

Challenge 8 Fall23: Instruction

Introduction

GitHub is one of the most popular platforms and cloud-based services for code-storage and management, software development, and version control. In this challenge, we will practice setting up GitHub, creating a R project with GitHub version control, and syncing an existing R project with a GitHub repository, allowing us to track changes, reverse codes, and share your R project with others.

For the detailed instruction, please review the class meeting videos on Dec 4. If you have any questions, you can also refer to the reading *Happy Git and GitHub for the useR*. It is a comprehensive user manual for using R with GitHub, but some chapters are too complicated.



Step 1: Ensure you have the latest version of R and RStudio. If not, update them.

Step 2: Set up a GitHub account (if you have not).

Happy Git and GitHub for the useR: Chapter 4

Step 4: Install Git on your local device.

Happy Git and GitHub for the useR: Chapter 6

(Optional for this challenge, but recommended for your future usage: install a git client; check Happy Git and Github for the useR: Chapter 8)

Step 5: Set up a Personal Access Token (PAT) for HTTPS

If you have not set up a personal access token, type the following code in RStudio:

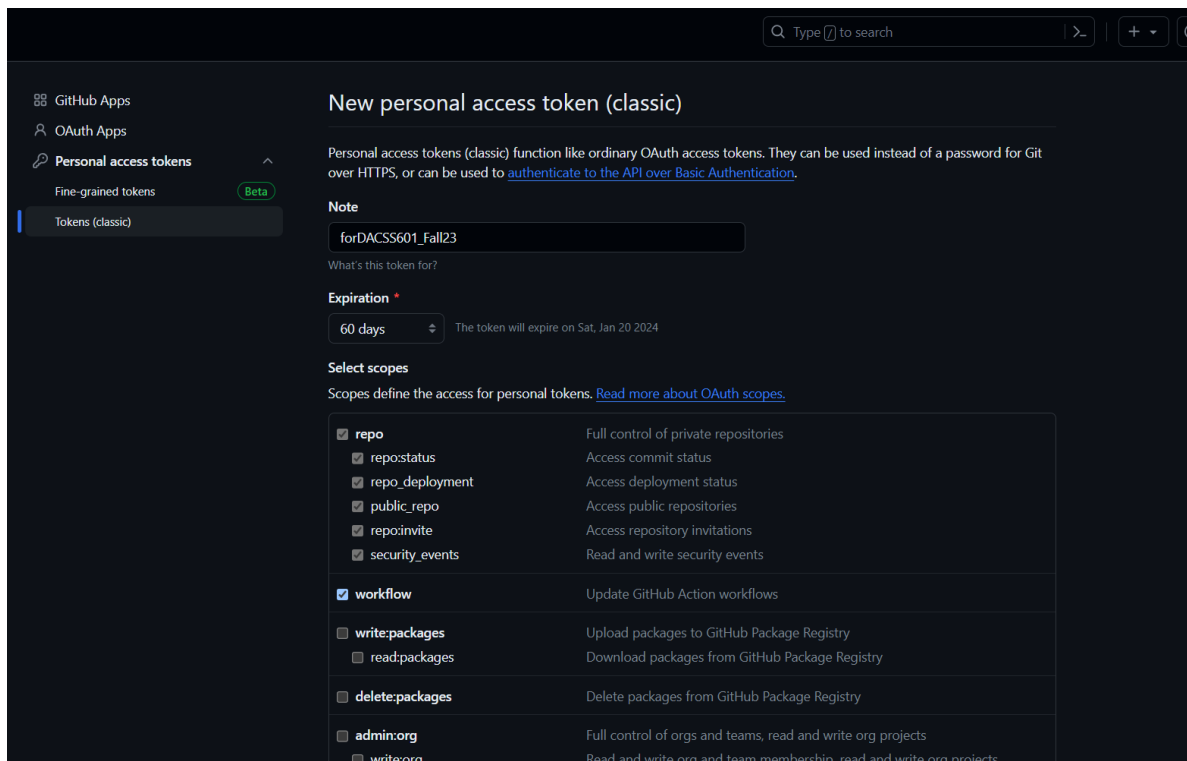
```
#Please install the package "usethis" first, if you have not installed it previously.  
  
library(usethis)  
  
usethis::create_github_token()
```

- * Call ``gitcreds::gitcreds_set()`` to register this token in the local Git credential store
- It is also a great idea to store this token in any password-management software that you use
- * Open URL `'https://github.com/settings/tokens/new?scopes=repo,user,gist,workflow&description'`

The usethis approach takes you to a pre-filled form where we have pre-selected some recommended scopes, which you can review and adjust before clicking “Generate token” (see screenshot below). At the time of writing, the usethis-recommended scopes are “repo”, “user”, “gist”, and “workflow”.

After setting up your PAT, you can type the following code in RStudio (make sure you delete # before running the following codes in RStudio). Copy and paste the PAT (see screenshot below) to RStudio after the pops-up message asks you for a password/token. **Please find a place to store this PAT because you will use it everytime you sync (push) to Github!**

```
#Please install the package "gitcreds" first, if you have not installed it previously.
```



```
#library(gitcreds) #works only with the latest R version 4.3.2

#gitcreds_set()
```

If you have already set up a personal access token, you can directly use the above code in RStudio.

Task#1. Create a New R Project and Connect it to a GitHub Repository.

Now, you should be ready to connect R with GitHub. Let's first try to create a new blank R project that allows GitHub to do version control.

1. First, you should create a GitHub repository on your own GitHub page. We walk through this in the class meeting on Dec 4. You can also check out the instructions on Happy Git and GitHub for the useR: Chapter 11.1 or 15.1.


Make sure you make this repository as "Public" so that I view it on your GitHub page for grading.

Personal access tokens (classic)

[Generate new token](#)[Revoke all](#)

Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_SrtZqPrrWqFtXIBujXXCdQ2tH2ACIQ0wWzoE 

[Delete](#)

EricoNew — gist, repo, user, workflow

Last used within the last 10 months

[Delete](#)

⚠ This token has no expiration date.

EricoYuDACSS601 — gist, repo, user, workflow

Last used within the last 10 months

[Delete](#)

⚠ This token has no expiration date.

git: <https://github.com/> on ERICO-THINKPAD at 13-Feb-2019 18:03 — gist, repo, workflow

Last used within the last 7 months

[Delete](#)

🔄 Regenerate this token to take advantage of the [new token formats](#)

⚠ This token has no expiration date.

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).


Repository template

No template

Start your repository with a template repository's contents.

Owner *

Repository name *

 dongericoyu

/

Great repository names are short and memorable. Need inspiration? How about [refactored-train](#) ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

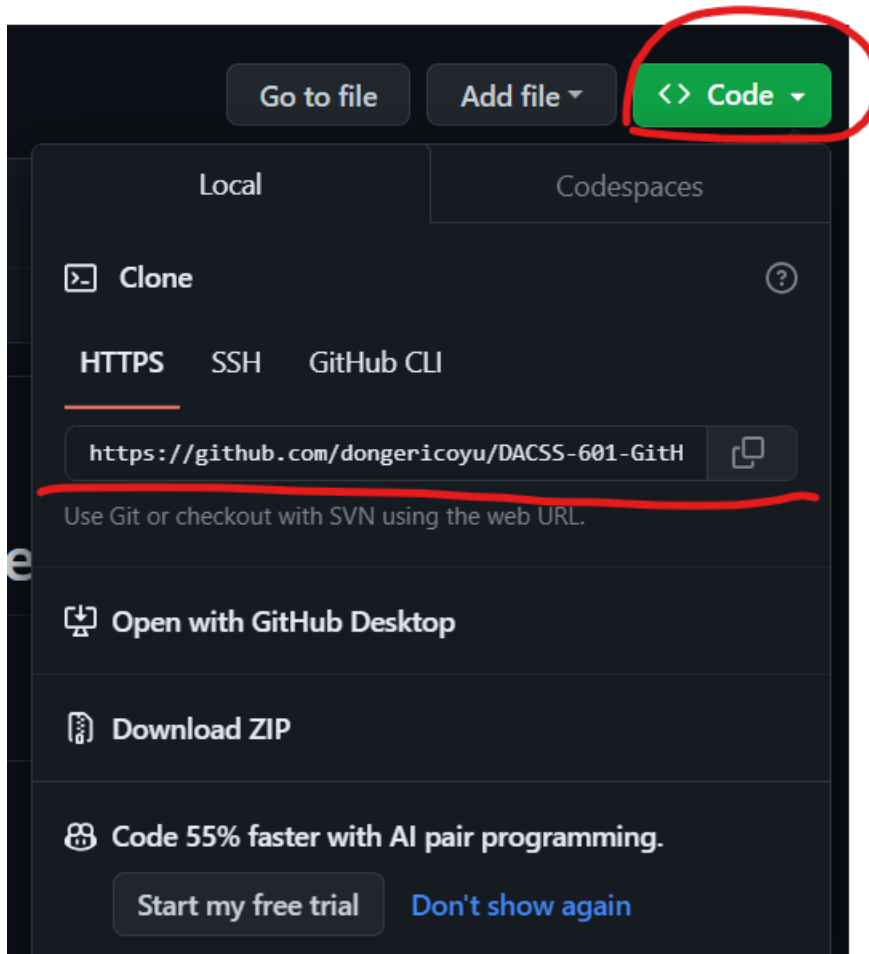
Add .gitignore

.gitignore template: None

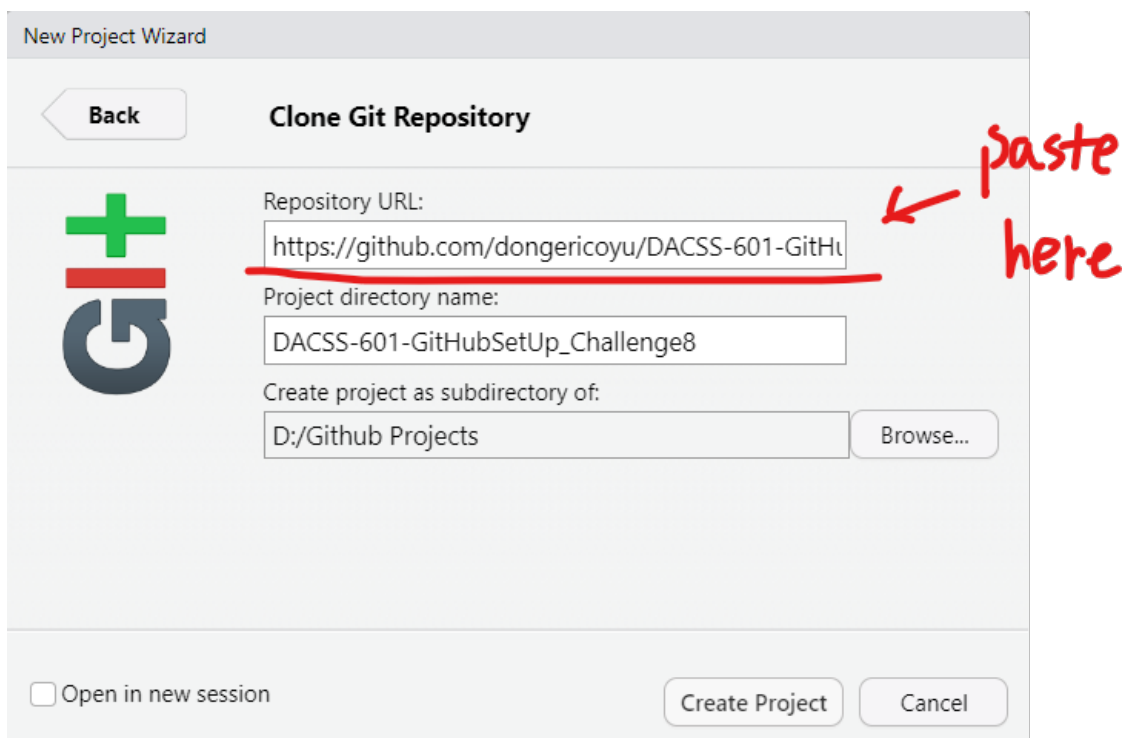
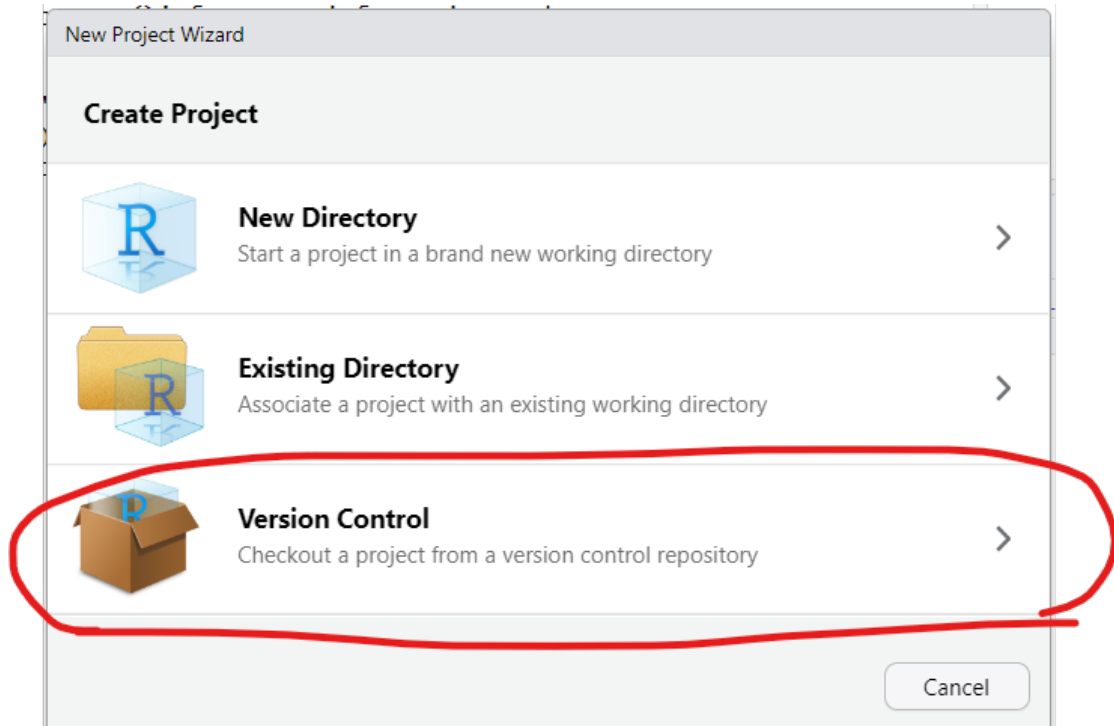
Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

2. After we create a repository, click the big green button that says “<> Code”. Copy a clone URL to your clipboard. Make sure you select and copy the HTTPS URL.



3. Now, go to the tool bar panel of your RStudio: File -> New Project. In the New Project Wizard, select Version Control->Git. Copy and paste the HTTPS URL to the “Repository URL” box, then create the project.

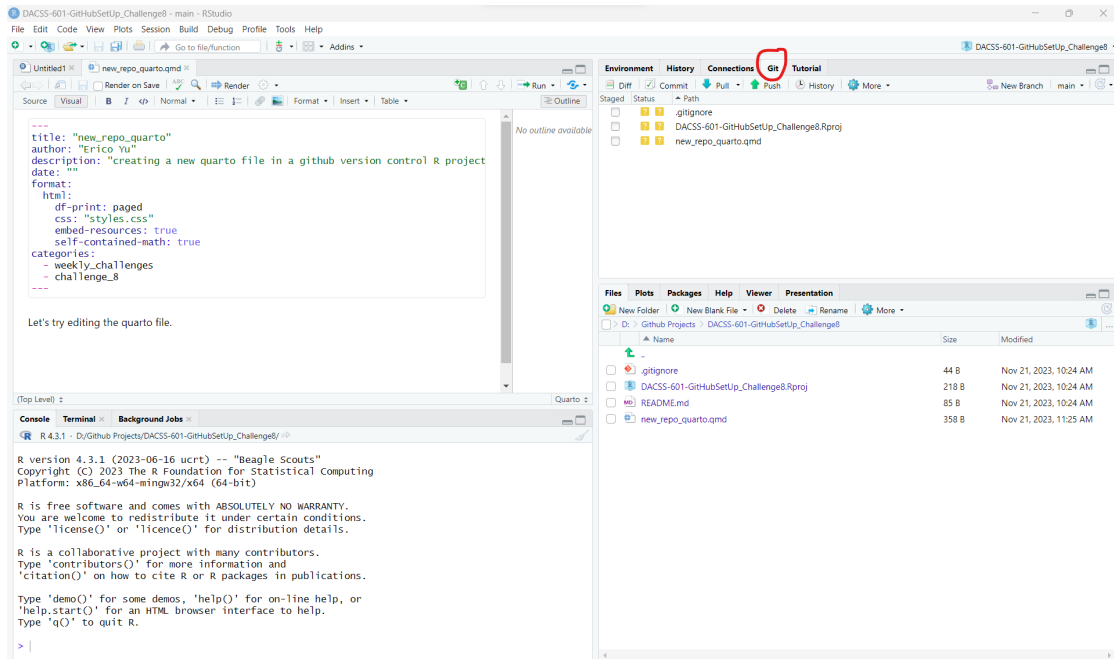


4. Now, you have created an R project that can be synced with GitHub. Let's learn how

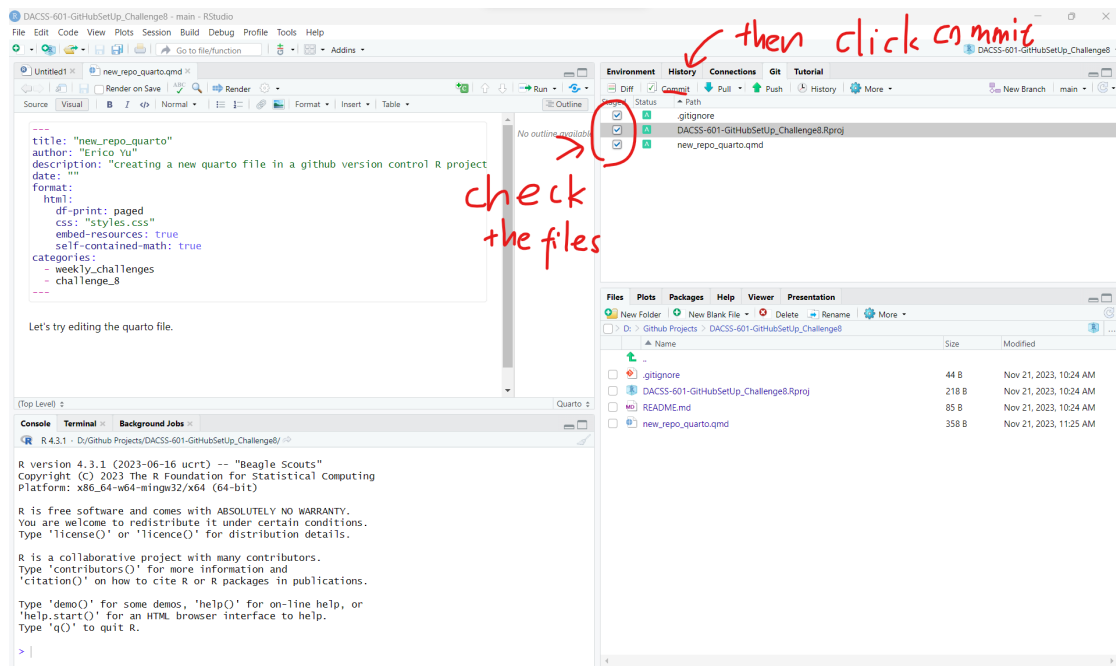
to use the version control function.

First, create a new .qmd file, and type something to answer the following question: “what was the last movie you watched in a theater?” Save it in the R project directory (in the same folder as your R project file).

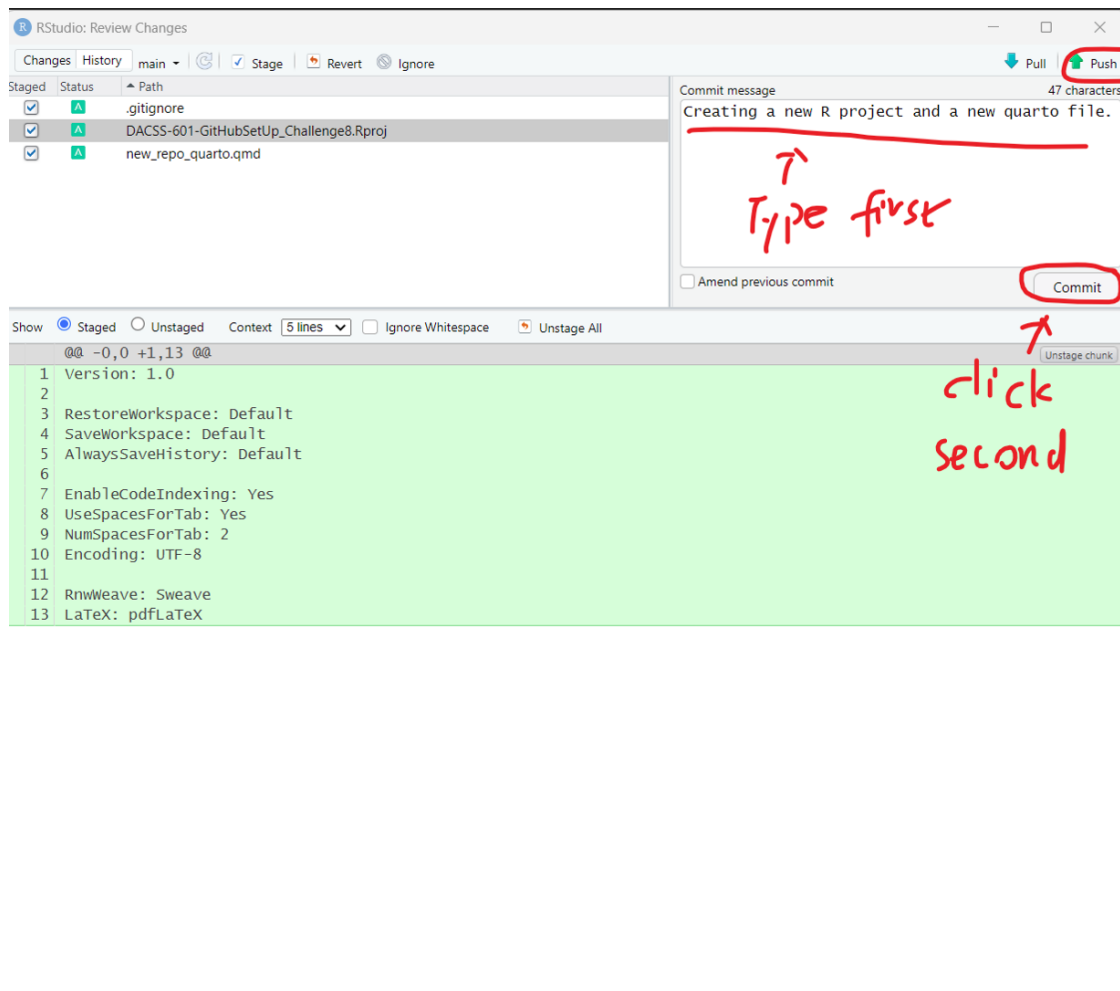
Next, follow the steps shown in the screenshots. Select “Git” on the toolbar of the Environment/History/Connection Panel.



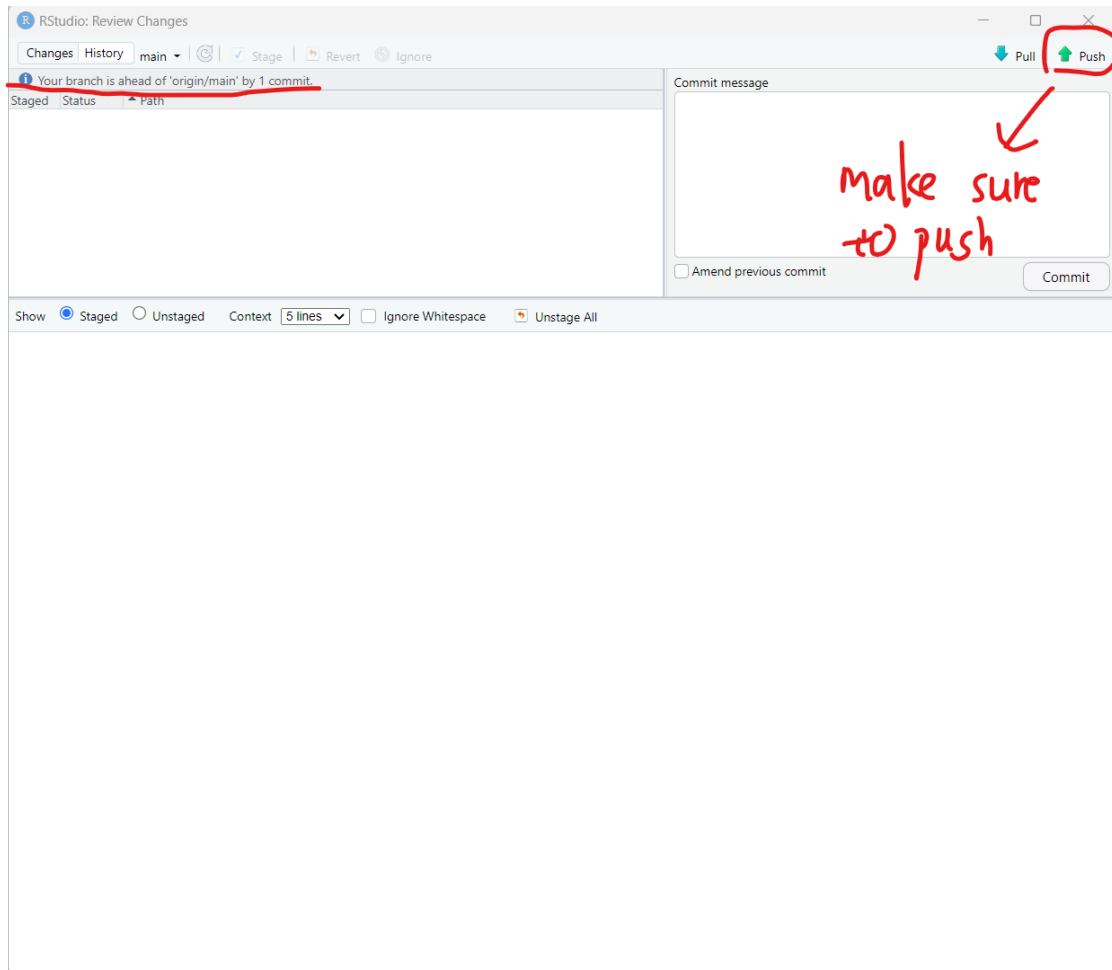
You can see the files we created/changed in this Git window. Check them all and click “Commit”.



After you click “commit”, a window will pop up. You can review the changes you did with the selected files. You can also type something to remind what you did with the codes and project. After editing, click “Commit”.

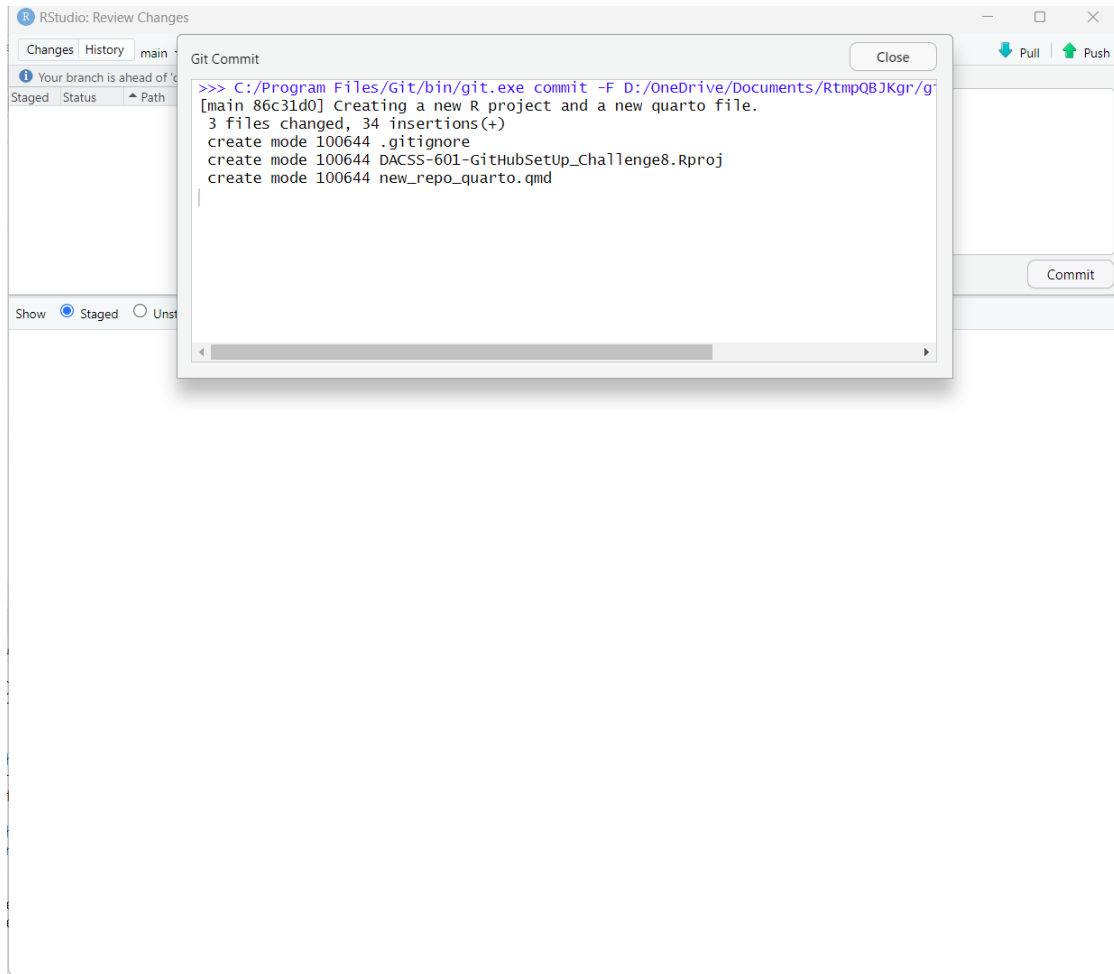


After you hit “commit”, a message will pop up: “Your branch is ahead of the”origin/main” by one commit”. This indicates that your local Git confirms the changes you make to the project in R and recognizes there is a difference between the R project on your local device and the repository stored in the GitHub cloud. This means you need to synchronize, or “push”, your changes to GitHub



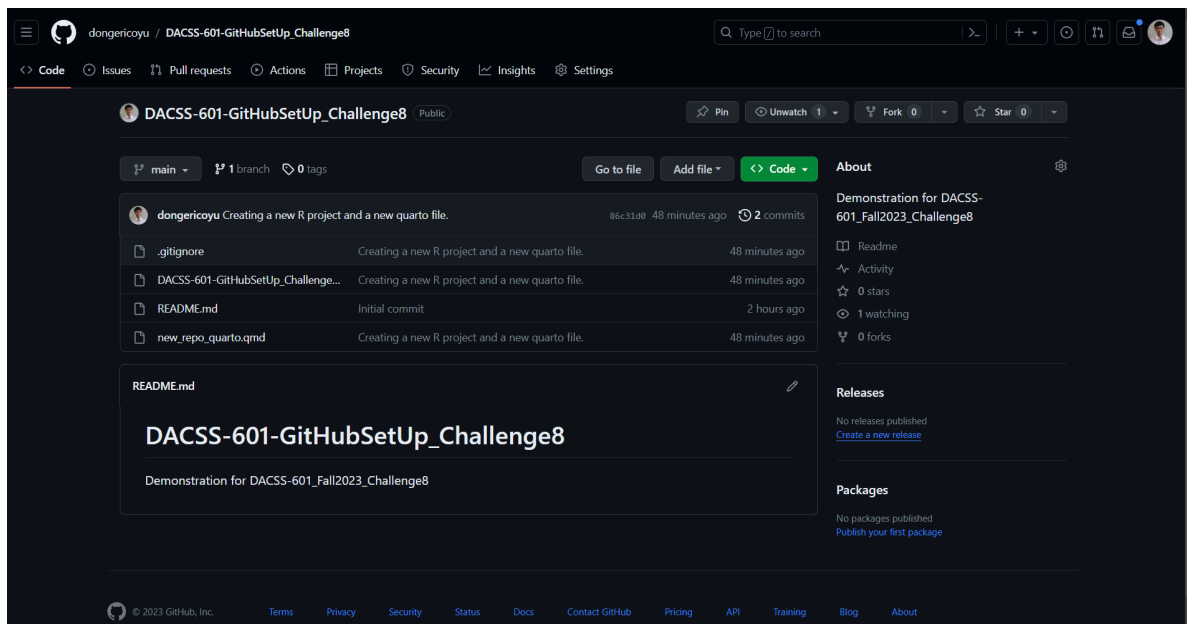
There will be a window pops up to ask you to enter your GitHub username, and copy and paste the PAT you created (not the password for logging in GitHub).

Once it is committed, you can see the message confirming the changes are pushed to GitHub.



A side note: Why it is so complicated: In the past, people could sync (push and pull) easily by just entering the password for their own GitHub accounts after setting up GitHub in RStudio once. However, due to increasing security breaches and concerns, GitHub enforced the adoption of PAT after 2020. The PAT only appears when you first create it on the website, and you must store it elsewhere. If you forget it and need to regenerate a new one, GitHub will send you an email to notify you of the change of the PAT (so that it can avoid your account being hacked and used by an unauthorized party).

5. Now, open your GitHub account page; you should be able to see the repository is changed and updated.



Task#2. Edit the quarto file and push the changes to GitHub.

Now, let's practice "push" one more time. In your RStudio, open the .qmd file you created and type something to answer the following question.

"What did you eat for today's lunch?"

Repeat Steps #4 and #5 in Task 1: save the .qmd file, commit the change, and push the change to GitHub again. When you check on your repository on GitHub.com, you should see the latest commit is updated.

Task#3. Revert a change in R Studio.

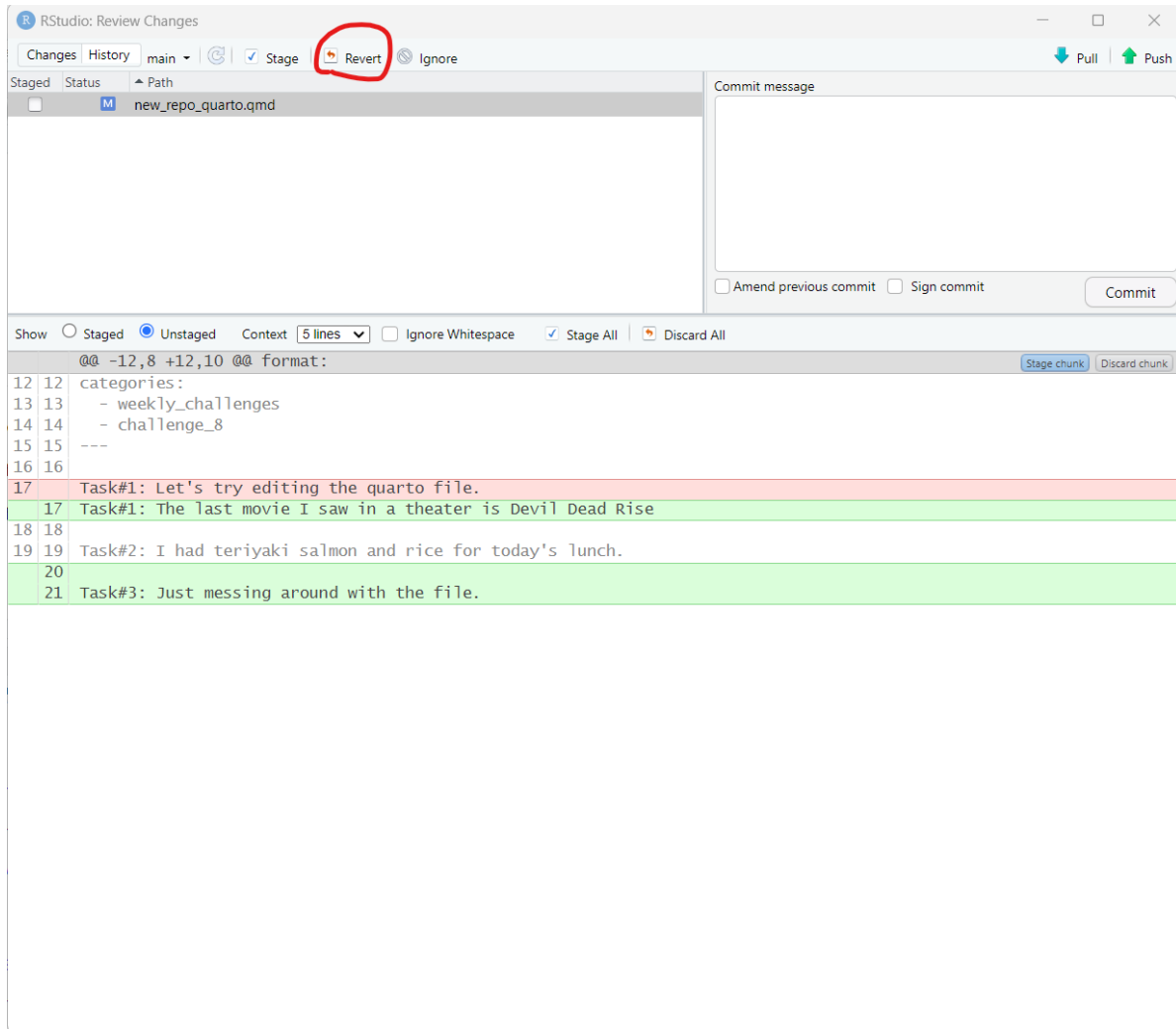
Note that GitHub is a platform or tool for code collaboration and sharing, and it is designed to record the history of all the commits/changes you made (especially when you only have one branch and it is the main branch). Generally, if you made a commit and push changes to GitHub, it is complicated to roll back to a previous commit/version (using either git-revert or git-reset, see this tutorial for more). Also, if you conduct a rollback by using git-reset, it **is dangerous in a collaborative environment: you're rewriting history**.

If you have not commit or push the changes, say if you just save and change the codes of the local files in the folder, we can still revert a change to the previously stored version in RStudio. Let's see how this work.

First, edit the .qmd file you created and then save it. When you save it, you will find a blue icon “M” in the Git window. This implies that Git recognizes a modification made to the file and reminds you that you have not synced or pushed the change to the online GitHub repository.

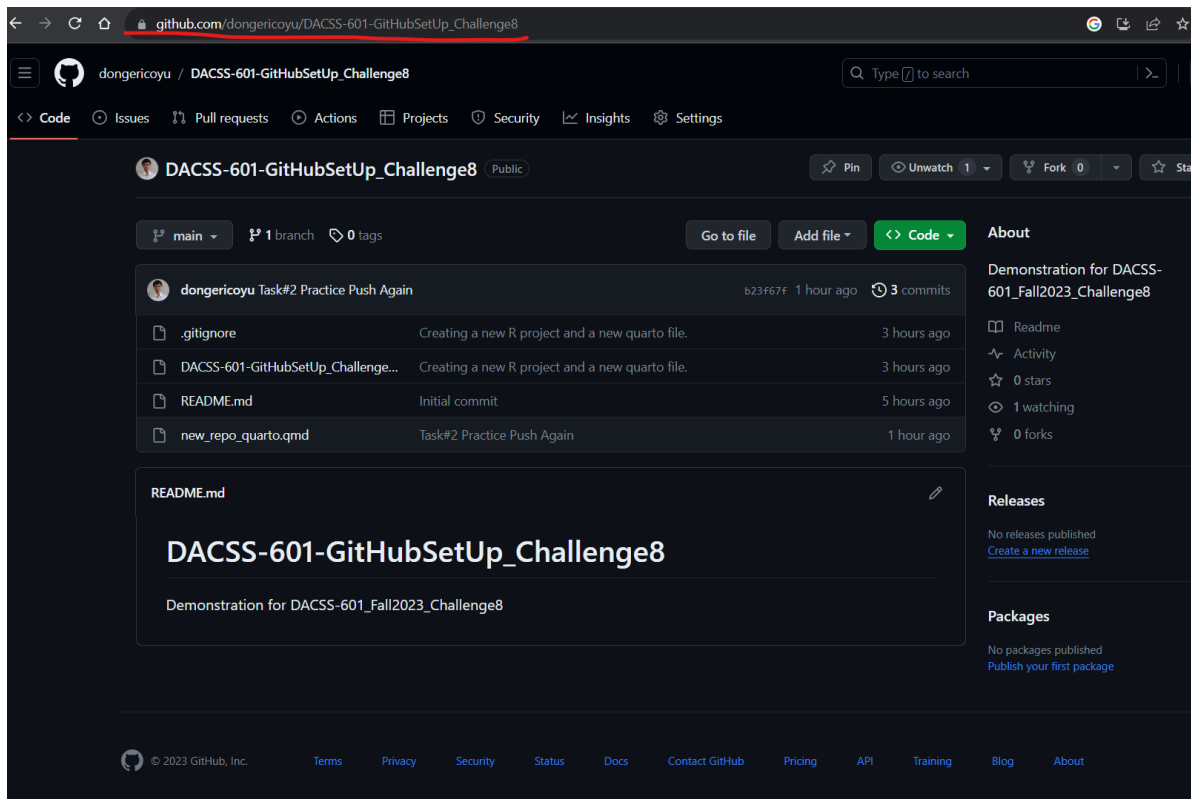
Note that at this point, we only save the changes locally (we have neither “committed” nor “pushed” the file changes to Github).

To retrieve the previous files and codes, you can select “Commit” on the toolbar and select “Revert” in the “RStudio Review Changes” Window (don’t hit commit!).



Submission Instruction

For challenge#8, please submit a Word document containing your GitHub repository's webpage link that connects to an RStudio project (see the highlighted link in the screenshot below). I will check the repository to see if you have a history of multiple commits of codes.



As for Task#3, please attach a screenshot just like the one I included in Task#3.

Optional#1: Connect an existing R project (such as the 601 challenge project) to GitHub.

Please watch the class meeting video of Dec 4 to see how this works. You can also refer to Happy Git and GitHub for the user: Chapter 16 and 17.

Optional#2: Using GitHub Desktop.

[GitHub Desktop](#) is a Git local client that allows you to monitor and manage project changes and do push and pull more efficiently. You can try this out through a few YouTube tutorials, such as [this one](#).

Optional#3: Collaboration: How to Clone, Folk and Pull a GitHub Repository to the local directory.

Folking, Cloning, and Pulling are very important Git functions when working on a collaborative project (either in R or in other program languages). If time permits, I will cover this on Dec 4 or 6 using the DACSS 601 course website used in the previous semesters. If there is no time and you are interested in how to do that, feel free to ask me for the manual and instructions on how to do that.

You can also learn more about them by reading Happy Git and GitHub for the useR: Chapter 22(Branches), Chapter 23(Remotes), Chapter 29(Pulling), and Chapter 30 (Clone and Folk),