# Setting up git for (solo) data science workflow

Ronald (Ryy) Glenn Thomas

2024-02-23

## Table of contents



## 1 Introduction

Figure 1: purrr

Version Control for biostatistics/data-science data analysis is the challenge.

Lets take it one step at a time.

Scenario 1: git user `@rgt47` has been working on a data analysis for some ADNI data. Its moderately complex and uses lots of packages. He's ready to ask his team to join the analysis process. What are the first steps to do that?

Start by adding the first github user `@rgt4748` to the project.

# 2 Methods

## 2.1 Invite a colleague to collaborate on a github repository

Start by logging into `Github` and navigating to the repository to be shared, say `x24`. Select `settings` (in top row of tabs) and then the `collaborators` tab (in the left panel)

Select green "add people" button, then "invite collaborator", then "add rgt4748 to this repository", in the center of the page.

Enter the github user name (rgt4748) and select "Add rgt4748" t Now rgt4748 should login and accept invitation:

click "message" icon in upper right corner.

Select "Invitation to join rgt47/x23 from rgt47", then "Accept Invitation" green button in center of page.

They'll be taken to the `rgt47/x24` repo.

Now on their workstation they can clone repository with the following code:

```
> git clone https://github.com/rgt4748/x24.git
```

```
> cd x24
> git branch myedits
> git checkout myedits
> vim x24.Rmd
```

modify `title: "R2"` to `title: "changed R2"` and save edits.

```
> git add .
> git commit -m "sample edit"
> git push origin myedits #(?)
> git checkout master
> git merge myedits
> git branch -d myedits #(delete branch)
```

Click `contribute` button, then `open pull request`, `create pull request`, enter `title` and `description`.

login as rgt47. you'll see one notification. check `files changed`

# 3 Appendix. GIT for nitwits

git init

git add fname

git status #see what happens on commit git commit -am "commit message"

git push

git branch work

git checkout work

… make changes … git add * git commit -m "something"

git checkout master

git merge work

git branch -d work

git log #see all commits

git checkout HASH #Restore old branch

Consider editing ./.git/config

View file in master branch. git show master:a101.Rmd | mvim -

Copy file from other branch (master) git checkout master uw.png

# 4 Appendix: Tips

Troubleshooting git pull –allow-unrelated-histories

Rule 6: Use the Imperative mood

A valuable practice involves crafting commit messages with the underlying understanding that the commit, when implemented, will achieve a precise action. Construct your commit message in a manner that logically completes the sentence "If applied, this commit will…". For instance, rather than,git commit -m "Fixed the bug on the layout page" . use this git commit -m "Fix the bug on the layout page"

In other words, if this commit were to be applied, it would indeed fix the bug on the layout page.

Rule 7: Explain "What" and "Why", but not "How".

Limiting commit messages to "what" and "why" creates concise yet informative explanations of each change. Developers seeking "How" the code was implemented can refer directly to the codebase. Instead, highlight what was altered and the rationale for the change, including which component or area was affected.

Case Study: Angular's Commit Message Practices

Angular stands as a prominent illustration of effective commit messaging practices. The Angular team advocates for the use of specific prefixes when crafting commit messages. These prefixes include "chore: ," "docs: ," "style: ," "feat: ," "fix: ," "refactor: ," and "test: ." By incorporating these prefixes, the commit history becomes a valuable resource for understanding the nature of each commit. Tips

Remember to prioritize clear and meaningful communication through your commit messages. A well-crafted commit message serves as a story that explains 'what,' 'why,' but not 'how' a change was made. Remember, your commit history is a collaborative resource that future you and your team will rely on. Make it a habit to create commit messages that stand as informative, concise, and consistent narratives.

Interested in deepening your understanding of Git and evolving into a proficient "version controller"? Explore these exceptional resources:

- https://git-scm.com/doc
- https://git-scm.com/book/en/v2
- https://lab.github.com/
- https://www.atlassian.com/git/tutorials
- https://learngitbranching.js.org/
- https://www.gitkraken.com/git-cheat-sheet
- https://www.git-tower.com/learn/

## 5 References:

Best way to manage your dotfiles.

Git Basics — All You Need To Know as a New Developer. | by Gabriel Bonfim | Sep, 2023 | Medium