How to start with GitHub and git for R*?*

Szymek Drobniak

December 27, 2024

This documnet describes the basics of using `git` and *GitHub*[1] - one of the most widely used version-control systems. It is not meant to be a complete guide or an exhaustive introduction, but hopefully it will provide an easy-enough start to most fresh users.

[1] It is assumed that the user has the necessary prerequisites in place. These would be: `git` installed in your OS (often it's present by default, if not - go to `https://git-scm.com`); the *GitHub Desktop* app downloaded and installed from `https://desktop.github.com/download/`; an account setup on `https://github.com`

What is `git`*?*

Although not the same thing, `git` and *GitHub* are related. `git` is a version-conrol system. It is a small app installed locally, that stores in each of its local repositories the modification history of all repository files. Usually, a repository is a folder where `git` was initiated[2]

Repositories that are kept locally are visible only to you. However, you can synchronise them to a remote, online instance of a repository. in such case, you can both send your changes to the online one (they become integrated in it, with the history of modifications kept) or download changes introduced by others if the repository is shared with multiple contributors. There are several main operations with their characteristic `git`-lingo names[3]:

[2] It can be done using the `git` program, but it can also be done in a much simpler way using the desktop *GitHub* app - see below.

[3] Note that I will provide selected, easiest ways to perform some of the `git` operations;many can be done in several ways, I'll leave exploration of possibilities to you.

- **Commit** - which means recording all changes done to the files in a repo as "registered" (i.e., included in the history of modifications of a repo);
- **Push** - which means sending the changes commited in the local repo to its remote, online version - such changes are integrated there if you have writing privileges in the repo, or are queued for review and integration as so-called **Pull Requests**;
- **Pull** - an operation of downloading changes from the online repo, to integrate them into our local copy of the repository[4];
- **Fork** - if you want to work on a repository but have no permission to modify it, you can fork it to your GitHub *account* - it will become your local copy of the parent repo; in it, you can introduce modifications, you can also synchronise your repo with its parent if needed; any changes you introduce can be pushed back to the original repository, where they will create a pull request, for review by the original repo's owners;

[4] Pulling and pushing can result in changes to be incorporated silently - if they cause no conflicts or unexpected overwriting. Otherwise, they usually generate an error with recommendations on how to resolve conflicts, or force the changes to go through regardless.

- **Clone** - it means creating your own local copy of some online repository, to which you have access (either as its owner, or contributor).

How to start using it?

Your user account

YOU NEED TO HAVE a user account[5,6] on `github.com`. Keep your username in mind as it is an important piece of information you will use linking your local projects with your online account. Exploring the webpage, or your future repos, you will notice, that many things (e.g., adding files, adding readme documents, etc.) can be done through the website. However, I do not recommend this - try to use local `git` or *GitHub Desktop* to communicate with your online repo (many apps - such as *RStudio* or *VSCode* - also have their own `git` routines communicating the app and its projects with online *GitHub* instances of each repo.

In order to keep your *GitHub* clean, try to adhere to several simple rules:

- always keep each project in a separate folder that then becomes your `git` repository;
- make very deliberate use of `.gitignore` files - its a file that can reside in your repo's main directory and it contains representations of files and/or folders that should not be tracked for changes and synchronised with your online repo;
- always accompany your repo with a readme file explaining what the repo is etc.

[5] If you happen to be emplyed by a university or other educational institution - you can apply for a free (!) *GitHub Pro* account. It has to be renewed every two years and it gives you access to many advanced functionalities of the service - among others, you have free access to the Copilot, a *GitHub* AI that can help you write better code, or just find ways of doing things much faster. See `https://github.com/education` for more info.

[6] *GitHub* account is free and public - everyone can see it. However, not all your repositories are publically visible - outside of our account, users will be able to see only repositories marked as *Public* by you. *Private* repos are visible only to their owners and people you invite to each of them as contributors.

Interacting with online GitHub

To create a repository online, just click the green *New* button (fig. 1), name your repository, decide whether it should be private or public, and setup some additional minor details. After creating it, you will be redirected to its website - its URL address is the simplest way to identify a repository.

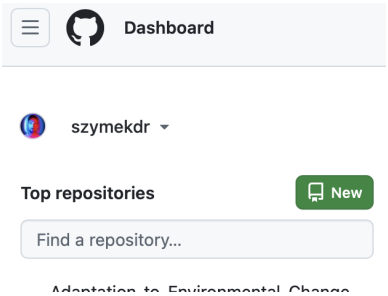

Figure 1: Creating a new repo online.

Prerequisites for RStudio and VSCode

In order to use `git` in *RStudio* or *VSCode* - you need several additional resources:

- RStudio - install the following packages: `usethis`, `gitcreds`;

- VSCode - assuming you want to use it with R: install the above packages through R, and (for a betetr experience) a few VSCode extensions (`Git Graph`, `GitHub Copilot`, `Git History`).

Creating a new repository

RStudio

`git` will only work, if you are inside of a R Project - make sure to create it in the folder you wish to turn into your new repository[7]

Once in an existing project, run the following:

```
usethis::use_git()
```

This function will setup a local repository for the active project, ask you whether you want to commit all files already existing in it (pick *Yes* if you have few small files or in generalk your project is small, otherwise I would first create and edit the `.gitignore` file to exclude the biggest files from tracking). It may also ask you to restart to display a new panel (*Git*) in the upper right panel of RStudio.

Before you can link your local repo to an online one - you need to set access permissions for RStudio. First, see if any credentials already exist in your RStudio:

```
gitcreds::gitcreds_get()
```

In my case the output looks like this:

```
<gitcreds>
  protocol: https
  host    : github.com
  username: PersonalAccessToken
  password: <-- hidden -->
```

indicating existing credentials. A message about no credentials will be generated if you have never set up *GitHub* before. In such case you will first need to generate a personal access token via `github.com`. On your profile page click your avatar, got to **Settings**, then **Developer settings**, and then **Personal access tokens (classic)**. Click **Generate new token** - there you can pick services you want this token to grant access to (if it's only for you you can tick all). You will be given with a long alfanumeric string (the token) - copy it and (if needed) write it down somwewhere (once you exit the settings page, you will not be able to display it). The copied token can then be used by running the function `gitcreds::gitcreds_set()` and providing there the token as your password.

Once your *RStudio* has access to your account, you can run the following code:

[7] You can actually create a new R project with a version control environment - check RStudio documentation how to do it.

```
usethis::use_github(private = TRUE)
```

This will create an online instance of your repo, push all existing committed changes to it, and set it private (you can also set it as public by changing the option `private` to `TRUE` - but most likely you will first want to publish your repos privately[8]).

The *Git* panel will give you access to all basic operations on your repo. You can *Pull* from the remote (download online changes), select and commit any files with local changes (they will appear in the panel), and *Push* any previously committed changes to the online repository.

[8] Any private repository can always be made public. The reverse is not possible - public repos must remain public.

Figure 2: The Git panel in RStudio with one changed, uncommitted file.

VSCode

If using *VSCode* with `R` - the way of setting up a `git` repo in a project folder, and connecting it to an online one works in the same way[9]. Pulling and pushing stuff is done via a **Refresh** button available in the **Source control** panel (can be accessed in the left bottom corner of the app window; see fig. 3). The same panel allows you to select files with changes for committing (**Stage** them), and to actually commit them.

[9] Note, that in *VSCode* you usually don't have to use `gitcreds` functions to setup access to online *GitHub* - the app is by default connected to *GitHub* (you can log in in *VSCode* to your account directly)

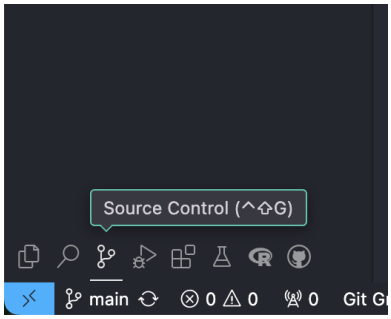

Figure 3: Location of the Source control panel.

GitHub Desktop

I don't recommend creating new repositories in *GitHub Desktop* - doing this via *RStudio* or *VSCode* is much better and ensures they are connected with the correct local folder.

Managing existing repositories in *GitHub Desktop* is very easy. First, you have to add an existing repo by using the **Add Existing Repository...** option (fig. 4).

After adding a repo, it will be available from the **Current Repository** list later, so you only have to add it once.

For active repo, the program displays files with uncommitted changes (you can actually preview the changes if needed; fig. 5), and at the bottom you can add comments and commit selected files. Once comitted, you can push changes to the remote (fig. 6) or pull things to the local repo (by clicking **Fetch origin** in the top part of the window.

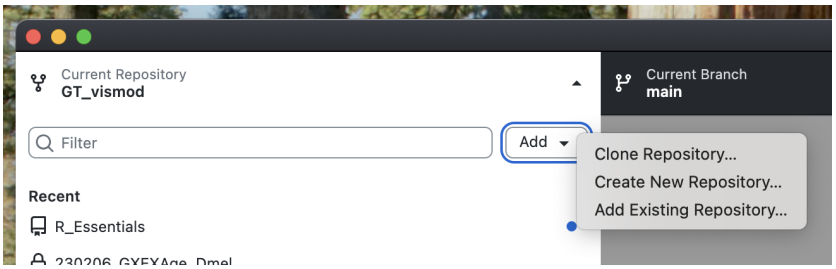

Figure 4: How to add an existing repo to GitHub Desktop?

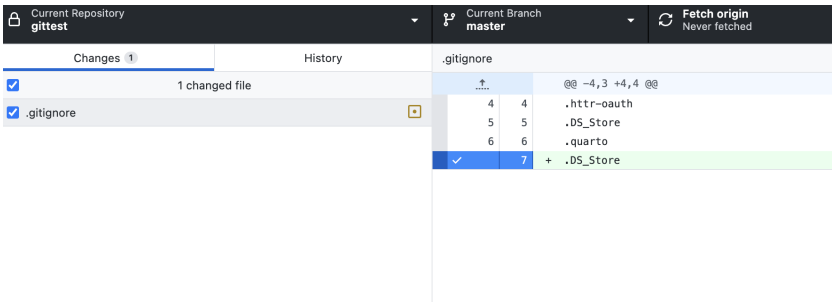

Figure 5: GitHub Desktop window with one uncommitted file.

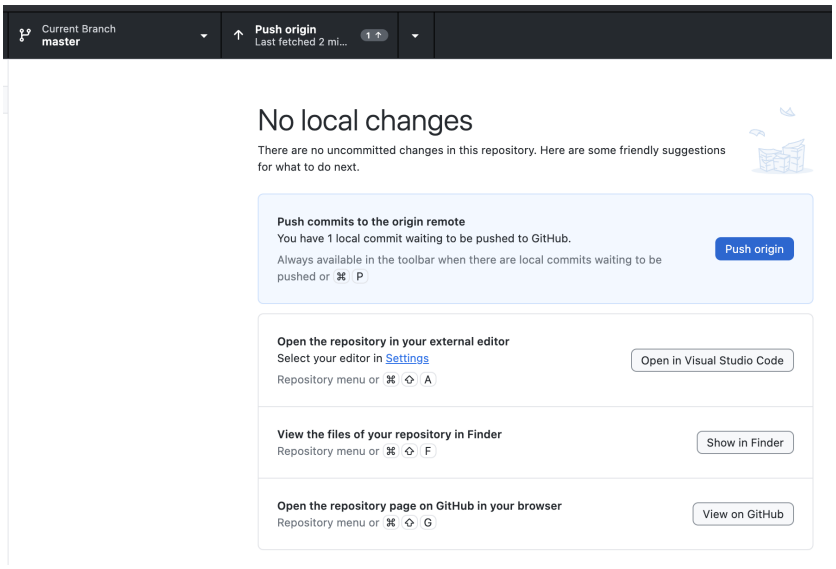

Figure 6: App window after committing all changes - an option to push stuff to the remote.

Cloning an existing online repo to your computer

Although you can do it via *Rstudio* or *VSCode* - I recommend proceeding with *GitHub Desktop*.

GitHub Desktop

In the app, navigate to the repository name and there, click **Add**, and then **Clone Repository**. In a window (fig. 7) you will be asked to select an existing repository (since you're logged in - the app will provide you with a searchable list of all repositories you own or collaborate on). Pick the one you wish to clone, select the directory to host it locally and press **clone**. The content of your online repo will be transferred to your HDD - from now on you can access it via *VSCode*, *RStudio* or *GitHub Desktop*, add things to it, commit and push/pull.

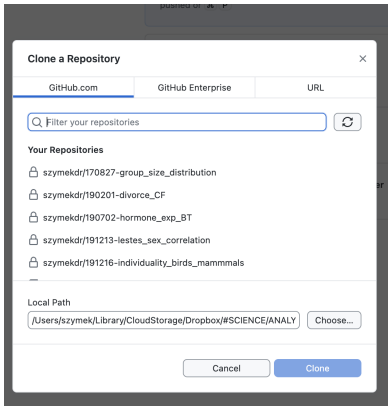

Figure 7: Repo cloning window in the desktop app

If the repo you are cloning is your private fork of someone else's repository - the changes ou commit and push will of course update your instance of the repository, not the one from which it was forked. However, after pushing something to such a repo - you can navigate there online; the online system will alert you that your repo is *ahead* of the original one by one or more commits (it may also be *behind* - in which case you can be given an option to sync your fork with its parent). If you're indeed ahead of the parent one - you can send your changes as a pull request to the parent one, where its admins can review it and integrate into the repo's content.

Our case: uploading data to the repository

You are most likely a contributor to the Nestbox Experiment repository. This gives you several options to interact with it, and in particular - to deposit your data and other files into it.

1. *You have cloned the repo to your HDD*. In such case, before you proceed with any update, pull from the remote repository to make sure your local copy is in sync with the online one. Then place your fails in a selected location within the repository - e.g., in you study site's folder, or wherever was agreed among the contributors. Having done that, you should see the new/changed files as uncommitted changes in the Git panel of your app (or in *GitHub Desktop*). Just commit the changes and push them up to the remote repository. All done!

2. *You don't have a local copy of the online repository.* If so, you can either follow the instructions in the section above to first clone

everything to your HDD - or do everything online. In the latter case: just navigate to the repository's main page on `github.com`, go to the desired folder and pick **Upload files** (fig. 8). Remember to avoid uploading files exceeding hundreds of megabytes.

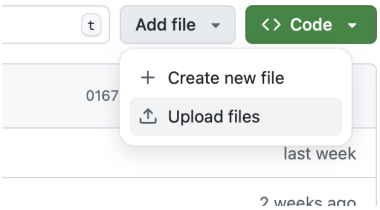

Figure 8: Uploading files online

Issues

The online *GitHub* service provides a number of tools that make collaboration easier. One of them is the system of *Issues*. You can access them in your repo via the top menu bar (fig. 9).

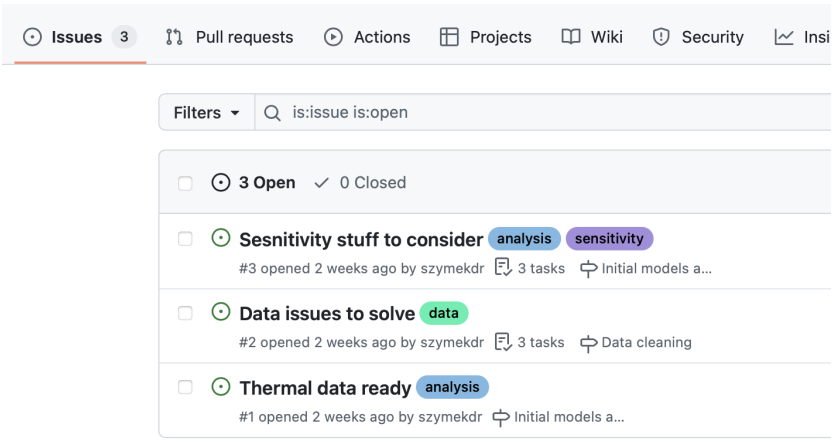

Figure 9: GitHub Issues

The *Issues* webpage is in fact a simple chat and discussion forum. You can create action topics there, you can even ping relevant contributors within each issue. It is a good practice to use this system to organise work, provide updates and communicate your progress on a task (and - of course - to close an issue once solved).