Setting up GitHub to work with RStudio

One time steps:

**Making a GitHub Account**:

1. Go to <https://github.com/>
2. Enter your email into the central box, labeled “Sign up for GitHub”
3. Give yourself a username – the best ones are something with your name or a moniker you use in other contexts. It helps people find your work easier and know who they are dealing with.
4. Enter a password for your account.
5. Register for a free account – this can always be changed later if you decide you need the above and beyond features included in

**Installing Git**:

Installing Git on your machine depends on what type of operating system you use. As such, my instructions are broken up by operating system.

**Linux**: This is the easiest case. That is because Linux users already have to use the command line for many things.

For Ubuntu or Debian Linux: run sudo apt-get install git

For Fedora or ReHat Linux: run sudo yum install git

**Mac**: The easiest way to install Git in macOS is through the shell/terminal. I recommend doing this via installing the Xcode command line tools, which includes Git.

1. Open the terminal (Applications -> Utilities -> Terminal)
2. Enter the following command in the terminal: git config
3. This will elicit an offer to install developer command line tools – accept this!
4. Click Install

**Windows**: Windows does not have a built in command line terminal. As such, you will want to install Git using an installer that gives you a terminal to work in. I recommend [Git for Windows](https://gitforwindows.org/) and will be referencing that process here.

1. Go to <https://gitforwindows.org/> and download the installer.
2. Select all the default and recommended options while running the installer.

**Letting Git know who you are**:

1. Go to your Git terminal (for Mac and Linux, the same terminal you used to install Git. For Windows, the installer will have installed a program called Git Bash – this is your terminal).
2. Enter the following three lines into the terminal, pressing enter after each one. NOTE: make sure to use the same email you used to create your GitHub account.

git config --global user.name ‘John Doe’

git config --global user.email ‘youremail@example.com’

git config --global –list

1. The last command should display your name and email, indicating that Git recorded your information.

**Installing a Git client** (optional):

This step gives you a graphical interface for Git is you are uncomfortable with the command line. If you want a graphical interface, [GitKraken](https://www.gitkraken.com/) is a great option. For Windows users, [SourceTree](https://www.sourcetreeapp.com/) is also a great option. Installation is simple – just click install and follow the installer prompts.

**Making a Repository & Connecting Git and GitHub**:

1. Got to your GitHub profile
2. Click the green “New” button to create a new repository.
3. Name your repository and give it a description
4. Select public or private
5. Check the box for “Add a README file” – this will give you a file in which you can record notes on the project and any details people looking at the repository should know.
6. Click the green “Create Repository” button
7. Copy the URL that is displayed in the quick setup section
8. Go to your GitHub terminal
9. Enter the following command, with close attention to accuracy. Replace my URL with your own. This command will clone the repository you just created on GitHub into your local Git.[[1]](#footnote-1)

git clone <https://github.com/MikaelaKarstens/GitHub-For-Research.git>

1. Set this repository as your working directory by entering[[2]](#footnote-2): cd Repository-Name
2. Add a line to your README file from your Git terminal or client to confirm that Git and GitHub are communicating
   1. Enter into the terminal: echo “Test Text Here” >> README.md
   2. Enter git status to check that your changes are noted by Git
   3. Enter git add -A to stage your changes for the commit
   4. Enter git commit -m “Explanation of what you did” to label what this version added or changed. The text you enter should be succinct and informative. Things like “added placebo test model” or “Created figure of main results” are good examples.
   5. Push this to the cloud stored repo by entering git push
   6. Go to your repository on GitHub to see that the changes have been made.

**Connecting that Repository to RStudio**:

1. Make sure your R and RStudio are up to date.
2. Update your packages while you are at it by running the following: update.packages(ask = FALSE, checkBuilt = TRUE)
3. Make a repository on GitHub, as we did in the previous section. Or click on the green “code” button on an existing repository to display the clone URL. Copy this.
4. In RStudio, start a new project with this GitHub repository
   1. File > New Project > Version Control > Git
   2. Enter your copied URL in the “Repository URL” field. The repository name should auto populate.
   3. Check the “Open in New Session” box – this is optional but recommended
   4. Click “Create Project”
5. Confirm that you can make changes to files in your repository through R
   1. Click on one of your files in the file window – the README file is a good test
   2. Enter changes to the file as normal in R.
   3. Click the save button, just like you normally would while coding.
   4. Click the “Git” tab in the upper right panel to start committing these changes to your GitHub repo.
   5. Verify that the files you worked on an saved have a blue “M” next to them. This indicated that the file was modified from what was originally in the repository.
   6. Check the “Staged” box next to any files you want to push to the cloud repository.
   7. Click “Commit” – this will open a window that displays all the changes that you have made. Additions are in green, removals in red.
   8. Enter a commit message stating what changes you made.
   9. Click “Commit”
   10. Click the green “Push” arrow to push all the commits you have made thus far to the cloud.
6. Go to your GitHub repository to confirm that the changes were made.
7. If it does not auto-populate, check the “Staged” box next to your R project file and go through the commitment process to upload the whole project file to GitHub.

**Workflow for working on existing projects – Solo Work**:

1. Open the R project file through GitHub
2. Click the blue “Pull” arrow to ensure you have all the latest file versions
3. Code as much as needed – I prefer making a commit per larger task but you can split it up however you like.
4. Save you file.
5. Check the “Staged” box next to the file you worked on
6. Click “Commit”
7. Enter a commit message regarding what you did.
8. Click “Commit” to confirm
9. Press the green “Push” arrow to push your work to the cloud.
10. Repeat as needed.

Workflow for working on existing projects – Coauthored Work:

1. Open the R project file through GitHub
2. Click the blue “Pull” arrow to ensure you have all the latest file versions – this is very important in coauthored work!
3. Code as much as needed. With coauthored work, I like to commit regularly so work is well tracked.
4. Save your file.
5. Press the blue “Pull” button to check any changes your coauthors have made since you last checked. Get in the habit of doing this often – this will prevent you from creating conflicts that will have to be resolved. If you get a message stating “Already up-to-date” that means there haven’t been any other changes pushed out tet.
6. Check the “Staged” box next to the file you worked on.
7. Click “Commit”
8. Enter a commit message regarding what you did
9. Click “Commit” to confirm
10. Press the green “Push” arrow to push your work to the cloud.

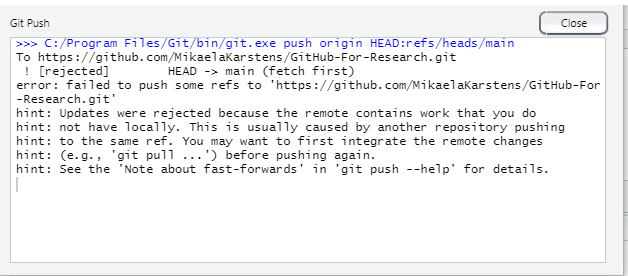
**How to make a branch**:

If you are going to be doing a lot of work by yourself that is different from what your coauthors are doing, it can be helpful to work in a branch until you are finished. This means that you will be basically working solo for a while until you decide to merge your branch back into the main code file. This will help you avoid stepping on each other’s toes while you work.

1. Go to your repository on GitHub
2. Click the drop down box labeled “main” in themed-upper left.
3. Type in a name for your branch
4. Click “Create Branch: Branch Name”
5. Open RStudio and look at the Git panel
6. In the upper right corner, select the drop down that says “main”
7. Change this to your newly created branch.
8. Continue working in the branch as normal.
9. When done, go to GitHub and submit a pull request to start merging the branch in with the main file.
10. If there are no conflicts, click the green “Merge pull request” button to rejoin your branch to the main file. Delete your branch by clicking the “Delete branch” button.
11. If there are conflicts, resolve them. See online guides for help.

**Managing Merge Conflicts in RStudio**:

You know this has happened when you try to commit and push to GitHub only to get error messages. At this point, do not panic. A code file will open displaying the differences between your code and the most recent version.



1. If you are unsure of what file is the problem, go to the Terminal in R and enter git status
2. Click the blue, “Pull” arrow to pull the most recent version.
3. You will now have a file showing that has both your version of the file and the conflicting versions. Conflicting sections will be sandwiched between lines like <<<<<< HEAD:file.html and ========= or ======== and issue-5:file.html. The code between the head and the dashes is your version. The code between the dashes and the issue footer is the conflict.
4. Go through the file and fix all the conflicts.
5. When done, go to the Terminal (tab to the right of console) and enter git add filename.html to record the fixed conflict.
6. Commit and push as usual.
7. Remember! The more often you pull, the less conflicts will be there to fix.

**Managing Pull Requests**:

1. Find any open pull requests in the “Pull Requests” tab of your GitHub Repository.
2. View the changes and leave a comment for your coauthor if needed.
3. If everything looks good and there aren’t any conflicts to resolve, click “Merge Branch” to add the new code into the master file.
4. Click “Delete Branch” to remove the branched off code.
5. If there are conflicts, resolve them before merging.

**Notes**:

* You don’t always have to make a full commit when you push updates to GitHub. In general, commits act like sign posts, letting you know where certain changes were made. The less certain you are about what you are doing, the more commit messages you might need. Feel it out as you go.
* When coauthoring, pull often! his prevents

Great resources:

<https://happygitwithr.com/>

<https://cfss.uchicago.edu/setup/>

<https://resources.github.com/whitepapers/github-and-rstudio/>

1. Note: If you are using a Git client, like GitKraken, you can do this step by clicking on “Clone a Repo” and entering the requested information, which will include the URL copied in step 7. [↑](#footnote-ref-1)
2. cd means change directory [↑](#footnote-ref-2)