GITHUB FOR WRITING PAPERS TOGETHER

It can make sense for co-authors to have a GitHub repository and project board together. It requires a bit of work at the beginning, but once it is set up, it is very easy and great. It might seem complicated at first, but one can get used to it quickly.

GitHub is an online platform that let's you save and share your files with your co-authors. It let's you save files in a way that allows you to restore any earlier version of the file easily. No former version of your paper will ever get lost. The versions get saved in the structure of a directed, acyclic graph, which gives you a great overview and many possibilities to work on a project simultaneously without any conflicts between files. If you make changes to files, Github let's you see the previous and current version of the file side by side and marks which lines have been edited. It is very easy to compare. It is possible for everyone in the project to add comments to each change or line.

Everyone has a local copy of the files on their laptop and can push changes they make to the files onto Github and can pull changes made by others from Github onto their own laptop.

Working together on a project on Github is great, because you have an amazing overview and if one person messes up, it is so easy to restore an earlier version. If you get confused, you can easily look up, what was written before. All drafts are automatically saved and can't get lost.

Github also allows you to create project boards. You can collect all the things that are left to do in the board. For example you could have a list there of all the lemmata that you still need to prove or all the ideas for future papers you come up with during your work on the project. It is a great tool for keeping organized.

Since setting it all up is the hardest part, I created this small pdf for help. I am not a senior software engineer and I am not deep into the GitHub universe. I am just trying to write mathematical papers with other people and managed to make it work for us. If you dislike how I described how to set everything up or if you follow these instruction and you get errors, please don't hate me.

If you experience difficulties, here are some good places to read:
`https://docs.github.com/en`
`https://git-scm.com/doc`

Kind regards to you
– Sira

Contents

For the sake of examples, we will assume the following in this whole document:

Your name is: Chiara Crunchcrystal
Your email address is: chiara.crunchcrystal@example.com
Your username will be: ChiaraCrunchcrystal

1. Get onto GitHub

Go to `https://nerdstagram.com/` and create a new account on that website. Click Sign up. Follow the prompts to create your personal account. (More information: `https://docs.github.com/en/get-started/quickstart/creating-an-account-on-github`)

2. Setting up git

2.1. **Installing git.**

2.1.1. *Linux debian flavored (Ubuntu, Mint).*

```
sudo apt-get install git
```

(More information: `https://git-scm.com/download/linux` or `https://docs.github.com/en/get-started/getting-started-with-git/set-up-git`)

2.1.2. *Mac.*

(1) Is Homebrew installed? If not, then install Homebrew first by opening a terminal and typing:
```
/bin/bash -c "\$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```
(More information: `https://brew.sh/`)

(2) Install git:
```
brew install git
```
(More information: `https://git-scm.com/download/mac` or `https://docs.github.com/en/get-started/getting-started-with-git/set-up-git`)

2.2. **Configure git.**

(1) Set git username for every repository on your computer:

```
git config --global user.name "Chiara Crunchcrystal"
```

Check if it worked by typing:

```
git config --global user.name
```

If you get:

```
$ git config --global user.name
> Chiara Crunchcrystal
```

then it worked.
(More information `https://docs.github.com/en/get-started/getting-started-with-git/setting-your-username-in-git`)

(2) Set commit email address:

```
git config --global user.email "chiara.crunchcrystal@example.com"
```

Check if it worked by typing:

```
git config --global user.email
```

If you get:

```
$ git config --global user.email
> chiara.crunchcrystal@example.com
```

then it worked.
(More information:
`https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-a`
`managing-email-preferences/setting-your-commit-email-address?platform=mac`)

(3) Add the email address to your account on GitHub, if it is not already there.

2.3. **SSH key.**

(1) Do you already have a ssh key? Find out:

```
ls -al ~/.ssh
```

Suppose yes. Then you can see a

```
[something].pub
```

(just something ending with .pub) in the list in the terminal. Type:

```
cat ~/.ssh/name.pub
```

Copy what you get.

(2) If you don't already have a ssh key:

```
ssh-keygen -t rsa
```

Press ENTER to accept the default location. Then the ssh-keygen utility prompts you for a passphrase. Type in a passphrase or hit ENTER to accept the default (no passphrase). Then either repeat the passphrase again or press enter again for confirmation. Now you can:

```
cat ~/.ssh/id\_rsa.pub
```

Copy what you get.

(3) Go on GitHub → Settings → SSH and GPG keys → New SSH key → paste

(More information: `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent`)

3. Create a new repository

Choose a location on your laptop for your repository. For example:

```
~/Documents
```

Go to that location by typing into the terminal:

```
cd ~/Documents
```

Think of a good folder name for your repository. Let's say you choose *CrunchcrystalRepo*. Create a new folder with that name:

```
mkdir CrunchcrystalRepo
```

Go into that folder an initiate a new git repository:

```
cd CrunchcrystalRepo
git init
```

(More information: `https://docs.github.com/en/get-started/using-git/about-git`)

4. Add something to your repository and push it to GitHub

(1) First you have to create a new file in your *CrunchcrystalRepo*-Folder. If you just want to try something out, you could do:

```
touch test.txt
nano test.txt
```

then write anything. Then: X → Y → ENTER.

(2) Now you want to stage that change. Type:

```
git add test.txt
```

Or:

```
git add .
```

The later option just adds all files in that folder.

(3) Now commit your changes. You can either do:

```
git commit -a --allow-empty-message -m ' '
```

and be done or you can do:

```
git commit
```

and then you have to type a little commit message and then: X → Y → ENTER.
(It allows you to give a bit of context and explain the changes you have made.)

(4) Now go on the GitHub webpage and create a new repository with the name *CrunchcrystalRepo*. Just follow the prompts.

(5) Type in the terminal:

```
git remote add origin git@github.com:ChiaraCrunchcrystal/CrunchcrystalRepo.git
```

(6) Push your Repo with your new file onto GitHub:[1]

```
git push origin -u main
```

If you now go on the GitHub webpage and are logged in, you should be able to see your new file *test.txt* in the *CrunchcrystalRepo* there.

[1] If this does not work, then type 'git branch -m master main' and try again.

5. Become or invite a collaborator of an already existing repository

(1) The person with the admin rights to that repository has to go into that repository on GitHub under *Settings* and under *Access* click *Collaborators*. There the admin has to invite a person as a collaborator. Then that person can accept that invitation.

(2) If you got invited and accepted: Determine a good location for the local copy of that repository on your laptop and go there with the cd command. On GitHub, go to that repository. There is a green rectangle with the text <> *Code* that you can click. Do that. Suppose this repository has the name *Chiara-Kevin* and was created by *kevinnimmersatt*. Then, under *SSH*, you should now be able to see something like:

`git@github.com:kevinnimmersatt/Chiara-Kevin.git`

Type the following in your terminal:

`git clone git@github.com:kevinnimmersatt/Chiara-Kevin.git`

Now you should have a local copy of that repository on your laptop.

6. Beeing a collaborator

(1) If you have an open terminal and you are in the folder with the repository in which you are working with your co-authors and if you type

`git branch`

and if the terminal gives you a list in which *main* is marked, then do not make any changes and do not add or commit or push anything. Except you all agreed that that would be okay to do. But even in that case: Type

`git pull`

first in order to make sure you have the most up-to-date version of that repository as your local copy and make sure no one else is making any changes simultaneously in order to avoid conflicts, chaos and drama.

(2) Git allows you to safe all versions of your repository in a directed, acyclic graph structure. So all versions of your repository get ordered in a directed, acyclic graph (DAG). When you start out with a repository, there only exists the main branch. Your graph is just a line and every time you commit something, another point gets added to that line.

If you work together on a project with multiple people and some people make changes at the same time, that can lead to conflicts between the files. To avoid that, every person can create their own working branch. For example if two people Chiara and Kevin have a project together, there could be the main branch, a branch called *Chiara* and a branch called *Kevin*. If Chiara makes changes in the Chiara branch and Kevin makes changes in the Kevin branch, there won't be any conflicts, if they push their changes onto their branch to GitHub. If a person is happy with their changes, this person can make a pull request on the GitHub webpage. If the other person is also happy with these changes, the pull request can get accepted and then the working branch of that person gets merged into the main branch, meaning: The main branch will get updated with these changes. Then everyone on their laptop has to checkout the main branch and pull the latest version in order to update their local copy.

If you want to make changes to the repository, the safest way is to first do:

`git pull`

And then creating a new branch for you to work on. You do it like this:

`git checkout -b branchname`

If you now type:

`git branch`

you should see your new branch and the main branch in the list that appears in the terminal. You can switch branches by typing:

`git checkout branchname`

If you want to delete a branch again, you can do:

`git branch -d branchname`

If you are sure you are on your working branch, you can start to make changes.

7. Overview over some git commands

Git command	What it does
git init	Initiates a new repository.
git add	Stages changes.
git commit	Saves everything that has been staged into a new commit.
git branch	Tells you the branch that you are on.
git checkout branchname	Let's you switch to the branch branchname.
git checkout -b branchname	Creates a new branch with the name branchname and switches to that branch.
git branch -d branchname	Deletes the branch with the name branchname.
git log	Shows you the commit logs. (More information: `https://git-scm.com/docs/git-log`)
git push origin -u branchname	Pushes your local changes onto the branch branchname on GitHub.
git pull repositoryname	Builds in the changes of a different repository called repositoryname into the curent branch.
git rebase	Take all the changes that were committed on one branch and replay them on a different branch. (This is used in case of merge conflicts. See: `https://git-scm.com/book/en/v2/Git-Branching-Rebasing`
git status	Let's you see whether or not you are up-to-date (`https://git-scm.com/docs/git-status`)

8. Add a .gitignore

If you want to work on tex files with your co-authors, but don't want all the .aux, .log, .out, .gz and .pdf files in your repository on GitHub, you can add a .gitignore file to your repo, which makes that all files ending with .aux, .log, .out, .gz or .pdf get ignored if you add, commit and push.

Go to your repo in the terminal. Type:

```
touch .gitignore
nano .gitignore
```

Add the following text:

```
# ignore generated aux, log, out, gz and pdf files,
**/*.aux
**/*.log
**/*.out
**/*.gz
**/*.pdf
```

And now:

```
git add .gitignore
```

Commit and push your changes.

If you already have files in your repo on GitHub that you don't want there, you can

```
git rm -f filename
```

9. Pdf-Builder for LaTeX files

If you want to make sure your tex files can be compiled globally and not just on your own laptop with your specific settings and you would like to have global pdf files and you would not like your repository to be filled with random pdfs, then you can add something to your repo that automatically creates pdf-files from all .tex-files in the repository and uploads them as *pdf-files.zip*. Clone the *latex-pipeline* repo to a good place on your laptop. Not in the repository you want to use it in.

```
git clone https://github.com/jakdimi/latex-pipeline.git
```

Copy the folder *.github* from there into your repository. Then add and commit the changes and push them to GitHub. Create a new tag (it has to start with a *v*):

```
git tag v0.0.1
```

Push it and create a new release:

```
git push origin --tags
```

Now you have to wait for the build and see if it fails or not. You can see protocols and all that on GitHub in your repository under *Actions*. If the build was successful, you can go to your repo on GitHub and on the right side under *Releases* you can click on the latest release and find the *pdf-files.zip* there.

Remark: When I first tried this, I ran into some trouble because of my image paths. I had several folders and subfolders in my repository. I had the images I used in tex files in the same folder as the tex files and just referred to them by their name. Locally I could compile my tex files, but the builds failed. I had to change the reference to the images to the path of the image in that repository. For example: Suppose I have an image *x.jpg* that I want to use in a tex file *X.tex*. Both *x.jpg* and *X.tex* are in a folder called *c* which is in a folder called *b* which is in a folder called *a* in the repository. I had to refer to *x.jpg* in *X.tex* like this:

```
\includegraphics[width=1\textwidth]{a/b/c/x}
```

10. Some useful terminal commands

Command	What it does
cd foldername	Go to the folder called foldername (cd for change directory).
cd ..	Moves up one folder.
ls	Shows you a list of all files.
ls foldername	Shows you a list of all files in the folder foldername.
mkdir foldername	Creates a new folder foldername.
.\file	Opens the file that is here and is called file.
cat filename	Tells you the content of a file (cat for concatenate).
nano filename	Opens the file filename and let's you edit it. It's a text editor. Some people prefer vim. I can never escape vim.
rm filename	Deletes the file named filename (rm for remove).
mv filename foldename	Moves the file filename into the folder foldername.
cp folder1/file.txt folder2	Copies the file file.txt from folder1 to folder2.
grep -R 'string' dir/	Searches for a string string in a folder dir/.

11. Create a project board

On GitHub go to Projects. In the green rectangle click on the downwards arrow. Choose New Project. Click New Project. Create project board. Edit it however you want. Next to the project board name on top on the right there are three little dots. Click them. Go to Settings. Manage access. Invite collaborator.