Problem Set #4

Grade: /30

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.

Part I: Concepts & Definitions

/0.5

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

   ANSWER: The hidden directory that is hidden whenever a git repository is created is '.git'.

/1

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

   ANSWER: Git objects are contents that can be addressed through a filesystema and is a simple data store system. There are 4 types of git objects which include: blob; tree, commit, and tag. Git objects are identified by a "hash", which is like a unique ID that points to the git object. Git objects are stored in the '.git/objects' directory and the first 2 characters of its hash are in the name of the sub-directory within '.git/objects' that it is located in.

/1.5

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

   ANSWER: The 4 types of git objects are:

   1) Blob: A file that stores data, in which the file must be added to the staging area to be created. The hash of the blob object is visible in the '.git/objects' directory or by using the 'git hash-object' command.
   2) Tree: A directory that contains references to blobs (also known as files) or other trees (also known as sub-directories). A sub-directory that is created inside the git repository is a tree object. The root directory of the git repository is also a tree and to create the tree object you must commit.
   3) Commit: A commit object is created after you commit and it contains information about the commit.
   4) Tag: The tag object is created after you generate 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 "# Paula Nazario" > problemset4.R
## echo "# Paula Nazario" >> problemset4.R (appends)
## echo "# Paula Nazario" > problemset4.R (overwrites)

# 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‘
git log
git hash-object problemset4.R

# Output
## fa721f14ef29289e1d0f87f1b3d80a2356d8fa2e
```

/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 fa721f1

# Output
# Paula Nazario

# Command to find type of the blob object
git cat-file -t fa721f1

# Output
blob

# Command to find size of the blob object
git cat-file -s fa721f1

# Output
16
```

/1

6. Commit the file and check the commit log to obtain the hash ID of the commit:

```
# 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?
8b144610f9ae38ccdfd82b1be53dd391c31d3100
```

/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 8b14461

# Output
tree d21bdd71e758997fcc49112f5ce26aafdfd87c80
author Paula Nazario <paulanazario@Paulas-MacBook-Pro.local> 1643268180 -0800
committer Paula Nazario <paulanazario@Paulas-MacBook-Pro.local> 1643268180 -0800

commit problemset4.R

# Command to find type of the commit object
git cat-file -t 8b14461
```

```
# Output
commit

# Command to find size of the commit object
git cat-file -s 8b14461

# Output
227
```

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.

/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`.

/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.

/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

/1

5. Add and commit your `problemset4.R` file.

Part IV: Practice with git

/2

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. Add the file to the staging area.

   Now let's say you are having second thoughts about committing this change. What command would you use to unstage this file?

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

/2

2. 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: The difference between `git reset` and `git revert` is that the `git reset' function will reset the current HEAD to the specified state, which means that it will unstage the specified file from the staging area to the working directory. The`git revert‘ function will revert the existing commit(s), which means that the it will revert all commits up to and inlcude the specified commit_hash.

/1

3. 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 37b5c3e
git log
```

/2

4. 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

5. 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_nazario_paula.git
git branch -M main

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

Part V: Create a GitHub issue

/2

- Go to the class repository and create a new issue.
- 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, @lizachavac, @briannawright135) 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_w22_student_issues/issues/133
- Paste the url to the issue you responded to here: https://github.com/anyone-can-cook/rclass2_w22_student_issues/issues/126

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.

Commands git add problemset4.Rmd problemset4.pdf git commit -m "Rmd and PDF" git push