30538 Problem Set 3: git

Peter Ganong, Maggie Shi, and Dema Therese Maria

2024-10-21

Due Sat Oct 26 at 5:00PM Central. Worth 50 points.

Before you begin this problem set, you will need to install command-line git.

Your submission does not need runnable code. Instead, you will tell us either what code you ran or what output you got.

1. Late coins used this pset: **___**
2. Late coins left after submission: **___**

Instructions

This problem set has two parts: a **Solo** section and a **Partnered** section. Each section has its own GitHub repository and GradeScope submission.

Please read their respective instructions carefully to ensure you follow the correct workflow for each part.

SECTION 1 - Solo

---

1. **Setup Instructions**:

- Each student must individually accept the solo repository from GitHub Classroom, and complete the solo exercises.

2. **Submission Process**:

- "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (1 point)
- Knit your `ps3_solo.qmd` as a pdf.
- Push `ps3_solo.qmd` and `ps3_solo.pdf` to your github repo. Use command line git (not github desktop)
- Submit `ps3_solo.pdf` through the `Problem Set 3 Solo` assignment on Gradescope. (4 points)
- Tag your submission in Gradescope

(applies only to the solo part) "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **___**

problem set setup (5 points)

Learn git branching (15 points)

Go to https://learngitbranching.js.org. This is the best visual git explainer we know of.

1. Complete all the levels of main "Introduction Sequence". Report the commands needed to complete "Git rebase" with one line per command.
2. Complete all the levels of main "Ramping up". Report the commands needed to complete "Reversing changes in git" with one line per command.
3. Complete all the levels of remote "Push & Pull – Git Remotes!". Report the commands needed to complete "Locked Main" with one line per command.

Exercises (25 points)

Now it's time to get your hands dirty! Clone https://github.com/eficode-academy/git-katas.git

Tips:

- These exercises have many steps. Keep a notebook (e.g. `.txt` or note-taking software) with what happens at every step.
- To find out what directory you are in, run `cd` on a PC or `pwd` on a Mac.
- Make sure you're navigating to the correct folder for each exercise.
- **When running git merge or git revert without a commit message, Git opens a VIM text editor by default. To avoid VIM, add the -m "commit message" option to the git merge or git revert command. If you find yourself stuck in VIM, type :q! and click Enter/Return.**

Basic Staging and Branching (5 points)

1. Exercise. For your pset submission, tell us only the answer to the last question (22).
2. Exercise. For your pset submission, tell us only the output to the last question (18).

Merging (10 points)

1. Exercise. After completing all the steps (1 through 12), run git log –oneline –graph –all and report the output.
2. Exercise. Report the answer to step 11.
3. Identify the type of merge used in Q1 and Q2 of this exercise. In words, explain the difference between the two merge types, and describe scenarios where each type would be most appropriate.

Undo, Clean, and Ignore (10 points)

1. Exercise. Report the answer to step 13.
2. Exercise. Look up `git clean` since we haven't seen this before. For context, this example is about cleaning up compiled C code, but the same set of issues apply to random files generated by knitting a document or by compiling in Python. Report the terminal output from step 7.
3. Exercise. Report the answer to 15 ("What does git status say?")

SECTION 2 - Partnered

Setup Instructions

1. **Partnered Part**

- **Partner 1** will **accept the PS3 assignment** through the provided GitHub Classroom link. After accepting, **Partner 1** must share the unique repository link with **Partner 2**.

- This link can only be shared with **one partner**, and **once accepted, the partner cannot be changed**.

- Both partners will collaboratively work by **pushing and pulling changes** to/from the shared repository.

- When you are both ready to submit, run the following command to generate a **`.txt` file** of your commit history:

```
git log --pretty=format:"%h - %an, %ar : %s" > commit_history.txt
```

- Since Gradescope does not allow `.txt` file uploads, please **convert the text file to a PDF** and submit it on Gradescope.

- **Before converting and turning in your PDF**, please include the following information at the start of the text file or just before the commit history:

```
Partner1: Partner1_Name, Partner1_GitHub_Username
Partner2: Partner2_Name, Partner2_GitHub_Username
```

- **Both partners must contribute to the commit history**. If there are no commits from a partner, that partner will receive **no credit** for the paired section.

2. **Gradescope Submission Process**:

- Partners will submit **as a team**, meaning only one partner's submission will count for both. Please select the **same partner** you worked with on Gradescope when submitting.

- Submit the converted PDF through the `Problem Set 3 Paired` assignment on Gradescope.

Git merge conflicts (10 pts)

For this problem set, please refer to the Instructions provided for the section.

You will need to turn in a `pdf` as part of this section. Please include both partner names as in the example solution provided.

You will be graded based on the commit history from doing the steps described below.

Prelude

i. Play paper, scissors, rock to determine who goes first. Call that person *Partner 1*.

First round of practice

i. a. *Partner 1*, Start a branch called `merge_conflict_name_1`. In `ps3_pair.qmd` replace “Dema Therese” with your name. Push your branch to github remote
   b. *Partner 2*, Start a branch called `merge_conflict_name_2`. In `ps3_pair.qmd` replace “Dema Therese” with your name. Push your branch to github remote

ii. *Partner 1* screen share and make a pull request on github.com.

iii. *Partner 2* screen share on github review the pull request. Accept your partners changes and merge the branch into `main`. Hooray! This is your first successful pull request!

iv. *Partner 2* make a pull request.

v. *Partner 1* screen share. On github.com review the pull request. There should be a merge conflict because you both changed the same line of the file. Adjust the file and then complete the merge.

Second round of practice

i. a. *Partner 2*, Start a branch called `describe`. In `ps3_pair.qmd`, modify the function so that it returns a list where the first object is the material printed as the head and the second object is the results from running `describe` on the data frame. Push your branch to github remote.
   b. *Partner 1*, Start a branch called `histogram`. In `ps3_pair.qmd`, modify the function so that it returns a list where the first object is the material printed as the head and the second object is an altair plot with a histogram of the values. Push your branch to github remote.

ii. *Partner 2* screen share and make a pull request on github.com.

iii. *Partner 1* screen share on github review the pull request. Accept your partners changes and merge the branch into `main`. Hooray! This is your first successful pull request!

iv. *Partner 1* make a pull request.

v. *Partner 2* screen share. On github.com review the pull request. There should be a merge conflict because you both changed the same line of the file. Rewrite the function so that it returns a list with three objects (`head`, `describe`, and `histogram`) and complete the merge.