30538 Problem Set 1: git

Ruilin Yao

2026-01-05

Due Sat Jan 10 at 5:00PM Central.

Github Classroom Assignment Setup and Submission Instructions

This problem set has two parts: a **Solo** section and a **Partnered** section. Please read the instructions carefully to ensure you follow the correct workflow for each part.

1. **Accepting and Setting up the PS1 Assignment Repository**
   - Before you begin this problem set, you will need to have set up everything listed in the welcome email:
     - Install git locally https://github.com/git-guides/install-git
     - Create Github account linked to your UChicago email and set up 2-factor authentication
     - Install Github CLI and link your Github account to your computer https://cli.github.com/
   - Each student must individually accept the repository for the problem set from Github Classroom ("ps1-solo") – https://classroom.github.com/a/mz25yNBL
     - You will be prompted to select your cnetid from the list in order to link your Github account to your cnetid.
     - If you can't find your cnetid in the link above, click "continue to next step" and accept the assignment, then add your name, cnetid, and Github account to this Google Sheet and we will manually link it: https://rb.gy/9u7fb6
   - If you authenticated and linked your Github account to your device, you should be able to clone your PS1 assignment repository locally.
   - Contents of PS1 assignment repository:
     - `ps1_template.qmd`: this is the Quarto file with the template for the problem set. You will write your answers to the problem set here.

    - `ps1_exercise_PAIR.qmd`: this will be used in the paired portion of the problem set (see instructions for Section 2 below).

2. **Submission Process**:
    - Knit your completed solution `ps1.qmd` as a pdf `ps1.pdf`.
        - Your submission does not need runnable code. Instead, you will tell us either what code you ran or what output you got.
    - To submit, push `ps1.qmd` and `ps1.pdf` to your PS1 assignment repository. Confirm on Github.com that your work was successfully pushed.

Grading

- You will be graded on what was last pushed to your PS1 assignment repository before the assignment deadline
- Problem sets will be graded for completion as: {missing (0%); - (incomplete, 50%); (complete, 90%); + (excellent, 100%)}
    - The percent values assigned to each problem denote how long we estimate the problem will take as a share of total time spent on the problem set, not the points they are associated with.
- In order for your submission to be considered complete, you need to push both your .qmd and the compiled PDF to your repository. Submissions that do not include both files will automatically receive 50% credit.

SECTION 1 - Solo

Learn git branching (30%)

Go to https://learngitbranching.js.org. This is the best visual git explainer we know of. The exercises go beyond what we covered in lecture. This is intentional.

1. Complete all the levels of main "Introduction Sequence". Report the commands needed to complete "Git rebase" with one line per command.

git branch bugFix

git checkout bugFix

git commit

git checkout main

git commit

git checkout bugFix

git rebase main

2. Complete all the levels of main “Ramping up”. Report the commands needed to complete “Reversing changes in git” with one line per command.

git reset HEAD^

git checkout pushed

git revert pushed

3. Complete all the levels of remote “Push & Pull – Git Remotes!”. Report the commands needed to complete “Locked Main” with one line per command.

git branch feature

git branch -f main HEAD^

git checkout feature

git push

Exercises (50%)

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.
- Review/rewatch the minilesson tips on using `vim`

Basic Staging and Branching (10%)

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

On branch master nothing to commit, working tree clean

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

```
diff –git a/file1.txt b/file1.txt
deleted file mode 100644
index 2447721..0000000
— a/file1.txt
+++ /dev/null
@@ -1 +0,0 @@
-ruilin yao
diff –git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..e69de29
```

Merging (20%)

1. Exercise. After completing all the steps (1 through 12), run git log –oneline –graph –all and report the output.

- 2933978 (HEAD -> master, feature/uppercase) Upper case
- 1cba2c1 Add content to greeting.txt
- 292f3eb Add file greeting.txt

2. Exercise. Report the answer to step 11.

- 4cf71f4 (HEAD -> master) merge

|

* 4c4b945 (greeting) GBY'

- 54a96b2 read

|/

- 0d140da Add content to greeting.txt
- 82f8aee Add file greeting.txt

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.

Q1 uses fast-forward merge, where Git directly moves the pointer to the latest commit of the feature branch when the main branch has no new commits, maintaining a linear history. Q2 uses three-way merge, where Git creates a new merge commit to integrate changes when both branches have new commits, preserving the complete branch history. Fast-forward merge is suitable for simple personal development or short-term branches, keeping history concise; three-way merge is more appropriate for team collaboration or long-term feature branches, clearly recording merge points and branch lifecycles.

Undo, Clean, and Ignore (20%)

1. Exercise. Report the answer to step 13.

commit df745cdeb0255fdd30b145a09fa44b7b99b9db1e

Author: git-katas trainer bot git-katas@example.com

Date: Thu Jan 8 21:42:04 2026 -0600

`Add credentials to repository`

diff –git a/credentials.txt b/credentials.txt

new file mode 100644

index 0000000..8995708

— /dev/null

+++ b/credentials.txt

@@ -0,0 +1 @@

+supersecretpassword

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.

Removing README.txt~

Removing obj/

Removing src/myapp.c~

Removing src/oldfile.c~

3. Exercise. Report the answer to 15 ("What does git status say?")

On branch master

Changes to be committed:

(use "git restore –staged ..." to unstage)

```
    deleted:    file1.txt
```

Changes not staged for commit:

(use "git add ..." to update what will be committed)

(use "git restore ..." to discard changes in working directory)

```
    modified:   .gitignore
```

Untracked files:

(use "git add ..." to include in what will be committed)

```
    file2.txt

    file3.txt
```

SECTION 2 - Partnered

Partnered Section Setup and Submissions Instructions

1. **Setup Instructions**
   - Find a partner who is also taking 30538 (they do not need to be in your section). If you do not know any student in the class, please try to match with a partner using this google sheet. Play paper, scissors, rock to determine who goes first. Call that person **Partner 1**.

- The paired team should work on the repository they accepted through Github Classroom ("ps1_git_pair") – (https://classroom.github.com/a/t0k2uNCd)
  - **Partner 1** should invite **Partner 2** as a contributor to this repository
  - The visibility of this repository should be set to public, so that the graders can view it as well
  - Copy the `ps1_exercise_PAIR.qmd` file from the PS1 assignment repository into this shared repository
- Both partners will then collaboratively work by **pushing and pulling changes** to/from the shared repository.

2. **Submission Process**
   - You will be graded based on the commit history in your shared repo from doing the steps described below

---

Git merge conflicts (20%)

First round of practice

i.
   a. *Partner 1*, Start a branch called `merge_conflict_name_1`. In `ps1_exercise_PAIR.qmd` replace "My Name" with your name. Push your branch to github remote
   b. *Partner 2*, Start a branch called `merge_conflict_name_2`. In `ps1_exercise_PAIR.qmd` replace "My Name" 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 `ps1_exercise_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 `ps1_exercise_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`.
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.