How to use GitHub

Dominique Maucieri

25 January, 2022

Contents

Getting started with Git

What is Git?

Git is a version control software that uses local files allowing ease of merging and branching.

Github is an interface that very user-friendly, helps with use of git software and collaboration with others. This makes it a popular choice for open source projects.

Prep

To prep for using Git and GitHub, follow these steps:

1. Update your R / R Studio, if you haven't recently. Do yourself this favour because there have been some large changes to R recently. This tutorial was created with the "Bird Hippie" release version 4.1.2 so if you are at least newer than 4.0.0 you will probably be fine but if your version starts with a 3, its time to update. You can update R **here** and R Studio (just do the desktop open source edition) **here**. Both require you to redownload the program to update them
2. Get an account on GitHub (https://github.com/)

3. Locate this tutorial on **my GitHub page**, this is where this repository is being stored and how you will access it
4. Then we heed to set up Git. See **here** for more information. We will also need git on our computers, so download git **here**. When prompted enter the username and email address you used for creating your GitHub account. You will need to locate your terminal on your computer to do this. For Macs, just search for terminal on your computer and open that application. For Windows, see the note in this **link**
5. If you found that using this terminal a bit terrifying, it doesn't fit well with your style of workflow or you don't want to feel like a computer hacker, try downloading another program to help out with this like the **GitHub Desktop**, a code editor like **Visual Studio Code** or even a great text editor like **Atom**. I have used all of these and will give some insight as we go. If you have the time, try them all out and figure out what works for you. You don't need to download any of this though, I usually only work in the terminal, but there are many options so you get the most out of your time. To help with making the terminal easier to learn, I have added a cheat sheet (GitHub_syntax.rtf) to the folder containing this script with a bunch of terminal syntax and their meanings.

With this prep, you should be able to start cloning repositories, though you will need to authenticate with GitHub if you want to be able to change any repositories.

Tokens

New updates in GitHub, to improve security involve using tokens to authenticate your account on different devices. These Tokens should be regenerated every so often to make sure they are secure, and every time you want to use your token to log into something, you have to regenerate it. To do this, follow **this** and essentially you will use this token as your password for logging into the terminal.

What you can do with GitHub

Cloning a repo

Since GitHub allows you to share code and data online, it means that you will have access to many different resources now. If you wanted to be able to take some of the code or data you see online, such as this script, and put it onto your own computer to look at and maybe play with, this is known as cloning a repository.

This is very easy to do.

1. Locate the repository you want to clone, such as **this one**
2. locate and click the large, green icon on the top right side, that reads "code"
3. follow the rest based on which program you want to use

a) GitHub Desktop just click the "Open with GitHub Desktop" button directly. When it brings you to the application, select where you want to put this file on your computer and clone. Now celebrate, you have this repository on your computer now!

b) Visual Studio Code just copy the url and then open the VSC application. On the left side, click extensions and then install the "GitHub Pull Requests and Issues" extension. Click the GitHub symbol and sign in. Then click the weird y like symbol (third from the top) on the left side when downloaded, and clone a repository (may need to click the three little dots at the top by source control to find clone). Paste the url when prompted, hit enter, then select where you want to save those folders. Now celebrate, you have this repository on your computer now!

c) Atom just copy the url and then open the application. Open the command palette (shift + command + p on a Mac), search for github: clone, and then paste the url in the "clone from" spot and select where on your computer with the "to directory spot". Click clone and celebrate, you have this repository on your computer now!

d) Terminal command line just copy the url and then open the terminal. First locate where on your computer you are. Using `cd foldername/foldername` etc locate where you want to put your new folder. When in that location with your terminal, enter `git clone 'url'`

With this repo now on your computer, you can open all the file, work on scripts, use the data etc.

Pulling a repo

Sometimes the repositories we have on our computers are updated by the creators, coauthors, or any other contributor, so before you were to send your changes to GitHub its best practice to figure out if there are changes that have been made that you don't have on your computer first. Additionally, if you were just looking at a repo and came back to it after a while, you might want to see if there have been any updates on it.

To do this, we must "pull" a new update on your repository. This command should be used at a minimum, **EVERY** time you interact with a repository.

How to pull:

a) GitHub Desktop

- Once in the repository you are working in, click fetch at the top
- Then if there are updates they will be listed below
- Click "pull origin"
- Done, you now have the updated changes on your computer

b) Visual Studio Code

- Once in the repository you are working in, go to the source control (the weird y like symbol (third from the top) on the left side)
- Click the three small dots and then click pull
- Done, you now have the updated changes on your computer

c) Atom

- Once in the repository you are working in, make sure you have logged into GitHub
- Click fetch at the bottom or open the command palette (shift + command + p on a Mac), search for github: pull
- Done, you now have the updated changes on your computer
- Note: these should be the steps, it doesn't actually work on my computer, due to the unsolved issue **#2720** so I pull in the terminal command line

d) Terminal

- Make sure you are in the location of the repo folder
- Then type `git pull` and hit enter
- Done, you now have the updated changes on your computer

Creating a repo

To create your own repo, the easiest is to start it on the GitHub browser. Go to your profile, then your repositories, and then click the large green "New"

From there you should name your repository something useful, and add a description. Even if its just "MSc thesis" at least thats something.

Private repositories are nice if you have lots of unplublished data and you don't want the stress of people looking at your half completed analyses. Don't worry though, you can still add collaborators to private repositories so you can share it, it just means that it won't be found by everyone.

Public repositories are great for completed manuscript analyses and making sure your data and code are open source. You can also archive your data with something like **Zenodo**

You should ALWAYS initialize your repository with a README file, but a .gitignore and license can be added after so I don't usually initialize with them.

Then just create the repository. Now with this repository created, you can clone it to your computer and start working in it. If you created files before hand, just move them into the folder for this repository that you have cloned onto your computer, and then go to the commit section to see what you do next.

There are ways to create a repository on your computer and then push it to git, but I have never had success with that, so this is how I do it. If you want to create repos on your computer, google is your friend and good luck.

Now if you have created a private repository, you probably want to share it with a collaborator or at least your supervisor. To do this while in your browser and on your repository, go to settings (located at the top of your page, in the menu under the repository name), then co to Collaborators (located on the left menu) and after verifying your password if prompted, you will add people based on their GitHub username.

Commiting changes to a repo

Once you go ahead and start working on scripts, adding data, filling out your README files, essentially any kind of edit to your repository, these changes are only on your local version of this repository. If I were to clone the repository from GitHub onto my computer, I wouldn't see these changes. So now we have to figure out how to get your changes back to the cloud.

There is a general workflow for this, which involves adding your changes, commiting those changes and then pushing these changes to the GitHub cloud. Only after the push will these changes be reflected on GitHub.

Okay so how does this work.

First is adding. You need to first add the changes you have made to a staging area. You do not need to add all changes, sometimes it can be useful to do a few changes, commit those changes and then repeat by adding a few more changes and committing those. You can also add all changes you have made, just depends on your workflow style and what types of changes you have made. After you add your changes to the staging area, you will commit these changes which should be accompanied by a message to let yourself and/or collaborators know what changes you made.

a) GitHub Desktop

- You should see on the left side a changes tab, open this if it is not already, and look at the changes you have made.
- Pull to ensure there are not changes to your repo you don't have on your computer (you are unable to pull after you commit so it is a good idea to pull add commit and push within a quick time span to ensure no changes are made by collaborators in that time. GitHub Desktop should pull as it pushes to ensure there are no conflicts but this will also ensure nothing happens unintentionally)
- To add changes, check the box beside them, you can check all or just select ones, up to you
- Then to commit these changes, write a message in the bottom where it says Description.
- When that message looks good, select the commit button on the bottom.
- These changes you have selected should disappear
- To check what commits you have done, click the history tab and see that there is now a new commit with a little arrow beside it, showing you haven't pushed these changes yet.

b) Visual Studio Code

- Make sure you are in the location of the repo folder
- Files that have uncommitted changes will show as a different color
- Go to the source control (the weird y like symbol (third from the top) on the left side)
- Clicking on a file will show what changes have been made.
- To add the changes, just click the little + beside the file, this will stage the changes
- There is a place to add a message for the commit at the top

- Once you have entered a message, you can click the check, or the three dots and select commit > commit staged
- It is not as good at showing what changes you have made, but it will tell you how many commits you have made without pushing to GitHub on the left menu.

c) Atom

- Make sure you are in the location of the repo folder
- Files that have uncommitted changes will show as a different color
- If you open the Git menu, it will list all the files you have made changes to
- If you click on one you can see what changes were made, and you can click the small square to stage the file, or the stage all button to stage all
- Then add a message for the commit, and hit commit to commit those changes
- Note: these should be the steps, it doesn't actually work on my computer, due to the unsolved issue **#2720** so I perfom these steps in the terminal command line

d) Terminal

- Make sure you are in the location of the repo folder
- To see what changes you have made you can use `git status` and it will tell you what status all your files are in
- To add files to the staging area use `git add "name of file"` for adding individual file changes, or `git add -A` for adding all file changes. Note: replace "name of file" with name of file, the " " are not needed
- Then to commit these changes you will use `git commit -m"a message of what you did"` Note: replace the " " are needed around your message
- And that is it, these changes should be commited, to check you can call `git status` or `git log` to see.

Pushing changes to GitHub

Once you have committed some changes, you will need to push these changes back to GitHub. Fist you should **ALWAYS** pull to ensure there have not been any changes to your repo, by yourself or collaborators. Whether you think you need to or not this is considered the standard and will save you from conflicts and issues with your push.

You can commit several times before you push, so it can be useful to group similar changes together when you commit changes and then push many commits together. Decide on what workflow works best fo you and your collaborators and try to be as clear and informative with your messages as you can for your commits. When you are ready to have your commits reflected on GitHub, then you will pull and then push your changes.

a) GitHub Desktop

- Make sure you are in the location of the repo folder
- On the changes tab it will tell you that you have commits that have not been pushed, you can just click that button to push the changes
- Alternatively at the top where the fetch button used to be it should say push, and you can click that to push the changes as well.
- Thats it, you pushed your changes
- Now if you go to your repo on GitHub in your browser, these changes should be there with your commit message shown

b) Visual Studio Code

- Make sure you are in the location of the repo folder
- Pull to ensure there are not changes to your repo you don't have on your computer
- On the left menu there will be a button to push changes
- Thats it, you pushed your changes
- Now if you go to your repo on GitHub in your browser, these changes should be there with your commit message shown

c) Atom

- Make sure you are in the location of the repo folder
- Pull to ensure there are not changes to your repo you don't have on your computer
- Then click push, which should be under where you entered your commit message
- You have now pushed these changes to GitHub
- Now if you go to your repo on GitHub in your browser, these changes should be there with your commit message shown
- Note: these should be the steps, it doesn't actually work on my computer, due to the unsolved issue **#2720** so I perfom these steps in the terminal command line

d) Terminal

- Make sure you are in the location of the repo folder
- Pull to ensure there are not changes to your repo you don't have on your computer
- To see see the changes you are pushing, you can look at `git status`
- To push these changes to GitHub use `git push`
- It may take a second and you may be prompted to add a username and password (this should actually be a token, see the token section above), once all that is entered you won't have to redo that step until the token expires
- You have now pushed these changes to GitHub
- Now if you go to your repo on GitHub in your browser, these changes should be there with your commit message shown

Branches and basic workflow

When you are collaborating, it can be helpful to use branches and standardized workflows to ensure there are no conflicts between the changes that your collaborators have done with what you have changed.

This is where branches can help. You can refer to **this website** for a more in depth explanation but essentially your main branch, what we have been working on up until now should be saved as your 'good' branch, where everything always works and there is nothing causing errors or unfinished. If you wanted to add something or change how something was working in a file on that branch, you will first create a new branch. What this does is make a duplicate of your repository and call it whatever you named your branch, for this example we will use dev as in development branch. You could name this after what you intended to do or by the name of the person working on that branch, however you want to keep everything organized.

Instead of editing and changing the main branch directly, you will work on this dev branch, knowing that you still have the main branch saved and working well. Then when you get your new feature working on the dev branch, you will merge the two branches, which will update the main branch and ensure the main branch is always working.

This can make collaboration easier as you don't need to worry about conflicts between collaborators or the main branch no longer working. I suggest looking into branches if this interests you.

Some other tools

Working with .Rmd files

Markdown files allow for you to create neat outputs of your code and comments.

How to create and set up R Markdown files

- To create a new R Markdown, go to "File >"New File" > "R Markdown"
- In the popup, title your file and list yourself as the author. Here is where you specify the format of your output, .html or .pdf.

  - This choice is up to you .pdf may require additional programs on your computer, but it will create cleaner looking outputs, while .html can support some functions that .pdfs cannot such as interactive plots

- Once you have your .Rmd file created, edit the top information so it is informative and will set up your R Markdown file, with your name and title:

For an .html knit output: title: "First R Markdown file" author: "Dominique Maucieri" date: "25 January, 2022" output: html_document: toc: yes toc_depth: 3 toc_float: yes

For a .pdf knit output: title: "First R Markdown file" author: 'Dominique Maucieri' date: "25 January, 2022" output: pdf_document: toc: yes toc_depth: 3

- This information will create the title for your outputs and allow you to format the table of contents of your final knit output. The date argument will automatically save the date at the time of knit. Knitting will take your R Markdown file and turn it in to a more polished final document that can be sent to collaborators to illustrate and compile your findings.

- You can choose to knit your R Markdown into an .html or a .pdf file, I will be using both so you can compare the difference. In order to have a .pdf output you will need to have a tex editor on your computer. https://ciser.cornell.edu/rmarkdown-knit-to-html-word-pdf/ may help if you want to make a .pdf output. No additional programs are needed for a .html output.

Syntax

- The use of "#" will allow you to create table of contents. The toc_depth argument above will tell you how many levels you can have in your table of contents (I have specified 3). A single "#" will be the largest table of contents heading, "##" will be smaller and "###" will be even smaller. Headings will be nested within the larger ones. More "#" the smaller the heading.

- https://rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf

There is a lot of syntax that can help you to make effects within your text.

You can **Bold** text with two "**" on both sides of the words you want to bold

You can *Italics* text with one "*" on both sides of the words you want to italicize

You can make lists with one "*", then a"+" and even a "-"

- First list item "*"
  - list sub-item "+"
    * list sub-sub item "-"

How to code R in R Markdown

In R Markdown, the text here will be normal text in your final output. To add R code at the top right of the script select "Insert" > "R" then in the curly brackets {} it will say r, add a title for that chuck such as "setup" as I have in the chunk below (can only see if you open this code in RStudio)

{r setup}

As long as you are coding within the R chunk, coding will be exactly the same as a .R file, other than setting a working directory. You may need to set the working directory in each chunk where you are loading or saving data as this only sets the working directory in a single chunk. Or to get around this you can work with R Projects as we will discuss later.

To run your code, you can either run each line individually, or highlight select rows and use the same commands we used in the .R script files, or if you want to run a whole chunk, you can select the play button at the top right of the chunk. There are many different ways to run an R Markdown file, and you can see these options in the pull down menu at the top right of the script, when you click 'Run'

To knit the data together, you use the knit function at the top left of the script. This will knit based on how you have specified your document above in 'Setup'.

```
library(MASS)

rm(list = ls(all=T)) # this code will clear your environment, giving you a clean slate from
# which you can start your new script

dev.off() # this code will close any plots that are currently open
```

```
## null device
##           1
```

What you see when this is knit is a lot of output. This is what would be outputted in the console when the code is run. But sometimes we don't want this to be seen in our final knit. To remove output and other things, we can add arguments to the curly brackets where I named the chunk {r setup}

Syntax examples Notice how I used 4 "#" and this heading did not show up in the table of contents, as I specified the levels I have in my table of contents and I only chose 3.

- {r setup, echo = TRUE}
  - will keep code the code in your knit output **use this as we want to see your code in the output**
- {r setup, warning = TRUE}
  - will show warnings in your final knit output, if you don't want them to show you could use warning = FALSE instead
- {r setup, message = FALSE}
  - will remove any messages from your final knit output
- {r setup, results ='hide'}
  - will hide the outputs from the code, however this will also remove plot outputs
- {r setup, fig.height = 3, fig.width = 3}
  - will specify plot output in inches, this example is a figure that is 3x3 inches

```
dev.off() # this time there will be no output
```

Sometimes you want to show the dataframe and outputs in your final knit though

```
data(Animals)
Animals
```

```
##                       body  brain
## Mountain beaver      1.350    8.1
## Cow                465.000  423.0
## Grey wolf           36.330  119.5
## Goat                27.660  115.0
## Guinea pig           1.040    5.5
## Dipliodocus      11700.000   50.0
## Asian elephant    2547.000 4603.0
## Donkey             187.100  419.0
## Horse              521.000  655.0
## Potar monkey        10.000  115.0
## Cat                  3.300   25.6
## Giraffe            529.000  680.0
```

```
## Gorilla            207.000  406.0
## Human               62.000 1320.0
## African elephant  6654.000 5712.0
## Triceratops       9400.000   70.0
## Rhesus monkey        6.800  179.0
## Kangaroo            35.000   56.0
## Golden hamster       0.120    1.0
## Mouse                0.023    0.4
## Rabbit               2.500   12.1
## Sheep               55.500  175.0
## Jaguar             100.000  157.0
## Chimpanzee          52.160  440.0
## Rat                  0.280    1.9
## Brachiosaurus    87000.000  154.5
## Mole                 0.122    3.0
## Pig                192.000  180.0
```

```
str(Animals)
```

```
## 'data.frame':    28 obs. of  2 variables:
##  $ body : num  1.35 465 36.33 27.66 1.04 ...
##  $ brain: num  8.1 423 119.5 115 5.5 ...
```

For more help with R Markdown

https://ourcodingclub.github.io/tutorials/rmarkdown/

Creating R Project

1. click "Project: (None)" in the top right corner of a clean R session in RStudio
2. click "New Project . . . "
   (a) "New Directory" if you do not have an existing folder/repository with your data or scripts in
      - here you will be asked to name this folder (Directory name), choose the location for this folder on your computer (Create project as subdirectory of) and choose if you want to create this as a git repository
      - in-order for this git repository to be initialized, you need to specify this in the git terminal, and because of this I think it is more difficult and I don't recommend. Instead I would follow (b) or (c)

   (b) "Exiting Directory" if you already have a folder that you are working in, and want to turn it into a project.
      - here you will choose the location of the fold with "Project working directory"
      - this is a good choice if you already have your repository on your computer

   (c) "Version Control" if you have a repository in git, and you want to clone it to your computer and create a project
      - here you will be asked to enter the "Repository URL" found on the GitHub repository under the green "Code" button

      - it will enter the directory name automatically and you can choose where to save the repository on your computer with "Create project as subdirectory of:"
3. Once specifying how to create the project, it's all set up. Now when you want to open your files, just double click on the project and it will set up RStudio for you, and where you left off. You may need to commit and push to git if this repository is on GitHub.

Working with R Projects

- R Projects run isolated code so the code in one Project does not affect other Projects even if open and running code at the same time
- Checkout the simplified_filepaths.R file and then the simplified_filepaths.Rmd file for why these R Projects are so useful and should be considered when working with R Markdown files

What is also helpful is that you can do "pull" "add" "commit" and "push" all from the R Studio browser when you are in a project that is also a git repository on your computer, in the top right corner of R Studio, you can see a git tab where you can perform all these actions from R Studio.

Pull will be the blue down arrow. Add will be checking each file. Commit is the button that says commit, and it will open a new window where you will add your commit message and look at your changes to files. Finally push is the green arrow.

READMEs

These files are markdown files, and can be edited in text editors, or for Mac's I like MacDown for visualizing my code and as I write it.

These files should have all necessary information in them for anyone looking at your repository. They act as a "README" before you look at all the files so you know whats going on. Don't forget about updating these as you update your code. There are also many great online sources of markup syntax like this cheatsheet. This syntax also works in the markdown parts of an .Rmd file, like what you are reading right now.