Let's collaborate (GitHub)!

Daniel Heimgartner

June 12, 2025

Code to be executed in a terminal running bash looks like this

```
echo "Let's collaborate!"
```

1 WHY?

What is GitHub

From Wikipedia (yes, it's ok to cite from there – at least, once you have defended): "GitHub [...] is a proprietary developer platform that allows developers to create, store, manage, and share their code. It uses Git to provide distributed version control and GitHub itself provides access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project. Headquartered in California, GitHub, Inc. has been a subsidiary of Microsoft since 2018."

- "It uses Git": The key underlying technology is git and we use git to "interact" with GitHub.
- GitHub is a webapp that has some very nice features (as mentioned above).
- For us in particular important are the features that allow us to inspect code and changes thereof, and discuss code (e.g., using the issue tracker or a pull request).

The main takeaway: GitHub hosts our source code and allows us to easily inspect, update and share code.

Are there alternatives?

- As with any proprietary software/product, one wonders: Where does my data live? Who has access? What if they suddenly stop the service (or make you pay for an initially "free" service)? Etc.
- The key feature (for us) is, that GitHub hosts our remote branch (more on this later) which allows us to sync and share our work. However, if we only collaborate internally, you could use any shared network location and add the address as remote `git remote add origin <url>`.
- Since the above raised questions matter, ETH Zurich hosts its own GitLab service. GitLab is very similar to GitHub and you will be able to easily transfer what you learn here.
- Bitbucket is another alternative... There are probably many more...

But GitHub is by far the most dominant player – quite a smart move from Microsoft to have acquired it: They now massively benefit from having billions and billions of lines of source code available which they can use to train their AI tools like Copilot...

What will I learn?

- Understand difference between local branch and remote branch (i.e., working locally and then publishing the changes remotely).
- Initialize a new (R) project and make it available to others on GitHub.
- Make a local copy of a GitHub project (clone a repository, i.e., easily share code; download files).
- Work on a GitHub project yourself (pull and push).
- Work on a GitHub project together (work on a new branch and then integrate the branch - i.e., make and discuss a pull request).
- (Understand the need of forks - i.e., if you want to contribute to a project you don't have "write" permission).

2 SETUP

ssh is a (cryptographic) network protocol – i.e., a "communication tool" to securely exchange information via an (unsecured) network. The alternative is to use HTTPS (just another transfer protocol) – however, it is recommended to use ssh.

The URL depends on which protocol you use (more on this later)... From `git clone --help` we learn that git supports further protocols than the ones mentioned.

The first thing you should do (this requires you to understand how to use a terminal – luckily we do!)

- Check for existing SSH key `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/checking-for-existing-ssh-keys`.
- Generate a new SSH key `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent`
- Add a new SSH key `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account`.
- Test your SSH connection `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/testing-your-ssh-connection`.

And then always use the ssh protocol (i.e., the appropriate URL) from now on.

ssh can also be used to log into a server and exchange information (e.g., files) with it...

3 LET'S COLLABORATE

There are two ways you can initialize a GitHub project: 1. Intialize a new repository/project *remotely* on GitHub and then *clone* it to your *local* computer or 2. *Push* a local git-repository to GitHub. I think the latter is more natural, as I usually first work locally and then at some point decide to push it to GitHub (either because I want to make it publicly available or I want to collaborate – possibly with myself from another computer). So I will introduce 2. (but you'll easily figure 1. out if you have to).

Push a local git-repository to GitHub

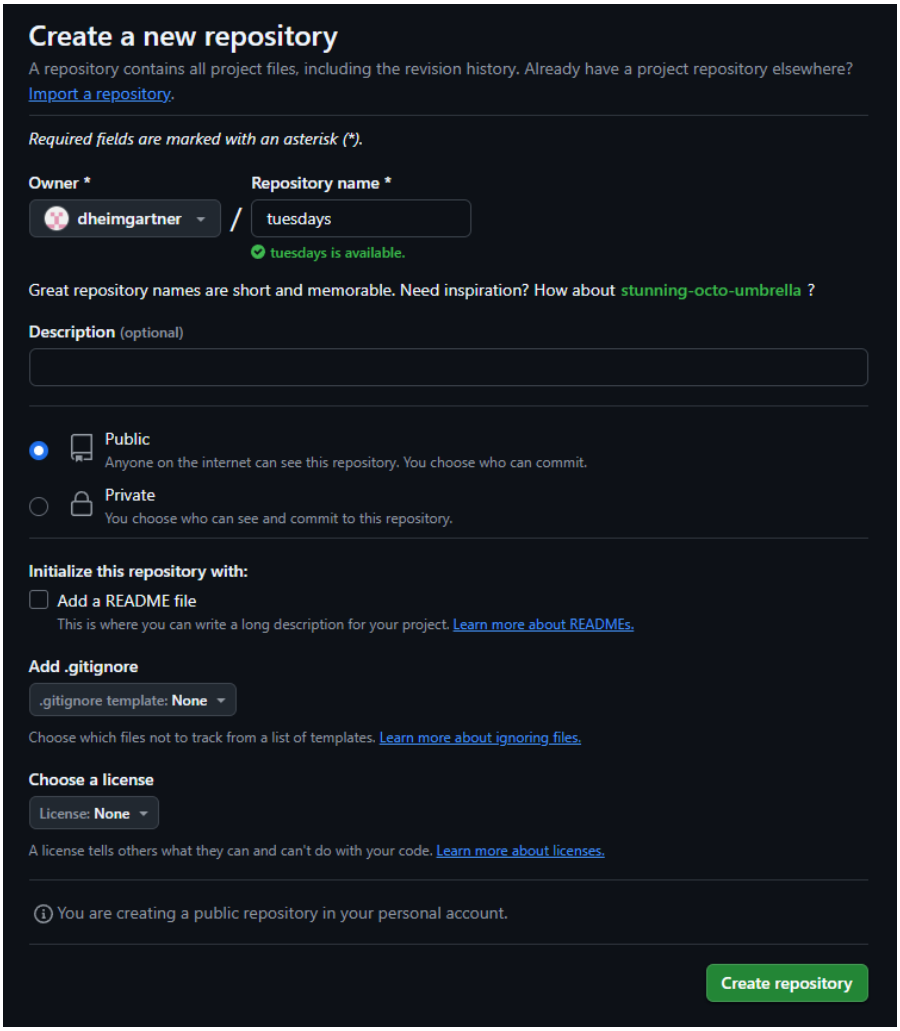
Let's first create a new R-package/project and make it a git-repository (nothing new here, we already know these steps...)

You might enjoy radian – a 21 centuryRconsole `https://github.com/randy3k/radian`.

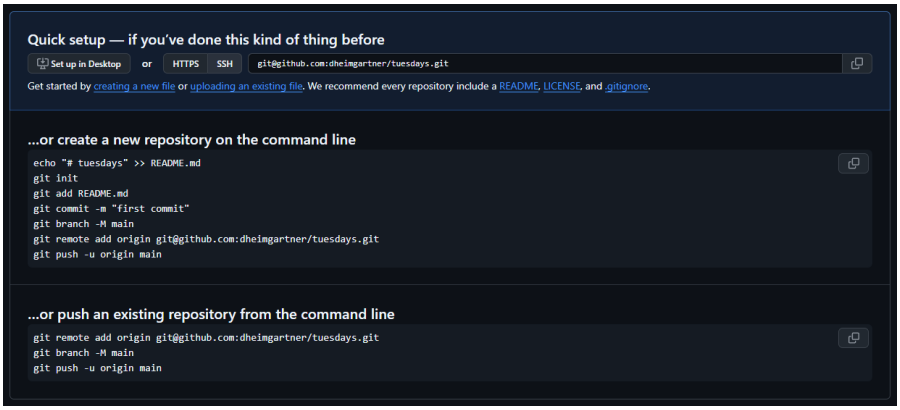
- Start the R console from within a terminal session (first navigate to your desired location... Hint: `cd`)
- `usethis::create_package("tuesdays",rstudio = TRUE)`
- `cd` into the directory: `cd tuesdays`
- `usethis::use_git()` (more or less the same as `git init . && git add . && git commit -m "init"`

Now we have to create a new empty "tuesdays" repository on GitHub and populate it with the files and stuff we just scaffolded.

- Navigate to `https://github.com/new` (or find and click the green "New" button – don't worry, it's ok to click!)
- Populate the repository name and click "Create repository"



This then brings you to the following page



- As outlined above, it suggests us the two options and we demonstrate the latter here (i.e., push an existing repository from the command line).
- Importantly, click on "SSH" in *Quick setup* which will adjust the URLs in the commands that follow to use the ssh protocol.
- Copy and paste the code into your terminal session (obviously, the current directory needs to be the "tuesdays" repository!)

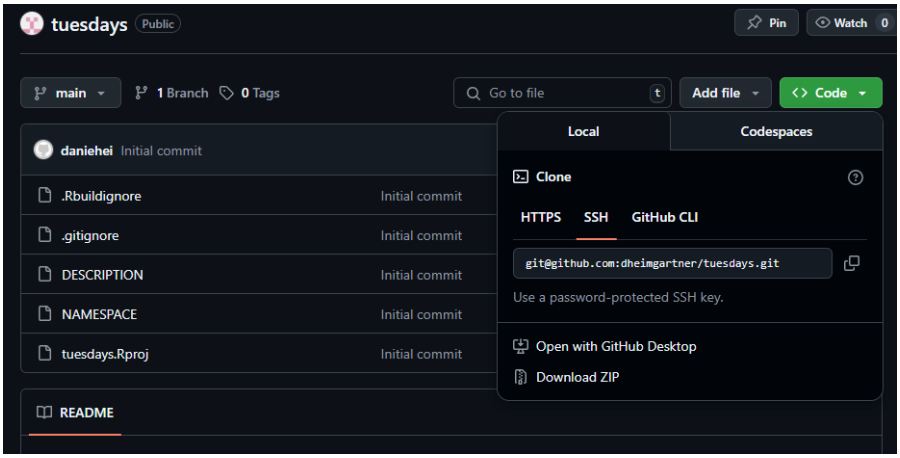
```
## add the remote branch
git remote add origin git@github.com:dheimgartner/tuesdays.git
## rename (move) the branch from master to main
git branch -M main
## push the local main branch to the remote origin => origin/mai
git push -u origin main
```

- Refresh the page to see the magic – your remote repository (living on some Microsoft server somewhere) now is an exact copy of your local git-repository (unless you have excluded files via the `.gitignore` file).

Now imagine, that your beloved cat (if you are a cat person) or dog (if you are a dog person) destroyed your computer... The cat probably pushed it over an edge, while the dog ate it. No worries – just *clone* the repository

```
## navigate one level up
cd ..
## simulate the cat or dog
rm -rf tuesdays
## clone again
git clone git@github.com:dheimgartner/tuesdays.git
## cd into the freshly cloned tuesdays repo
cd tuesdays
## forgive your cat or dog
```

You find the URL by clicking on "Code" when having navigated to a GitHub repository (e.g., `https://github.com/dheimgartner/tuesdays`). Again, don't forget to click on SSH.



git pull

I didn't find a better place to tell you about pulling remote changes. But if you expect that the remote branch has updated, you can integrate the changes locally by *pulling* them (down from remote) `git pull`.

If you are the sole developer

Now, the only thing that changes (compared to working locally only), is, that you have to *push* the local changes to remote from time to time (when exactly is up to you). So the workflow looks very similar to what we already know

This is also fine if you are few developers with write permission and you coordinate your work...

```
## work on the project locally
## add the changes
git add .
## commit
git commit -m "blob"
## push it to remote (not necessarily after every commit)
git push
```

If multiple people contribute to the same project

Here is what we will do then: Create a local development branch and work on this branch. Push the development branch and open a pull request. Inspect and discuss (if necessary) the changes and merge the pull request. Delete the remote development branch. Pull the updated main branch again (to update the local copy). Let's do it!

```
## create local development branch
git branch dev
## switch the branch from main to dev
git checkout dev
## work on it and add commits
git add . && git commit -m "blob"
## try to push it
git push
```

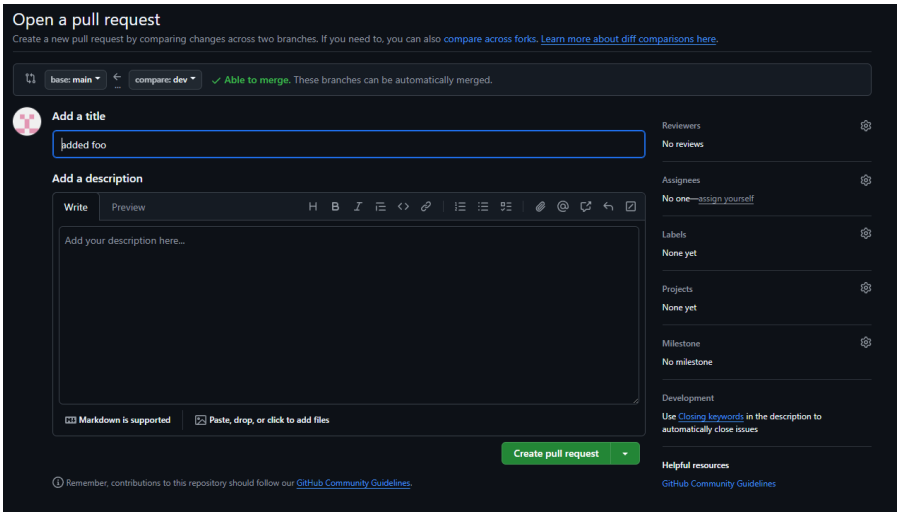
This will complain and tell you what to do

```
fatal: The current branch dev has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin dev
```

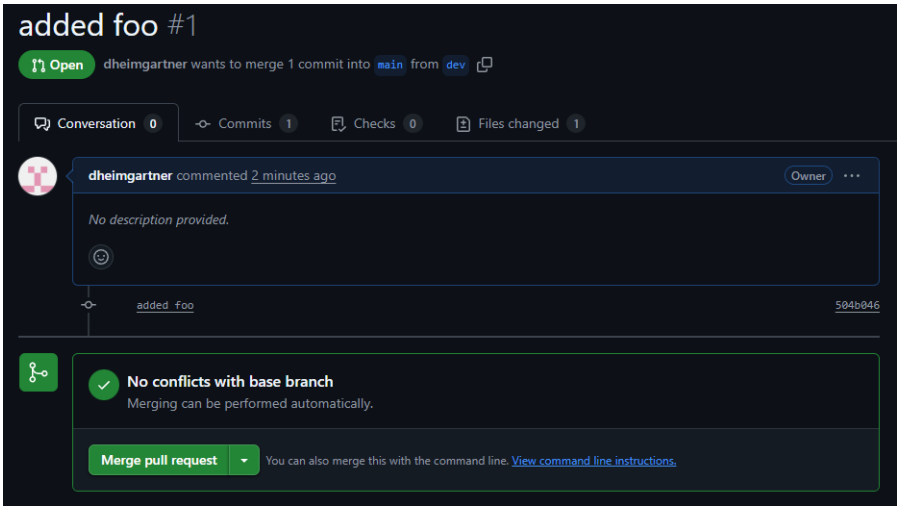
So let's do it (it's always a good idea to reason why something did not work and try to understand what the suggestion does)

```
git push --set-upstream origin dev

Enumerating objects: 4, done.
Counting objects: 100\% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100\% (2/2), done.
Writing objects: 100\% (3/3), 271 bytes | 271.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100\% (1/1), completed with 1 local ob ect.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/dheimgartner/tuesdays/pull/new/d v
remote:
To github.com:dheimgartner/tuesdays.git
 * [new branch]      dev -> dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
```

Again, read the command's output – it already suggests to "Create a pull request for 'dev' on GitHub by visiting: `https://github.com/dheimgartner/tuesdays/pull/new/dev`". So again, let's do it!



Discussing and closing a pull request should be self-explanatory: Just navigate to the "Pull requests" tab (e.g., something like `https://github.com/dheimgartner/tuesdays/pull/1`) and use the webapp/GUI. Merge the pull request – It's always safe to click on green buttons, right?



4 NOW IT'S YOUR TURN

Again, this by far does not cover all the edge cases and it is essential that you just practice (and encounter such edge cases/problems yourself and resolve them). However, for this exercise, all the steps are outlined above!

1. Create a new R-package.
2. Push it to GitHub.
3. Add one or multiple collaborators (you'll figure it out).
4. Create a pull request (with hopefully a very nice R-function or new dataset: see also `https://github.com/lets-use-xxx/Rpackage/blob/main/vignette.pdf`).

Advanced

Try to simulate a merge conflict and resolve it... Simulating a merge conflict is actually a very good exercise – it requires you to understand why they emerge in the first place!

5 FINAL REMARKS

This concludes the *let's use xxx* series – and they lived happily ever after, with complete mental peace, always knowing that their work is version controlled, securely stored on some remote GitHub server.

Resources

- Foo