Problem Set #4

Grade: /33

Overview

One of the goals for this quarter is to get you comfortable using git and GitHub. In this problem set we will be practicing more git/GitHub workflow basics, manipulating data in R, and creating GitHub issues. We are asking you to create a git repository on your local computer which you will later connect to a remote repository on GitHub. This local repository will have an `.R` file where you will read in data and practice manipulating this data. We recommend doing the reading (e.g. lecture and other required readings) prior to completing the problem set.

Note: In PS3, you created the remote GitHub repo first and then cloned it to your local machine/computer. In this PS, we will create a repo on our local machine/computer first and then connect it to the remote repository. See section 4.2 of the Git and GitHub lecture.

Part I: Concepts & Definitions

/0.5

1. What hidden directory is created whenever a git repository is created?

   ANSWER: The hidden directory is .git/ which holds internal data structure required for version control

/1

2. Describe what git objects are, what they are identified by, and where they are stored.

   ANSWER: An object is any file or directory in the git repository. They are identified by a unique hash - a series of letters and numbers that points to the object. Git objects are stored inside the .git/objects directory.

/1.5

3. List the 4 types of git objects and define each of them.

   ANSWER: Blob - a file that stores data, Tree - a directory that references blobs or trees, Commit - created after a commit occurs and contains information about the commit, Tag - reference to a point in git history, tag object is created after generating a tag

Part II: Exploring git objects

/1

1. You'll create a new **RStudio project** for this problem set:
   - Open up **RStudio** and on the top right corner of the window, select `New Project` under the dropdown menu
   - Select `New Directory` then `New Project` to create a new project
   - Name your new project directory `ps4_lastname_firstname` (fill in your name), browse where you want to create the project (e.g., `Downloads` or `Documents` folder, etc.), and click `Create Project`
   - Move this `problemset4.Rmd` you're working on into your newly created `ps4_lastname_firstname` directory

/1

2. In your **RStudio Terminal**, run the command that will turn your `ps4_lastname_firstname` directory into a git repository and write the command you used below:

```
# Command to initialize git repository
git init
```

/1.5

3. Use the `echo` command to output the text `"# YOUR NAME HERE"` (fill in your name) and redirect it using `>` to a file called `problemset4.R`. Then, print the contents of `problemset4.R` to the terminal screen. Write the commands you used here:

```
# Command to redirect text into ‘problemset4.R‘
echo "#Brittany Kaplan" > problemset4.R

# Command to print contents of ‘problemset4.R‘
cat problemset4.R
```

/1.5

4. Add `problemset4.R` to the staging area. Then, use a git command to compute the hash ID for `problemset4.R`. Write the commands you used and paste the output you see below:

```
# Command to add ‘problemset4.R‘
git add problemset4.R

# Command to find hash ID for ‘problemset4.R‘
find .git/objects -print | sed -e 's;[^/]*/;|____;g;s;____|; |;g'

# Output
|____objects
| |____bb
| | |____6121fd97505eb22094bd8719ac960186cd5244
| |____pack
| |____info
```

/2

5. Use the hash you obtained in the previous step to find the content, type, and size of the blob object. Write the commands you used and the outputs you got here:

```
# Command to find content of the blob object
git cat-file -p bb6121f

# Output
#Brittany Kaplan

# Command to find type of the blob object
git cat-file -t bb6121f

# Output
blob

# Command to find size of the blob object
git cat-file -s bb6121f

# Output
17
```

/1

6. Commit the file and check the commit log to obtain the hash ID of the commit.
Tip: You can exit the log window back to your terminal by pressing `q` or `ctrl+z`.

```
# Command to commit ‘problemset4.R‘
git commit -m "Commit problemset4.R"

# Command to check the commit log
git log

# What is the hash ID of your commit?
1b1d54f6ba00e2df6a7ecbaccf41d2d8da582521
```

/2

7. Use the hash you obtained in the previous step to find the content, type, and size of the commit object. Write the commands you used and the outputs you got here:

```
# Command to find content of the commit object
git cat-file -p 1b1d54f

# Output
tree 43a4f636f25196ad651cf55ba9202fdf420d41ed
author Brittany Kaplan <bkaplan@MacBook-Pro-70.local> 1675645515 -0800
committer Brittany Kaplan <bkaplan@MacBook-Pro-70.local> 1675645515 -0800

Commit problemset4.R
```

```
# Command to find type of the commit object
git cat-file -t 1b1d54f

# Output
commit

# Command to find size of the commit object
git cat-file -s 1b1d54f

# Output
213
```

Part III: Manipulating data in R

/0.5

1. Open up your `problemset4.R` file in **RStudio** and erase the comment containing your name at the top of the file. Replace it with a line to import the `tidyverse` library.

echo “library(tidyverse)” > problemset4.R

/1

2. Next, you’ll be reading in some CSV data directly from the web (i.e., without downloading it first). Navigate to the **Equality of Opportunity Project** data page here and scroll to the **Mobility Report Cards: The Role of Colleges in Intergenerational Mobility** section. Expand the data section and copy the URL for the **Excel** link of **Online Data Table 2**. (*Note*: This is actually a CSV file, not Excel)

   In your `problemset4.R` file, use an R function to directly read in the CSV data using the URL and store it in an object called `mrc`. echo “mrc <- read_csv(file = ‘http://www.equality-of-opportunity.org/data/college/mrc_table2.csv’)” » problemset4.R

/0.5

3. Create a new object called `mrc_subset` based on `mrc` that contains only the following variables: `name, par_q1, par_q2, par_q3, par_q4, par_q5`. Refer to the codebook to see what these variables mean. echo “mrc_subset <- subset(mrc, select = c(name, par_q1, par_q2, par_q3, par_q4, par_q5))” » problemset4.R

/2

4. Create another object called `mrc_pivot` based on `mrc_subset` that pivots the table from wide to long such that:

   - There are 3 columns in the pivoted table called: `name`, `quintile`, `fraction`
   - The `name` column should just be the same `name` variable from the original `mrc_subset` table
   - The `quintile` column should contain the following values obtained from the column names of `mrc_subset`: `q1`, `q2`, `q3`, `q4`, `q5` (*Note that we want to drop the `par_` prefix*)
   - The `fraction` column should contain the corresponding values for each quintile

echo “mrc_pivot<- mrc_subset %>% pivot_longer(cols = ‘par_q1’:‘par_q5’, names_to = ‘quantile’, values_to = ‘fraction’)” » problemset4.R

/1

5. Add and commit your `problemset4.R` file. git add problemset4.R git commit -m “Committing changes in problemset4.R”

/0.5

6. In your terminal, enter `git ls-tree -rt HEAD` which shows the contents of the .git directory. Then write `git cat-file -p <hash>`. to print the contents of problemset4.R using the hash we found from the first command. Paste the output below:

```
# Git command you used to see the contents of problemset4.R
git cat-file -p 876ff50f6173f553c7199e0a3311f650a75d8f5a

# The output of git cat-file
library(tidyverse)
mrc <- read_csv(file = 'http://www.equality-of-opportunity.org/data/college/mrc_table2.csv')
mrc_subset <- subset(mrc, select = c(name, par_q1, par_q2, par_q3, par_q4, par_q5))
mrc_pivot<- mrc_subset %>%
  pivot_longer(cols = 'par_q1':'par_q5',
               names_to = 'quantile',
               values_to = 'fraction')
```

/0.5

7. Go back to question 2.4 and copy the hash ID you obtained for problemset4.R back then. Type `git cat-file -p <hash>` using that previous hash and paste the output below:

```
# Git command to see contents of hash ID from question 2.4
git cat-file -p bb6121fd97505eb22094bd8719ac960186cd5244

# The output of git cat-file
#Brittany Kaplan
```

/1

8. In your own words, explain why the output for git cat-file was different in questions six and seven, even though they pointed to the same `problemset4.R` file.

   ANSWER: The output of git cat-file was different because they pointed to different hashes (different points in time that problemset4.R was saved).

Part IV: Practice with git

/1

1. Let’s make some more changes to `problemset4.R`. Modify `problemset4.R` by adding a comment at the end of your file where you write down your guilty pleasure. Save your changes but don’t stage the problem set script to git yet.

echo “#My guilty pleasure is eating gummy worms” » problemset4.R

```
On the terminal, run ‘cat problemset4.R‘. You decide you want to discard this change. Write the git com
```

```
# Git command to discard your changes to problemset4.R
git restore problemset4.R
```

Check that your guilty pleasure has been deleted by executing `cat problemset4.R` once again.

/2

2. You decide you *must* share your guilty pleasure. Edit your `problemset4.R` script once again with your guilty pleasure in a comment at the end of the script. Add the file to the staging area.

echo "#My guilty pleasure is eating gummy worms" » problemset4.R

```
Now let's say you are having second thoughts about committing this change. What command would you use t
```

```
(git add problemset4.R)

# Command to use to unstage 'problemset4.R' after you've added it
git reset HEAD problemset4.R
```

/2

3. But in the end, you decide to go ahead and commit this change anyway. Re-add `problemset4.R` to the staging area and make a commit with the message `my guilty pleasure`.

   You regret it instantly! You remember that `git reset` and `git revert` are two commands to undo changes from a commit. What is the difference between them?

   ANSWER: git reset is for when you stage changes and you no longer wish to stage them. git revert undoes previous commits but keeps a record of those commits.

/1

4. Let's say you decided to use `git revert`. Revert the `my guilty pleasure` commit and write the command you used below. (*Hint*: Include the `--no-edit` option to use default commit message)

```
# Command to revert your 'my guilty pleasure' commit
git revert --no-edit 876ff50f6173f553c7199e0a3311f650a75d8f5a
```

/2

5. Now, head over to GitHub in your browser and create a new private repository in the `anyone-can-cook` organization here. Name your repo `ps4_lastname_firstname` (fill in your name) and do **NOT** initialize it with a `README.md` or `.gitignore` file.

/3

6. Connect your local `ps4_lastname_firstname` repository to the remote and push your changes. (*Hint*: Refer to Section 4.2 in the lecture)

```
# Command to add remote repository
git remote add origin https://github.com/anyone-can-cook/ps4_kaplan_brittany.git
git add ps4_kaplan_brittany.Rmd
git commit -m "add problemset4.Rmd"

# Command to push to remote
git branch -M main
git push -u origin main
```

Part V: Create a GitHub issue

/2

- Go to the class repository and create a new issue.
- Please refer to rclass2 student issues readme for instructions on how to post questions or things you've learned.
- You can either:
  - Ask a question that you have about this problem set or the course in general. Make sure to assign the instructors (@ozanj, @xochilthlopez, @joycehnguy, @augias) and mention your team (e.g., @anyone-can-cook/your_team_name).
  - Share something you learned from this problem set or the course. Please mention your team (e.g., @anyone-can-cook/your_team_name).
- You are also required to respond to at least one issue posted by another student.
- Paste the url to your issue here: https://github.com/anyone-can-cook/rclass2_student_issues_w23/issues/193
- Paste the url to the issue you responded to here:https://github.com/anyone-can-cook/rclass2_student_issues_w23/issues/183

Knit to pdf and submit problem set

Knit to pdf by clicking the "Knit" button near the top of your RStudio window (icon with blue yarn ball) or drop down and select "Knit to PDF"

You will submit this problem set by pushing it to your repository. Make sure to push both the `.Rmd` and `.pdf` files.