Lab 00 Git

[Sicily Xie]

2022-09-13

Prework

1. Check your Canvas profile settings to ensure the email associated with your Canvas account is correct.
2. Review your Canvas notification settings and decide what you want to be notified about.
3. Visit the Course website. In particular, as you might expect, this course requires computing. We will use R and RStudio as well as Git and GitHub. See the Computing tab.
4. If you have never used GitHub before, go to https://github.com/ create an account. You should be aware that this data is stored on US servers. Please exercise caution whenever using personal information. You may wish to use a pseudonym to protect your privacy if you have concerns. k_asking_for_help/).

If you haven't already, visit https://ubc-stat.github.io/stat-406/computing/ and follow the instructions to set up your computer.

- Clone your `labs-<username>` repo. Navigate to the Course Github using the link at the top of the Course Website or from Canvas. Then go to your `labs-<username>`, Click the Green "Code" button, and copy the url by clicking the two overlapping squares. Then in RStudio, choose "New project" > "Version Control" > "Git" and paste the address. Choose a location on your machine where you want all your labs to be. Select "Create Project".

Lab overview

If you followed all the steps above, you should have an RStudio project for all the labs for this course. You will never have to do any of those steps again (except the last one, once, for your Homework).

In this lab, we'll do two things. The first is to demonstrate the "correct" way to do a lab or assignment and submit it. So we'll get this unfinished lab all ready for submission.

The second thing is to explore Git just a little. Specifically, we'll pretend like we're doing another submission and mess everything up. Then we'll fix it. Finally, we'll submit the whole lab.

The right way

Beginning a new Lab or homework assignment always starts out like this.

1. In the upper right quadrant of RStudio, you should see a number of tabs. Click the one that says "Git".
2. On the right side it should say `main`. This is the branch you're on.

3. ALWAYS, start from `main`.
4. Create a new branch by clicking the thing that looks like two purple squares pointing at a diamond.
5. You can name your branch anything you like, but I'd suggest `lab00-git` to denote the work that will happen here.
6. Ensure that "Sync branch with remote" is checked and click "Create".
7. Now open `lab00-git.Rmd`. You'll see all the instructions you've been looking at there in that document.
8. Try to click `Knit` or use `Cmd+Shift+K` (`Ctrl+Shift+K` on Windows/Linux). To render the lab to `.pdf`.
9. On the previous line, Delete the last sentence "You should always do this first...". And Save.
10. Now, looking in the Git panel, you should see `lab00-git.Rmd` and `lab00-git.pdf` with check boxes next to them.
11. Git "Stages" your changes when you click the check box. Click the box by `lab00-git.Rmd`.
12. Commit your changes by clicking Commit. A message window will pop up. Type a message about what you did. Something like "I hate verbose instructions". Click `Commit`.

The last two steps are the basic procedure you will always use. You save regularly. Every so often (maybe each time you finish a section), you Stage + Commit. For a lab to get full credit, you must have done this at least three times.

Note, only files that are "Committed" will ever go to Github for grading. You should only commit the `.Rmd` and the `.pdf`. If other things appear, you may be doing something wrong.

So at this point you've made one commit.

Aside

In other labs and homework, you'll see something that looks like

```
::: {.solbox data-latex=""}

some dumb text

:::
```

It's called `.solbox` because that's where your solutions will go. It makes it easier for the TAs to find your answers. So don't delete this. You should safely put code and markdown text inside, and it will all "work".

But watch out. It **MUST** contain text. That's why it starts with text in it. If you delete the text, resulting in an empty box, your document won't Knit.

End Aside

Let's pretend, now, that we're *Done With The Lab.* You should still see the unstaged `.pdf` file in that window. If so, don't worry. Click the Green Up Arrow.

That "Pushes" your changes to Github. There, the TAs can see what you've done. Go back to your Browser and take a look. You may need to refresh the page.

You should see a Yellow bar that says something like "`lab00-git` had recent pushes" and a Green Button that says "Compare & pull request". Click that button!

Now you can write notes to the TA that will review your work. You give it a title like "Submission of Lab 00". There are also some prompts for you to address. Go ahead and answer all the questions.

Click the "Preview" tab to see the warnings with emoji!

While you're there, scroll down and examine the changes you've made. You should only see a small modification to 1 file. Go ahead and click the Green Button that says "Create pull request".

At this point, you would be done (kinda). The TAs will automatically be triggered to review your work. You should see your Lab TA's name under Reviewers. You should also see the messages you left for them.

They can comment on what you've done and leave the grade. But we're not done. That's OK Even though you've opened the Pull Request (PR), you can still push more changes to the branch. So that's what we'll do.

In General, you would
`for (i in niters) {`

1. `for (j in work)` Do work and save.
2. Stage and commit changes.

`}`

3. Push your work to Github.

When done, go to Github and open a PR. Request review from the TAs.

> To avoid future headaches: use the dropdown menu to go back to `main`.

The wrong way(s)

Scenario 1. You do work on the wrong branch.

Make sure that you are on `main`. Remember that the actual submission is on the `lab00-git` branch.

In the R code chunk below, fit a linear model to the data and print the estimated coefficients, rounded to 2 decimal places.

```
library(tibble)
set.seed(12345)
dat <- tibble(
  x1 = rnorm(100),
  x2 = rnorm(100),
  y = 2 + 3*x1 - x2 + rnorm(100)
)
## fit a model
```

Now, stage the `.Rmd`. Commit with the message "on the wrong branch" and push.

You likely see an error like:

```
remote: error: GH006: Protected branch update failed for refs/heads/main.
```

That's because you're on `main`. Ugh! But I did some work, and now I need to be on a different branch!

So let's fix it. We want the stuff we just did on `main` to be on `lab00-git`. Note that everything you did is saved! Here are the steps:

Get our changes onto the correct branch

1. Use the dropdown to switch branches to `lab00-git`.
2. Go to the Terminal (next to console).
3. Type `git merge main`.

That should copy all your changes in the `.Rmd` that you made on `main` into the correct place. And open the commit message window with a prefilled commit message. Did it? Exit the window by following the prompt.

If you do this and you ever see stuff like

```
<<<<<<< HEAD
Adding some content to mess with it later
Append this text to initial commit
=======
Changing the contents of text file from new branch
>>>>>>> new_branch_for_merge_conflict
```

This means that there were conflicts between the two versions. The stuff above `======` was in your current branch. The stuff below is what you're trying to merge in. You decide what to keep, the top, the bottom, or both (or neither). Just be sure to delete the junk lines with `<`, `>`, or `=`.

Ok. So now we have our changes in the right spot. Commit and Push the `.Rmd` (only). Let's clean up `main` so we don't have problems later. Switch back to `main`.

Undo mistakes on the wrong branch.

In the Terminal, type `git log`. You should see some commits, one with the message "on the wrong branch". There's a bunch of other text that I won't try to explain, but look at the stuff *before* (meaning below) that message. That's the commit from before you did work on the wrong branch. You should see something like:

```
commit 1851c738984690b039a79a04e070f766e19993d5
```

That long string is what we're after. It's called a hash, and it uniquely identifies the commit. Take note of the first characters (like 5). That's usually enough to get away with.

Type `q` to exit the log viewer.‘

Now type `git reset --hard 1851c` (replacing `1851c` with the numbers from your unique hash). This command will undo any changes after that commmit, but only for this branch. (You may have to click the little circle-arrow in the git pane to update the contents there.)

To recap, now the work we want is in the right place (on the other branch), and the mess on `main` is cleaned up. Boom.

Scenario 2. You did something you shouldn't have

Switch your branch back to `lab00-git` (or whatever you named it).

Open the file `lab01.Rmd`. Select everything after `# Instructions` and delete it. Save. Then Knit (producing a pdf). Commit both files with a message "did the wrong lab, and built a pdf". Push your commits with the Green up arrow.

Take a look at the PR on Github now. There's a bunch of crud that shouldn't be there.

We've done 3 things here that we shouldn't have.

1. We built a pdf that we don't want at all. It needs to go away.
2. We bollixed up the `lab01.Rmd` file. We don't want that or it will screw up the lab next week.

3. We pushed it all into our submission for this week.

The first instinct is to Delete both files, commit, and push. This is **VERY BAD**. That will further screw up everything. Basically, you're telling git "I don't want these files at all" when you mean "I don't want changes to these files in this branch". The difference is subtle but important. Because you DO want these files (without the changes) at some point, but you don't want them here.

Let's fix these issues.

First, we want to "get rid of" the pdf. In the Terminal type

```
git reset HEAD^ -- lab01.pdf
```

Click the little "Refresh" arrow in the Git panel. You should now see `lab01.pdf` twice, once with a red D that is checked and once with two yellow question marks that is NOT checked. This is what we want.

Commit exactly as is. Use a message like "remove the stray pdf" and Push. Now, take a look at the PR on Github. The `lab01.pdf` should be gone from the list of files in the PR.

There's still that annoying two-yellow-question-mark version in the Git panel. Don't click the check box (that will just redo everything we undid). Instead, look in the lower right quadrant at the "Files" tab. Find `lab01.pdf`, click the checkbox, and choose Delete. Now it's gone.

The first step removed the `.pdf` from Git's tracking. The second deleted it from your machine.

Second, let's "undo" the deletion in the `.Rmd`. This is easy, and a useful pattern to remember.

In the Terminal, type

```
git checkout main -- lab01.Rmd
```

What this does is grabs the version on `main` that isn't messed up and puts it here, overwriting your changes. This isn't the only way to fix your problem (you could have done the same thing we did with the pdf), but it's pretty easy.

Stage commit and push. Now look at the PR on Github. Even though you made two changes (one deleting everything, and one restoring everything) to the `lab01.Rmd`, it should be "gone" from the PR now. That's because the version on this branch looks just like the version on `main`, so there are no changes to be made into the `main` branch. This is just what we want.

Now we've also fixed the third error already. None of those bogus changes to `lab01` are in our PR for this week anymore.

Finish up

Your Git panel should still have the `.pdf`. Go ahead and Save and Knit `lab00-git.Rmd` again just to make sure. Then stage and push it and the `.pdf`.

Take this opportunity to change your branch to `main`. This will avoid issues when you start the lab next week.

We're now done with this lab. This all probably seems a bit painful, but our goal is to avoid all these things in the future. If you're careful, in this class, you won't have to do any of this junk again. In real life, if you work in Data Science or Software Development or Machine Learning, you might. Let's just recap THE RIGHT WAY.

1. For HW or Labs, always start on `main`.

2. Create a branch for your HW/Lab and switch to it. The name doesn't matter, but it's good practice to name in something meaningful (rather than something like `stat406-lab-1` when you're doing lab 4).
3. Open the HW/Lab `.Rmd` and click Knit. Make sure it works.
4. Do the work, saving regularly. When you complete a section, Commit the file with a useful message (Push or Not).
5. Once you're done, make sure that you have done the minimum number of Commits, push your `.Rmd` and the knitted `.pdf`.
6. Open a PR and respond to the questions.
7. Make sure that only the `.Rmd` and the `.pdf` for this HW/Lab are there. And Create Pull Request.
8. On your machine, switch the branch to `main` to prepare for the next HW/Lab. And click the Blue Pull button to sync your `main` with the one on Github.

If the TA asks for changes, just switch to the branch for this assignment, and make the requested changes. It's all on your machine (even if the pdf disappears when you switch).