Setting up git for (solo) data science workflow

Ronald (Ryy) Glenn Thomas

2023-11-21

Table of contents

1 Introduction

Version Control for biostatistics is the challenge.

Lets take it one step at a time.

Scenario 1: rgt47 has been working on a data analysis for some ADNI data. Its moderately complex and uses lots of packages. He's ready to have his team join the analysis process. What are the first steps to do that? Start by adding git user rgt4748 to the team….

Reference:

Best way to manage your dotfiles.

2 Methods

create a branch:

create new branch for testing

git checkout -b test; git pull origin master; git push origin test; git checkout master; git pull origin test

- #merging branch back into master
-
- git checkout test
- git pull origin master
- git checkout master
- git merge test
- git push origin master
-
-
- #to delete branch
- git branch -d test
- git push origin –delete test

Draft ….

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

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/

3 References:

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