines at the same time.
- Use a branch



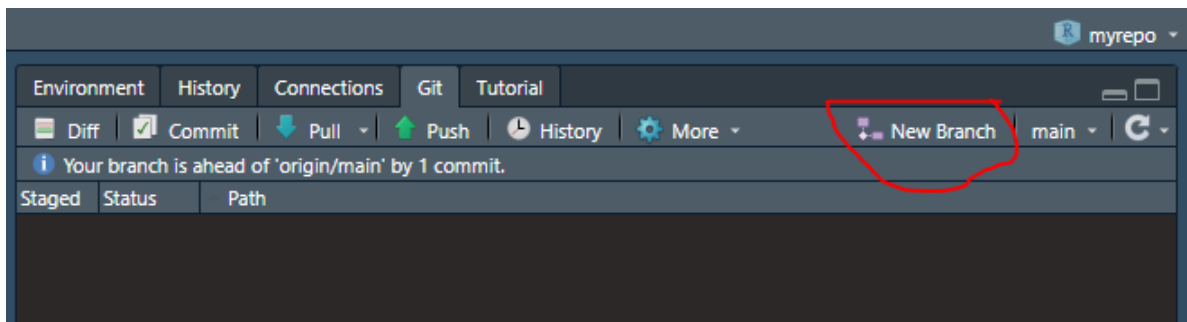
## 2.3 Tools for Collaboration

### 2.3.1 Branches

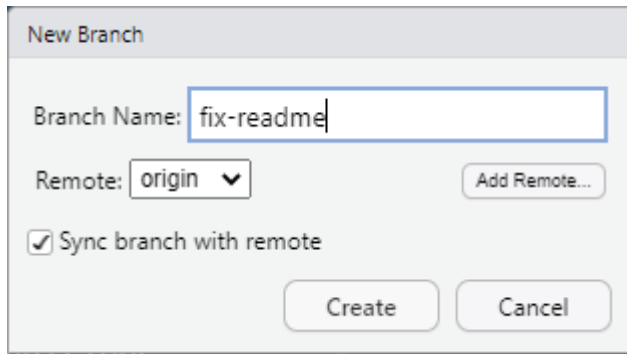
- A stream of commits that diverges from the main stream until it is ready to re-join.
- Helpful for starting a new version of something while making sure others can keep using the old version
- Example we want to convert some functions used in a paper to become an R package but Josie is working on writing the paper and needs the old version to keep working. I make a branch where I re-arrange everything into a package. If Josie makes changes to the main branch that affect the functions I can see those and merge them into my branch.
- See <https://happygitwithr.com/git-branches.html#git-branches> for how to manage branches with the command line but it can also be done through the Rstudio IDE and GitHub for the most part.

#### 2.3.1.1 Make a new branch in RStudio

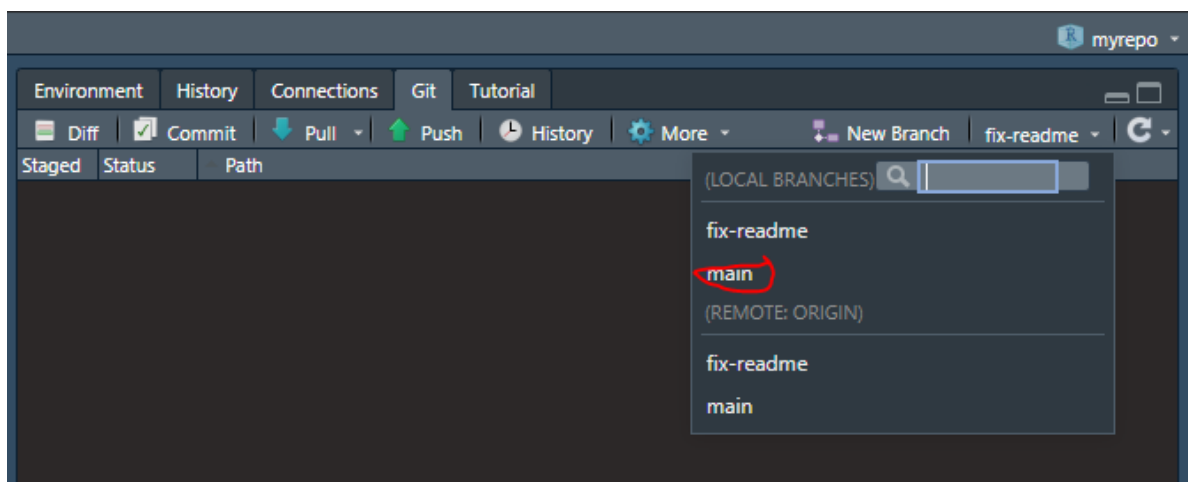
- Click New Branch in the Git pane



- Give it a name using dashes or underscores for spaces and no special characters. The “Remote” is the GitHub version of the repository, almost always leave the default. “Sync branch with remote” will also create the branch on GitHub which we typically want. Click “Create”

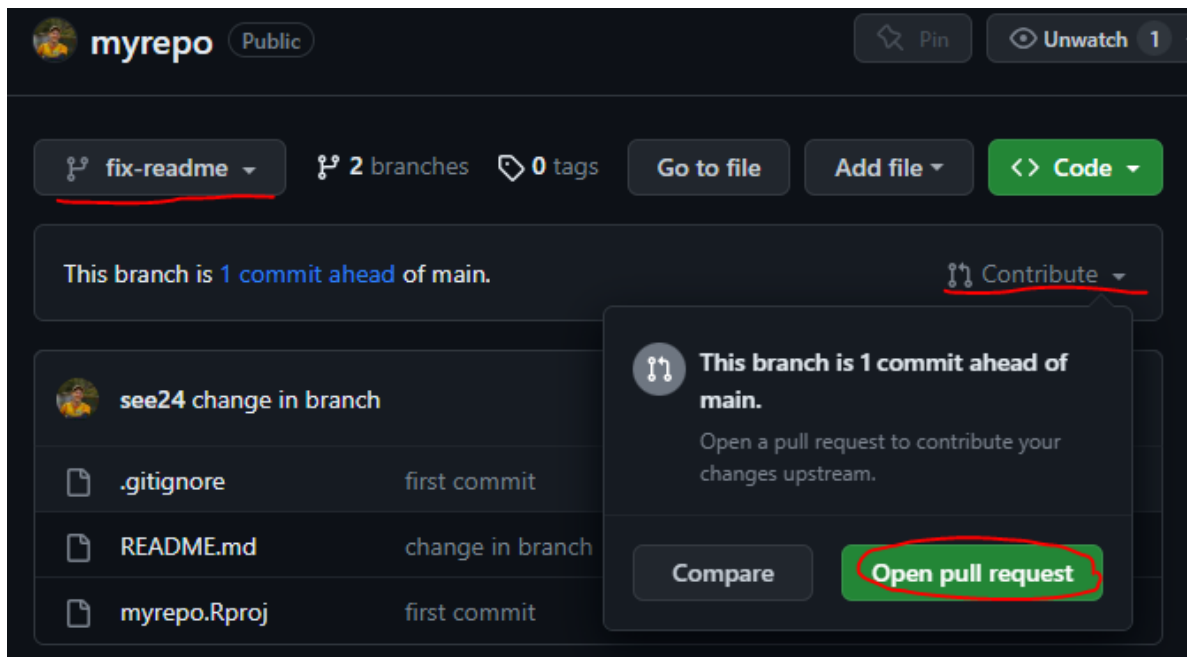


- Make a change, save, commit and push as usual.
- To change back to the main branch click the dropdown beside the “New Branch” button and under “Local Branches” select “main”. In git this is called “checking out” the branch.



### 2.3.1.2 Merge the branch back into main with a Pull Request

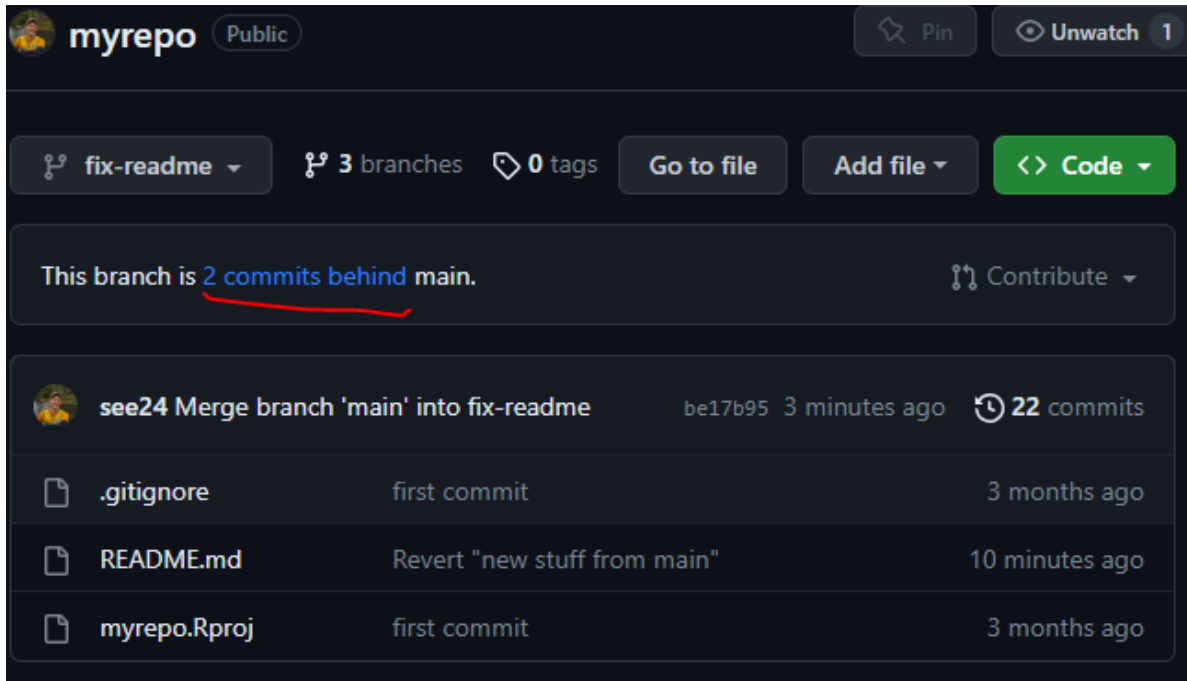
- Once you are ready to incorporate the changes in your branch back into the main code stream you “merge” your branch with the main branch.
- The easiest way to do that is on GitHub. Once you have made commits on your branch and pushed to the repo on GitHub
- On the GitHub repo page select the new branch from the drop down, then click Contribute > Open pull request



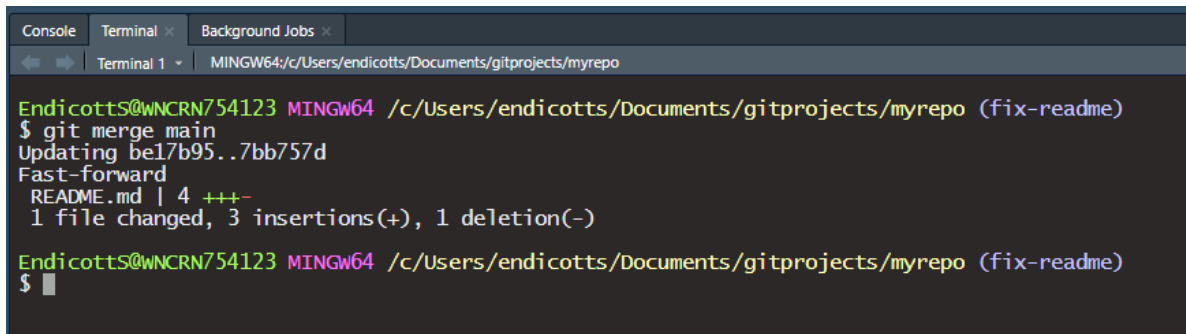
- You can give the pull request a descriptive name and add comments to explain what you are trying to do or link to issues that are being addressed. Then create the pull request.
- The pull request (PR) is a page that summarizes the changes you propose to make to the main branch. It includes the list of commits in the branch since it diverged from main and the diffs for files that have changed. Collaborators can leave comments on the PR and discuss it or add new commits before it is merged.
- PRs are the main way you can contribute to open source projects, and in that case the owner of the project will get the final decision whether to merge your changes into the main branch.
- If there are no merge conflicts and you have the authority to commit to the repos main branch you can click “Merge pull request” to merge your branch with main branch. Then you can delete the branch if it is no longer needed. On your local Rstudio session you will need to change back to the main branch and pull so that the changes are reflected in your local copy of the main branch.
- Branches don’t really prevent merge conflicts. But they do allow you to decide when you want to deal with reconciling your branch and the main branch so you can be more intentional about it. If the same parts of the code are changed in the main branch and your new branch you have to decide manually which one to keep by resolving the merge conflicts as described above. GitHub provides an editor to resolve conflicts online. In some situations the conflicts are too complex to resolve online and the merge must be done from the command line. GitHub provides instructions if this happens.

### 2.3.1.3 Merge changes from main into your branch

- Sometimes changes are made in main (or any other branch) that you want to include in your branch without incorporating the changes in your branch into main
- To do this you merge the main branch with yours. On GitHub if you go to your branch and click the “This branch is x commits behind main” url it will open a PR for merging main into your branch.



- A PR is often unnecessary for merging main into your branch, because most times your collaborators don't need to know about it. To avoid this you can use the command line to do the merge.
- In RStudio on the Console pane click the Terminal tab. This will open a command line where you can type git commands.
- Make sure you are in your branch that you want to merge into. Then type: `git merge main`

A screenshot of a Windows terminal window with tabs for Console, Terminal, and Background Jobs. The active tab is Terminal, showing a command prompt session. The user is at the directory /c/Users/indicotts/Documents/gitprojects/myrepo. They run the command 'git merge main'. The output shows the commit being updated from be17b95 to 7bb757d, a fast-forward merge, and a summary of changes to README.md (4 lines, 3 insertions, 1 deletion).

```
Endicotts@WNCNRN754123 MINGW64 /c/Users/indicotts/Documents/gitprojects/myrepo (fix-readme)
$ git merge main
Updating be17b95..7bb757d
Fast-forward
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

Endicotts@WNCNRN754123 MINGW64 /c/Users/indicotts/Documents/gitprojects/myrepo (fix-readme)
$
```

### 2.3.2 Forks

- A fork is a copy of another repository which is used to contribute to an open source project that you do not have permission to edit, or to simply make a copy and edit as you wish from there but with an acknowledgement that you used another repo as a starting point.
- you can create a fork on GitHub by clicking the fork button in the top right of all GitHub repos or with the `usethis::create_from_github()` command below which will also set up useful default settings. See [here](#) for more details.

```
usethis::create_from_github(  
  "https://github.com/OWNER/REPO",  
  destdir = "~/path/to/where/you/want/the/local/repo/",  
  fork = TRUE  
)
```

## 2.4 Git Terminology Summary 2

- Merge: Combine two versions of a file. These can be the local and GitHub (remote) copies or two different branches.
- Conflict: When merging can't be done automatically because two versions of the code have edited the same part of a file.
- Branch: A copy of the code that you want to keep separate from the main code at least for a time.
- Pull request: A special GitHub page that shows what will be added to the main branch when another branch is merged.
- Fork: A copy of a repository that you don't own. Used for contributing to open source projects.

## 3 Miscellaneous tips and tricks

### 3.1 Easy Extensions

- Making a repo work like a simple website <https://happygitwithr.com/workflows-browsability.html#workflows-browsability>
- Installing a Git Client <https://happygitwithr.com/git-client.html#git-client>
- Good default folder structure and setup for a typical analysis project: <https://frbcesab.github.io/rcompendium/index.html>

### 3.2 Advanced Uses

- Host a website: [https://www.emilyzabor.com/tutorials/rmarkdown\\_websites\\_tutorial.html](https://www.emilyzabor.com/tutorials/rmarkdown_websites_tutorial.html)
- Use GitHub Actions for continuous integration: <https://beamilz.com/posts/series-gha/2022-series-gha-1-what-is/en/>
- Using GitHub to manage frequently updated data:
  - <https://doi.org/10.1371/journal.pbio.3000125>
  - <https://www.updatingdata.org/githubactions/>
  - <https://doi.org/10.1111/2041-210X.13982>

# References

[Happy Git and GitHub for the useR](#) by Jennifer Bryan ::: {#refs} :::